The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

An Embedded System Artefact Organisation and Adaptation Knowledge Management System for Embedded System Product Prototyping

A thesis submitted to the Department of Electrical Engineering,
University of Cape Town, in fulfilment of the requirements
for the degree of

Doctor of Philosophy

at the

UNIVERSITY OF CAPE TOWN

by

Simon L Winberg

BSc (CS), BSc (Hons) (CS) (University of Cape Town), MSc (CS) (University of Tennessee)

Supervisors:

Professor S. Schach (Vanderbilt University)
Professor M. Inggs (University of Cape Town)
Doctor M. Linck (University of Cape Town)



University of Cape Town
December 2010

University of Cape Town

Declaration

I, Simon L. Winberg, declare that the contents of this thesis represent my own unaided work, and that this thesis has not previously been submitted for academic examination towards any degree in any other university.

Signature of Author . . . . . Signed by candidate . . . . . .

Department of Electrical Engineering
Cape Town, December 2010.

To my parents

Thanks for your patience, motivation and generous support

University of Cape Town

Acknowledgements

I would like most sincerely to thank Professor Stephen Schach for his continued support, excellent feedback, guidance, and constructive approach to research supervision. Many thanks to Professor Michael Inggs for his constant support, many considered opinions, and assistance in acquiring resources, organising project teams and connecting me with collaborators in the field. Thanks to Dr Michael Linck, particularly during the early parts of this project while the thesis was planned, research strategies considered and project teams structured, as well as for his part in evaluating project teams.

Thanks to Professor Martin Braae, Head of the Department of Electrical Engineering in 2005, and Professor Trevor Gaunt, Head of Department in 2006, for the provision of equipment, laboratory space, and funding for additional hardware. I am grateful to the UCT and the UCT Postgraduate Funding Office for university research scholarships.

My thanks to Dr Alan Langman (of the KAT / SKA Project) for his collaboration in organising and evaluating the project teams. Thanks to Peter Golda and Allen Wallis (both of the KAT / SKA Project) for their input and willingness to listen to my ideas. Thanks to David George, Nico Gevers, Thomas Davies, Kalen Watermeyer and Richard Lord for their feedback and testing of some of my tools and ideas. Thanks to Regine Lord for her assistance in administrative issues. I also express gratitude to the many members of the Radar and Remote Sensing Research Group for discussions about my thesis, for providing an enjoyable and collegial environment, and their general involvement and interest in my work.

University of Cape Town

Abstract

This thesis presents an innovative approach to knowledge management (KM) from the perspective of embedded system (ES) development, a form of development that is highly knowledge intensive and depends on specialised forms of knowledge obtained from a variety of complex knowledge artefacts. This study follows an experimental methodology that involves integrating a knowledge management system (KMS) into ES product prototyping projects, in order to facilitate KM of a specific form of knowledge, namely embedded system artefact organisation and adaptation (ESAOA) knowledge. ESAOA knowledge is produced during ESAOA activities, which concern organising artefacts that are used to construct an ES and techniques by which engineers adapt and learn from these artefacts. The focus of this thesis is narrowed to determining an effective structure for the ESAOA KMS that facilitates successful completion of ES implementation tasks. This thesis consequently contributes to KM research at a *meso level* of operations.

The research methodology involved constructing an experimental KMS, named the ESAOA KMS, which comprises a structured collection of knowledge worker roles, processes, and artefacts together with a collection of support tools. A pilot study was first performed to gain insights into research methods and the KM needs of the users. These research methods were published in order to improve them further and to confirm their validity. Next, an initial version of the ESAOA KMS was built. This KMS was applied by development teams in the context of ES prototyping projects. The data obtained from this experiment were evaluated to develop a refined version of the ESAOA KMS, and to draw conclusions for this research.

Findings from this research included the following: defining different forms of ESAOA knowledge; establishing evaluation methods for KM of ESAOA activities; identifying conditions that enable a KMS to facilitate ESAOA activities; assessing the factors that affect ESAOA KM activities; determining different types of KM needs that occurred in projects, and showing that the ESAOA workspace approach was an effective means to integrate the knowledge worker roles, processes, artefacts and support tools of the ESAOA KMS.

University of Cape Town

The conclusion of this thesis identifies situations in which the ESAOA KMS was found to be beneficial, as well as conditions where the KMS was of little use or possibly added to the difficulty of completing ESAOA activities. Generally, for the projects investigated in this study, the ESAOA KMS was of the least benefit to users during *simple* activities (a term defined in the thesis, which essentially relates to tasks where the needed knowledge was obtained in a trivial manner or produced by following easily remembered or routine procedures). However, users working on *complex* activities (which are activities that draw on knowledge obtained previously in the project through prior non-trivial procedures) made extensive use of the ESAOA KMS. In such situations, the ESAOA KMS was shown to provide benefit to these complex activities. In particular, the ESAOA workspaces improved conformity of artefact classification and location, and assignment of the KMS roles made it easier for team members to assign responsibilities, to divide knowledge work among each other, and to guide knowledge production.

Further research plans that follow on from this thesis include broadening the scope of the ESAOA KMS to support additional phases of the development lifecycle (e.g., the requirements phase), conducting a study focused on KM for ES innovation, and establishing a method for incrementally phasing the ESAOA KMS into longer-term on-going projects.

University of Cape Town

List of Acronyms

ADC	Analogue to digital converter
ANTCON	Antenna controller
AO	Artefact organisation
AOA	Artefact organisation and adaptation
AOD	Artefact organisation drawing
API	Application Programming Interface
ARM	Advanced RISC Machine
CASE	Computer-aided software engineering
CKO	Chief knowledge officer
CKS	Communal knowledge steward
CR	Component researcher
DK	Data knowledge
DS	Data steward
ES	Embedded system
ESAOA	Embedded system artefact organisation and adaptation
FA	Framework analysis
FC	Framework construction
GCC	GNU computer collection
GPRS	General packet radio service
GPS	Global positioning system
GUI	Graphic user interface
HDL	High-level description language
HTML	Hypertext mark-up language
IC	Integrated circuit

University of Cape Town

List of Acronyms

ADC	Analogue to digital converter
ANTCON	Antenna controller
AO	Artefact organisation
AOA	Artefact organisation and adaptation
AOD	Artefact organisation drawing
API	Application Programming Interface
ARM	Advanced RISC Machine
CASE	Computer-aided software engineering
CKO	Chief knowledge officer
CKS	Communal knowledge steward
CR	Component researcher
DK	Data knowledge
DS	Data steward
ES	Embedded system
ESAOA	Embedded system artefact organisation and adaptation
FA	Framework analysis
FC	Framework construction
GCC	GNU computer collection
GPRS	General packet radio service
GPS	Global positioning system
GUI	Graphic user interface
HDL	High-level description language
HTML	Hypertext mark-up language
IC	Integrated circuit

University of Cape Town

IDE	Integrated development environment
IE	Innovation engineer
IK	Innovation knowledge
IPC	Inter-process communication
KB	Knowledge base
KET	Knowledge event type
KIT	Kit for Information Technology
KM	Knowledge management
KMS	Knowledge management system
NPK	Non-productive knowledge
PC	Personal computer
PE	Process engineer
PK	Productive knowledge
PK	Process knowledge
PTHC	Partitioned Time History Calculator
SoSiG	Software Signal Generator
TL	Team leader
TM	Team member
UDP	User datagram protocol
UML	Unified modelling language
URL	Uniform Resource Locator
VAS	Voice Activation System
VoIP	Voice of Internet Protocol
VUT	Vehicle Usage Tracker
WA	Workspace administrator

University of Cape Town

Contents

Declaration i
Acknowledgements iii
Abstract iv
List of Acronyms vi
Contents viii
List of Figures xvii
List of Tables xx
Chapter 1: Introduction 1-1
1.1 Terminology and definitions 1-1
1.1.1 Embedded systems and embedded software development 1-2
1.1.2 ES products and prototypes 1-2
1.1.3 Computer engineering and embedded engineers 1-2
1.1.4 Tasks and activities 1-3
1.1.5 Implementation tasks 1-4
1.1.6 Implementation artefacts and soft/hard artefact classification 1-5
1.1.7 ESAOA activities 1-6
1.1.8 Data, Information and Knowledge 1-7
1.1.9 Knowledge management 1-8
1.1.10 Knowledge management systems 1-9
1.2 Rationale: a KMS for ESAOA activities 1-9
1.2.1 The growing demand for embedded software 1-10
1.2.2 The rapid expansion of knowledge 1-10
1.2.3 Knowledge management as a potential means to facilitate embedded software development 1-11
1.2.4 The need for an ESAOA KMS 1-12
1.3 Thesis Objective 1-13
1.4 Problem Statement 1-14
1.5 Focus 1-15
1.5.1 Focusing on new product development 1-15
1.5.2 Focusing on product prototyping during a proof-of-concept 1-16
1.5.3 Focusing on ESAOA activities related to component integration 1-16
1.6 Delimitations 1-17
1.6.1 Task-oriented ESAOA activities 1-18
1.6.2 Level of developers 1-19
1.6.3 Team size and composition 1-19
1.6.4 Number of experiments and case studies 1-19
1.6.5 Time-frame for case studies 1-20
1.6.6 Products developed 1-20
1.7 Thesis Structure 1-20
1.8 Summary 1-21
Chapter 2: Literature review: Knowledge management and embedded system engineering 2-1
2.1 Methodology of the literature review 2-1
2.2 The ES development process 2-4
2.3 Inefficiencies of ES development 2-7
2.3.1 General software engineering difficulties 2-7
2.3.2 Complex and lengthy learning processes 2-7
2.3.3 The value and temporality of intellectual capital 2-8
2.3.4 Decentralised development, speed of obsolescence and availability of new technology 2-8
2.3.5 Embedded software maintenance issues 2-8

University of Cape Town

2.4 Knowledge Management Terminology 2-9
2.4.1 The Data, Information and Knowledge (D-I-K) Hierarchy 2-9
2.4.1.1 Data, information and knowledge scenario 2-10
2.4.1.2 Knowledge acquisition and limitations of the D-I-K hierarchy 2-11
2.4.1.3 Tacit and explicit knowledge 2-12
2.4.1.4 A definition of knowledge and where knowledge resides 2-12
2.4.2 Knowledge management (KM) 2-13
2.4.2.1 Knowledge-focused vs. information-focused streams of KM 2-13
2.4.2.2 The overall goal of KM 2-14
2.4.3 Knowledge Processes 2-14
2.4.4 Knowledge Flows 2-15
2.4.5 Knowledge Forms 2-16
2.5 A typology of KM 2-16
2.5.1 Transactional KM 2-19
2.5.2 Analytical KM 2-19
2.5.3 Management of knowledge assets 2-19
2.5.4 Process-based KM 2-20
2.5.5 Developmental KM 2-20
2.5.6 Innovation management 2-20
2.6 Knowledge management systems (KMSs) 2-21
2.6.1 The two principle uses of a KMS 2-22
2.6.2 Growth of a KMS 2-22
2.6.3 Establishment and evolution of a KMS 2-23
2.6.4 Structure of a generic KMS 2-25
2.6.5 Visibility of a KMS 2-26
2.6.6 Framework of a KMS 2-27
2.7 Roles of people involved with KM 2-27
2.7.1 Knowledge suppliers and knowledge consumers 2-27
2.7.2 Chief Knowledge Officer (CKO) 2-28
2.7.3 KMS user 2-28
2.7.4 Knowledge worker 2-29
2.7.5 Change agent 2-29
2.7.6 Knowledge engineer 2-29
2.7.7 Knowledge steward 2-30
2.7.8 Knowledge analyst 2-31
2.7.9 Knowledge broker 2-31
2.8 KM in technical product development 2-32
2.8.1 Managing development teams and their knowledge 2-33
2.8.1.1 KM steering committee 2-33
2.8.1.2 Communities of practice 2-33
2.8.1.3 Team learning 2-34
2.8.1.4 Team knowledge sharing 2-34
2.8.1.5 Distributed teams 2-35
2.8.1.6 Sub-contracting 2-35
2.8.2 KM tools for managing individual and team knowledge 2-35
2.8.2.1 Training workshops 2-35
2.8.2.2 Yellow Pages 2-35
2.8.2.3 Performance analysis 2-36
2.8.2.4 Responsibility charts 2-36
2.8.2.5 Status tracking 2-37
2.8.2.6 AI tools 2-37
2.8.2.7 Shared buffers 2-37
2.8.3 Managing information in technical development projects 2-38
2.8.3.1 Issues in information management 2-38
2.8.3.2 Tools for information management 2-39

University of Cape Town

2.8.4 Managing knowledge of technical development processes 2-42
2.8.4.1 Development process knowledge: the input, in-situ and output knowledge types 2-42
2.8.4.2 Input, in-situ and output knowledge in embedded software development projects 2-43
2.8.4.3 Approaches to software engineering processes improvement 2-44
2.8.4.4 Issues in software processes KM 2-44
2.8.4.5 Tools for managing knowledge of software development processes 2-49
2.8.5 Managing innovation in technical product development 2-51
2.8.5.1 Management of innovation issues in product development 2-53
2.8.5.2 Tools for managing innovation in product development 2-54
2.8.6 Dealing with information overload 2-56
2.8.6.1 Dimensions of information overload 2-57
2.8.6.2 Addressing information overloading with infomediary tools 2-57
2.9 Conceptual framework for researching a KMS 2-58
2.10 Summary and conclusion 2-60
Chapter 3: Researching embedded system artefact organisation and adaptation (ESAOA) knowledge 3-1
3.1 Key concepts 3-2
3.1.1 ESAOA knowledge 3-2
3.1.2 Towards a study of directed KMS evolution 3-3
3.1.3 Directed KMS evolution 3-3
3.2 Research objective: A KMS for ESAOA activities 3-4
3.2.1 Specific objective: Moving from an *ad hoc* to a formalised KMS 3-5
3.2.2 Scope and delimitation: ESAOA during component integration 3-6
3.3 Research problems 3-9
3.3.1 Associative memory, time-limited knowledge, and repeated learning 3-9
3.3.2 Information overload 3-10
3.3.3 Research challenges: Confidence, confusion, and lost property 3-11
3.4 Problem statement 3-11
3.4.1 Research question 3-11
3.4.2 Sub-problems 3-12
3.4.3 Research assumption 3-13
3.5 Research design 3-14
3.5.1 Research design for evolving the ESAOA KMS 3-14
3.5.2 Overview of Experiment 1 3-16
3.5.3 Construction of the initial ESAOA KMS 3-17
3.5.4 Overview of Experiment 2 3-17
3.5.5 Construction of the refined ESAOA KMS 3-17
3.6 Selection criteria: ESAOA activities 3-17
3.6.1 ESAOA project selection and project briefs 3-18
3.6.2 Site selection 3-21
3.6.3 Selection of embedded platform, cross-compilers, and IDE 3-22
3.6.4 Participant selection 3-24
3.6.5 Reviewer selection 3-25
3.7 Ethical considerations in the ESAOA activities 3-26
3.8 Data collection 3-26
3.8.1 Code and design reviews 3-27
3.8.2 Email archive 3-29
3.8.3 Group forums 3-29
3.8.4 Project meetings 3-30
3.8.5 Developer logs 3-31
3.8.6 Product demonstrations and project evaluations 3-32

University of Cape Town

3.8.7 End-of-project survey 3-38
3.8.8 Limitations of the data capture methods 3-41
3.9 Data analysis 3-41
3.9.1 Overview of data analysis 3-42
3.9.2 Systematising the data (step 1) 3-43
3.9.3 Categorising knowledge events by knowledge type (step 2) 3-45
3.9.4 Mapping problems and solutions (step 3) 3-46
3.9.5 Categorising productive vs. non-productive knowledge (step 4) 3-46
3.9.6 Finalizing the knowledge register (step 5) 3-48
3.9.7 Analysing trends 3-50
3.9.8 Analysing other forms of data 3-55
3.10 Data synthesis 3-56
3.11 The ESAOA Conceptual Modelling Language 3-58
3.11.1 ESAOA modelling atoms 3-59
3.11.2 Connectors 3-63
3.11.3 Spaces 3-66
3.11.4 Comments and constrains 3-66
3.11.5 External processes and artefacts 3-67
3.12 Comparing artefact and prototype quality with KMS analysis results 3-68
3.13 Conclusion 3-69
Chapter 4: First experiment findings and ESAOA KMS version 1 4-1
4.1 The First Experiment 4-2
4.2 Preliminary study to establish the data analysis method 4-3
4.2.1 Denoting artefacts and ESAOA activities in the data 4-3
4.2.2 Verification of KM models 4-3
4.2.3 Problem-solution cycles 4-5
4.2.4 Trivial and non-trivial solution cycles 4-8
4.2.5 Knowledge events 4-11
4.2.6 Knowledge event types (KETs) 4-11
4.2.7 Data, process and innovation knowledge categories 4-12
4.2.8 Productive and non-productive knowledge categories 4-14
4.2.8.1 Definition of non-productive and productive knowledge 4-15
4.2.8.2 Using dead-ends to determine non-productive knowledge 4-16
4.2.8.3 Backwards tracing to classify knowledge events as productive or non-productive 4-16
4.2.9 Productive time and non-productive time 4-18
4.2.10 Knowledge event chains 4-18
4.2.11 Visualizing event chains using event chain graphs 4-19
4.2.12 Development of the KMS analysis strategy 4-21
4.3 Results 4-21
4.3.1 Results of data synthesis (step 1): Initial knowledge registers 4-21
4.3.2 Results of categorising knowledge events (step 2) 4-22
4.3.3 Results of problem/solution mapping (step 3): Event chains and event chain tables 4-22
4.3.4 Categorising knowledge events according to productive and non-productive knowledge (step 4) 4-25
4.3.4.1 Classifying knowledge events as productive or non-productive 4-25
4.3.4.2 Calculation of non-productive and productive time 4-26
4.3.5 Finalizing the knowledge registers (step 5) 4-27
4.4 Trend analysis and graphing 4-28
4.4.1 Results of P1-1 (SoSiG) 4-30
4.4.1.1 Productivity graphs 4-30
4.4.1.2 Productive and non-productive time summary tables 4-33
4.4.2 Results of P1-2 (ANTCON) 4-34
4.4.2.1 Productivity graphs 4-34

University of Cape Town

4.4.2.2 Productive and non-productive time summary tables 4-36
4.4.3 Synopsis of Experiment 1 results 4-37
4.5 Design of the second iteration of framework analysis 4-38
4.5.1 Refinements to data capture methods for Experiment 2 4-39
4.5.1.1 Focusing on the knowledge-rich data sources 4-39
4.5.1.2 Changing the unit of analysis to event chains 4-40
4.5.1.3 Data capture supporting event chains 4-41
4.5.2 Changes to the analysis methods 4-41
4.5.2.1 Refinements of data synthesis methods 4-41
4.5.2.2 Refinements to graphing methods 4-43
4.5.3 Establishing a basis for comparison between experiments using knowledge occurrences 4-43
4.5.3.1 Knowledge occurrence tables and graphs for P1-1 (SoSiG) 4-43
4.5.3.2 Knowledge occurrence tables and graphs for P1-2 (ANTCON) 4-48
4.5.4 Overall results of Experiment 1 in knowledge occurrences 4-51
4.6 First application of framework construction: ESAOA KMS version 1 4-51
4.6.1 Overview of ESAOA KMS version 1 4-52
4.6.2 ESAOA workspaces and workstations 4-55
4.6.2.1 ESAOA workspaces 4-55
4.6.2.2 ESAOA workstations 4-56
4.6.3 The ESAOA knowledge ontology 4-59
4.6.3.1 Levels of the ESAOA knowledge ontology 4-60
4.6.3.2 Top-level terms of the ESAOA knowledge ontology 4-60
4.6.3.3 Knowledge artefacts and boundary artefacts 4-62
4.6.3.4 Evolving the ESAOA knowledge ontology 4-62
4.6.4 Roles 4-63
4.6.4.1 Representation of roles in the ESAOA modelling language 4-64
4.6.4.2 General relations between the roles 4-64
4.6.4.3 Maximising support for the IE using a feed-forward approach .. 4-65
4.6.5 ESAOA artefacts for knowledge representation and transfer 4-67
4.6.5.1 Artefact form classifications: hard and soft artefacts 4-68
4.6.5.2 Artefact functionality classifications and functionality hierarchy 4-69
4.6.5.3 Artefact role and workspace classifications 4-72
4.6.5.4 Artefact organisation 4-74
4.6.5.5 Specialised KM artefacts 4-76
4.6.6 ESAOA KM workflows and processes 4-79
4.6.6.1 Processes of the chief knowledge officer (CKO) 4-79
4.6.6.2 Processes of the communal knowledge steward (CKS) 4-81
4.6.6.3 Processes of the team leader (TL) 4-81
4.6.6.4 Processes of the data steward (DS) 4-82
4.6.6.5 Processes of the process engineer (PE) 4-83
4.6.6.6 Processes of the innovation engineer (IE) 4-84
4.6.7 Software design of ESAOA workspaces 4-86
4.6.7.1 ESAOA scripts and tools 4-87
4.6.7.2 The Kit for Information Technology (KIT) 4-89
4.6.7.3 The central server and the networking infrastructure 4-89
4.6.8 Implementation and distribution of ESAOA workspaces 4-90
4.6.8.1 Implementation of the ESAOA communal workspace 4-91
4.6.8.2 ESAOA team and personal workspace 4-96
4.6.8.3 ESAOA workstation distribution 4-97
4.6.8.4 Sample installation of ESAOA workspaces 4-98
4.7 Towards Experiment 2 4-99
Chapter 5: The Second Experiment 5-1
5.1 Overview of the second experiment 5-1
5.2 Results of the second experiment 5-3

UNIVERSITY OF CAPE TOWN

5.2.1 P2-1 Location-aware Tourist Information System (TIS) 5-5
5.2.2 P2-2 GPS Bus Tracker (GBT) 5-10
5.2.3 P2-3 Vibynet 5-15
5.2.4 P2-4 MyIP Phone Station (MPS) 5-19
5.2.5 P2-5 Home Automation System (HAS) 5-23
5.2.6 P2-6 Automation Headlights Dimmer (AHD) 5-27
5.2.7 P2-7 Field Sensor for Maglev Trains (FSMT) 5-31
5.2.8 P2-8 Cordless Stereo (CST) 5-35
5.2.9 P2-9 Central Alarm Clock (CAC) 5-39
5.2.10 P2-10 Voice Activation System (VAS) 5-43
5.2.11 P2-11 Supermarket Query Device (SQD) 5-48
5.2.12 P2-12 Personal Protection Device (PPD) 5-53
5.2.13 P2-13 Vehicle Usage Tracker (VUT) 5-56
5.3 Summary of knowledge occurrences 5-60
5.4 Evaluations of artefacts, prototypes and demonstrations 5-61
5.4.1 Evaluations of code and design reviews 5-62
5.4.1.1 Results from evaluation forms 5-62
5.4.1.2 Comments from knowledge production questions 5-65
5.4.1.3 Notes from design review 3 5-67
5.4.2 Review panel's evaluations 5-67
5.4.2.1 Demonstration check sheet results 5-68
5.4.2.2 Requirements check sheet results 5-69
5.5 Comparisons 5-70
5.5.1 Comparing requirements and demonstration check sheets scores .. 5-70
5.5.2 Comparing design reviews ratings to check sheet scores 5-71
5.5.3 Comparing design reviews to knowledge production statistics 5-72
5.5.3.1 Comparing code and design reviews to productive knowledge occurrences 5-72
5.5.3.2 Comparing code design reviews to knowledge occurrences 5-74
5.5.4 Comparisons with productive innovation knowledge 5-75
5.5.4.1 Comparing productive innovation knowledge and averaged scores for check sheets 5-75
5.5.4.2 Comparing productive innovation knowledge to demonstration check sheet scores 5-76
5.5.4.3 Comparing productive innovation knowledge to requirements check sheet scores 5-76
5.5.5 Comparing check sheet scores and knowledge occurrences 5-77
5.5.6 Comparing check sheet scores and proportions of knowledge 5-78
5.6 Team members' evaluation of ESAOA KMS 5-79
5.6.1 Quantitative data: 5-point scale rankings 5-79
5.6.2 Qualitative data: comments from participants 5-82
5.6.2.1 Difficulties 5-82
5.6.2.2 Benefits 5-82
5.7 General conclusions for ESAOA KMS version 1 5-83
5.7.1 Summary of knowledge occurrences 5-84
5.7.2 Process knowledge components: role, logistics and innovation knowledge 5-86
5.7.3 Comparison of Experiments 1 and 2 5-88
5.7.4 Trends noted from application of ESAOA KMS (version 1) 5-90
5.7.4.1 Emerging relationship between innovation knowledge and quality of prototype 5-90
5.7.4.2 Emerging trends across event chains 5-92
5.7.4.3 Progression towards innovation 5-93
5.7.5 Variables that affected the ESAOA KMS (version 1) 5-97
5.7.6 Effect of the ESAOA KMS (version 1) 5-99

University of Cape Town

5.7.7 Study of knowledge forms contributed by roles 5-99
5.7.8 Tool versus component knowledge occurrences 5-103
5.7.9 Logistics and role process knowledge 5-105
5.8 Implications for ESAOA KMS version 2 5-107
5.8.1 ESAOA distribution 5-107
5.8.1.1 ESAOA tools – technical installation guidelines (version 2) 5-107
5.8.1.2 Increased flexibility in ESAOA tools (version 2) 5-107
5.8.2 ESAOA roles 5-108
5.8.2.1 Training 5-108
5.8.3 ESAOA technical manual 5-108
5.8.4 ESAOA project management 5-108
5.8.5 Team workspace 5-109
5.8.6 Towards ESAOA version 2 5-109
Chapter 6: ESAOA KMS version 2 6-1
6.1 Overview of ESAOA KMS version 2 6-2
6.1.1 Use of ESAOA workspaces and workstations 6-3
6.1.2 Changes to roles and role support structures 6-3
6.1.2.1 Reducing priority of innovation and flattening role hierarchy 6-4
6.1.2.2 Towards a bi-directional flow of innovation 6-4
6.1.3 Revised roles and artefact classifications 6-6
6.1.3.1 Component researcher (CR), workspace administrator (WA) 6-6
6.1.3.2 Revisions to the PE and IE roles 6-6
6.1.3.3 Revision to the TL role 6-8
6.1.3.4 Role extension 6-8
6.1.4 Upgrading of support tools 6-9
6.1.5 Improving ESAOA documentation 6-9
6.2 ESAOA knowledge ontology 6-9
6.2.1 Additions to the ESAOA knowledge ontology 6-10
6.2.2 Evolving the ESAOA knowledge ontology 6-11
6.2.2.1 Maintaining the lower level of the knowledge ontology 6-11
6.2.2.2 Maintaining the upper level of the knowledge ontology 6-11
6.3 ESAOA version 2 workspaces 6-13
6.3.1 Definition of an ESAOA workspace 6-13
6.3.2 Definition of an ESAOA workstation 6-14
6.3.3 ESAOA workspaces implementation and access levels 6-15
6.3.4 Installing workspaces using ESAOA distributions 6-17
6.3.5 GUI installation tool for ESAOA personal workspaces 6-18
6.3.6 ESAOA version 2 distribution support documentation 6-19
6.3.7 ESAOA version 2 workspace directory structures 6-20
6.3.8 The knowledge base within ESAOA workspaces 6-22
6.4 ESAOA support tools 6-23
6.4.1 The ESAOA file classification (fclass) tool 6-24
6.4.1.1 Review of version 1 of *fclass* 6-25
6.4.1.2 Version 2 of *fclass* and addition of the PEP service 6-27
6.4.1.3 Speeding-up the operation of *fclass using the PEP service* 6-28
6.4.1.4 Operation of the PEP service 6-30
6.4.1.5 Improvements to the CSV files for storing file metadata 6-30
6.4.1.6 The *fclass* HTML generator mode 6-32
6.4.2 Addition of the hotspot logging (*hsl*) tool 6-33
6.4.3 Improvement to the *esaoa-project* tool 6-36
6.4.4 Tools for synchronizing team and personal workspaces 6-36
6.4.5 Workstation-side scripts 6-37
6.5 ESAOA roles 6-38
6.5.1 The WA and CR roles 6-39
6.5.2 Chain of command 6-40

University of Cape Town

6.5.3 Role responsibilities 6-41
6.5.4 Division of labour in development team 6-45
6.5.5 Role interrelations and workspaces 6-47
6.6 ESAOA Processes 6-49
6.6.1 Processes of the chief knowledge officer (CKO) 6-49
6.6.2 Processes of the communal knowledge steward (CKS) 6-51
6.6.3 Processes of the team leader (TL) 6-52
6.6.4 Processes of the component researcher (CR) 6-55
6.6.5 Processes of the process engineer (PE) 6-56
6.6.6 Processes of the workspace administrator (WA) 6-59
6.6.7 Processes of the innovation engineer (IE) 6-60
6.7 Artefacts 6-62
Chapter 7: Conclusions and Future Work 7-1
7.1 Response to research questions and sub-problems 7-2
7.1.1 Sub-problem 1 response: Different forms of ESAOA knowledge. 7-3
7.1.2 Sub-problem 2 response: The relative complexities of ESAOA KM tasks were found to differ 7-4
7.1.3 Sub-problem 3 response: Difficulty of producing different forms of ESAOA knowledge varied 7-4
7.1.4 Sub-problem 4 response: the time to complete ESAOA activities depends on their complexity, their dependence on other activities and the provision and understanding of KMS support 7-6
7.1.5 Sub-problem 5 response: developers encounter similar types of ESAOA KM problems and solutions in different projects 7-9
7.1.6 Sub-problem 6 response: although dead-ends did occur in ESAOA knowledge production, their number was reduced 7-10
7.1.7 Sub-problem 7 response: the degree to which the ESAOA KMS is used depends on the complexity of ESAOA activities concerned 7-10
7.1.8 Sub-problem 8 response: benefit of the ESAOA KMS depends on the complexity, difficulty and duration of the activities performed 7-11
7.2 Reflection of research findings and resolution of research question 7-12
7.3 Summary of contributions 7-14
7.4 Future work 7-14
7.4.1 Testing ESAOA KMS version 2 7-14
7.4.2 Testing ESAOS KMS on different type of ES engineering 7-14
7.4.3 Phasing in a KMS within existing/ongoing projects 7-15
7.4.4 Broadening the context for the ESAOA KMS 7-15
7.4.5 A KMS that allows for future software and hardware developments 7-15
7.4.6 Need for further research into KM in ES development 7-15
7.4.7 Focus on ES innovation knowledge 7-15
References R-1
Appendix A: Experiment 1 appendices A-1
A.1 Knowledge register for first case study (P1-1) A-1
A.2 Knowledge register for the second case study (P1-2) A-8
Appendix B: Experiment 2 appendices B-1
B.1 Knowledge register for Project P2-1 ..... B-1
B.2 Requirements check sheets for Experiment 2 ..... B-5
B.3 Comments from requirements check sheets for Experiment 2 ..... B-6
B.4 Evaluation forms used to rate code and design reviews B-9
B.4.1 Evaluation of concept creativity B-9
B.4.2 Evaluation of design quality B-10
B.4.3 Evaluation of artefact quality B-11
B.5 Design review 2 questions regarding knowledge production B-12
Appendix C: ESAOA KMS version 2 appendices C-1
C.1 Knowledge ontology for ESAOA KMS version 2 ..... C-1

University of Pretoria

C.2 Details concerning the ESAOA modelling language C-3
C.2.1 Further detail on connectors C-3
C.3 Design details related to ESAOA support tools C-5
C.3.1 Design issues of the Personal Expert Program (PEP) C-5
C.3.2 Detailed design issues related to *fclass* and related CSV files C-6
C.3.3 Hotspot logging (hsl) tool C-7
C.3.4 List of ESAOA support tools C-7
C.4 Kit for Information Technology (KIT) C-13
C.4.1 KIT modules and UML class model C-13
C.4.2 KIT sample application: esaoa-apps C-16
Appendix D: Case study participants D-1
D.1 Experiment 1 participants D-1
D.2 Experiment 2 participants D-1
Appendix E: Supplementary documentation E-1
E.1 A comparison of search results E-1
Glossary G: Glossary G-1

University of Cape Town

List of Figures

Figure 1.1: The association between projects, tasks and activities. 1-4
Figure 1.2: Examples of implementation artefacts. 1-6
Figure 1.3: Thesis objective – evolving a KMS for ESAOA activities. 1-14
Figure 1.4: Visualization of the thesis focus and delimitations. 1-17
Figure 2.1: Embedded system lifecycle model 2-5
Figure 2.2: The data, information, and knowledge hierarchy, with scenario. 2-11
Figure 2.3: The knowledge process adapted from Radding [1998]. 2-15
Figure 2.4: Flow of knowledge from supplier to consumer. 2-16
Figure 2.5: The KM spectrum (adapted from Binney [2001]). 2-18
Figure 2.6: ES project knowledge sharing chart. 2-37
Figure 2.7: Embedded software project KM planning template. 2-41
Figure 3.1: Directed KMS evolution. 3-4
Figure 3.2: Research design for studying evolution of the ESAOA KMS 3-15
Figure 3.3: The CSB337 evaluation board produced by Cogent Computers 3-23
Figure 3.4: Consent letter emailed to participants in Experiments 1 and 2 3-26
Figure 3.5: Example of activities log. 3-27
Figure 3.6: Example of email message (email #283) stored in the email archive. 3-29
Figure 3.7: Example of a posting from the 'Project Discussion Forum'. 3-30
Figure 3.8: Example of developer log. 3-32
Figure 3.9: Example of ES prototype set up for a demonstration. 3-34
Figure 3.10: Demonstration check sheet for project P2-6. 3-36
Figure 3.11: First page of the requirements check sheet. 3-37
Figure 3.12: Second page of the requirements check sheet. 3-38
Figure 3.13: First page of questionnaire completed by a team member. 3-39
Figure 3.14: Second page of questionnaire completed by a team member. 3-40
Figure 3.15: The data analysis process. 3-42
Figure 3.16: Sample annotated forum posting taken from Experiment 2. 3-48
Figure 3.17: How knowledge occurrence graphs relate to event chains. 3-53
Figure 3.18: Data synthesis process, in which ecology maps were produced 3-57
Figure 3.19: The principal shapes of ESAOA modelling atoms 3-59
Figure 3.20: Role modelling atoms of ESAOA KMS version 2. 3-60
Figure 3.21: Artefacts atoms showing general artefact, hard artefact, soft artefact, and more specialised artefact atoms. 3-61
Figure 3.22: A team boundary data artefact. 3-61
Figure 3.23: A group-work process that involves application of knowledge. 3-62
Figure 3.24: Knowledge atom classified as process knowledge. 3-62
Figure 3.25: List of ESAOA modelling language connector types. 3-64
Figure 3.26: Examples of (a) explicit junction, and (b) tacit junction 3-65
Figure 3.27: Model demonstrating connector labels and multiplicity. 3-66
Figure 3.28: Model showing space modeling elements. 3-67
Figure 3.29: Model showing an external process and external artefact. 3-67
Figure 4.1: An initial version of the ESAOA modelling language used to describe relations between ESAOA artefacts and activities in Project P1-1 4-4
Figure 4.2: The 'trivial solution cycle' 4-9
Figure 4.3: Illustration of the 'non-trivial solution' cycle 4-10
Figure 4.4: Relationship between ESAOA activities and knowledge events 4-11
Figure 4.5: Determining non-productive and productive knowledge acquisition. 4-17
Figure 4.6: Event chains graph 4-20
Figure 4.7: ESAOA workspace and ESAOA workstation. 4-54

University of Cape Town

Figure 4.8: Screenshot of the Bash shell environment of an ESAOA workspace.. 4-56
Figure 4.9: Annotated screenshot of an ESAOA workstation. 4-57
Figure 4.10: Photograph illustrating the broader concept of an ESAOA workstation. 4-59
Figure 4.11: UML diagram visualizing part of higher-level ESAOA knowledge ontology. 4-61
Figure 4.12: Roles of ESAOA KMS version 1. 4-65
Figure 4.13: Role support structure for ESAOA version 1 4-66
Figure 4.14: The feed-forward flow from the DS, to the PE, ending at the IE. 4-66
Figure 4.15: Screenshot showing use of the *fclass* program 4-72
Figure 4.16: Model describing the classification of ESAOA artefacts. 4-74
Figure 4.17: ESAOA version 1 communal distribution directory structure. 4-75
Figure 4.18: ESAOA version 1 team distribution directory structure. 4-76
Figure 4.19: Concept sketch created in a project P1-2 meeting (event chain 44). 4-77
Figure 4.20: Concept drawing produced in project P1-2 (part of event chain 44).. 4-78
Figure 4.21: AOD for project P1-2 (see Section 3.11 for modelling language). 4-79
Figure 4.22: Processes performed by the CKO. 4-80
Figure 4.23: Major processes performed and maintained by the CKS. 4-81
Figure 4.24: Major role interrelations and processes performed by the TL. 4-82
Figure 4.25: (a) DS training process; (b) DS search process. 4-83
Figure 4.26: (a) Interaction between PE, CKO and CKS; (b) Processes performed by the PE that involve development process knowledge. 4-84
Figure 4.27: (a) Collaboration between CKO, CKS and IE; (b) Processes and artefacts used and managed by the IE. 4-86
Figure 4.28: Implementation of the esaoa-snap script. 4-88
Figure 4.29: Implementation of the esaoa-fm program. 4-88
Figure 4.30: Consecutive loads of Bash environments for ESAOA workspaces. .. 4-90
Figure 4.31: Top-level directory structure of ESAOA communal workspace. 4-91
Figure 4.32: Using Cygwin to access a team workspace on the central server. 4-97
Figure 4.33: Using Cygwin to access a project in an ESAOA team workspace. 4-97
Figure 4.34: Directory structure of ESAOA workstation distribution. 4-98
Figure 4.35: Sample ESAOA workspace installation. 4-99
Figure 5.2.1 (a): Component interconnection diagram; (b): Enclosure drawing. 5-6
Graph 5.2.1 (a): Data knowledge in P2-1. 5-8
Graph 5.2.1 (b): Process knowledge in P2-1. 5-8
Graph 5.2.1 (c): Innovation knowledge in P2-1 5-9
Graph 5.2.1 (d): Productive and non-productive knowledge in P2-1. 5-9
Figure 5.2.2: Concept scenario for the GPS Bus Tracker (Project P2-2). 5-12
Graph 5.2.2 (a): Data knowledge in P2-2. 5-13
Graph 5.2.2 (b): Process knowledge in P2-2. 5-13
Graph 5.2.2 (c): Innovation knowledge in P2-2. 5-14
Graph 5.2.2 (d): Productive and non-productive knowledge in P2-2. 5-14
Figure 5.2.3: Concept drawing for Vibynet (Project P2-3). 5-16
Graph 5.2.3 (a): Data knowledge in P2-3. 5-17
Graph 5.2.3 (b): Process knowledge in P2-3. 5-17
Graph 5.2.3 (c): Innovation knowledge in P2-3. 5-18
Graph 5.2.3 (d): Productive and non-productive knowledge in P2-2. 5-18
Figure 5.2.4: Topology of myIP Phone Station (Project P2-4) 5-20
Graph 5.2.4 (a): Data knowledge in P2-4. 5-21
Graph 5.2.4 (b): Process knowledge in P2-4. 5-21
Graph 5.2.4 (c): Innovation knowledge in P2-4. 5-22
Graph 5.2.4 (d): Productive and non-productive knowledge in P2-4. 5-22
Figure 5.2.5: Installation diagram of Home Automation System (Project P2-5). 5-24
Graph 5.2.5 (a): Data knowledge in P2-5. 5-25
Graph 5.2.5 (b): Process knowledge in P2-5. 5-25

Cambridge University Press

Graph 5.2.5 (c): Innovation knowledge in P2-5. 5-26
Graph 5.2.5 (d): Productive and non-productive knowledge in P2-5. 5-26
Figure 5.3.6: Installation of the Automation Headlight Dimmer (Project P2-6). 5-28
Graph 5.2.6 (a): Data knowledge in P2-6. 5-29
Graph 5.2.6 (b): Process knowledge in P2-6. 5-29
Graph 5.2.6 (c): Innovation knowledge in P2-6. 5-30
Graph 5.2.6 (d): Productive and non-productive knowledge in P2-6. 5-30
Figure 5.2.7: Component interconnection diagram (Project P2-7). 5-32
Graph 5.2.7 (a): Data knowledge in P2-7. 5-33
Graph 5.2.7 (b): Process knowledge in P2-7. 5-33
Graph 5.2.7 (c): Innovation knowledge in P2-7. 5-34
Graph 5.2.7 (d): Productive and non-productive knowledge in P2-7. 5-34
Figure 5.2.8: Concept drawing for the Cordless Stereo (Project P2-8). 5-36
Graph 5.2.8 (a): Data knowledge in P2-8. 5-37
Graph 5.2.8 (b): Process knowledge in P2-8. 5-37
Graph 5.2.8 (c): Innovation knowledge in P2-8. 5-38
Graph 5.2.8 (d): Productive and non-productive knowledge in P2-8. 5-38
Figure 5.2.9: Installation diagram for Central Alarm Clock (Project P2-9). 5-40
Graph 5.2.9 (a): Data knowledge in P2-9. 5-41
Graph 5.2.9 (b): Process knowledge in P2-9. 5-41
Graph 5.2.9 (c): Innovation knowledge in P2-9. 5-42
Graph 5.2.9 (d): Productive and non-productive knowledge in P2-9. 5-42
Figure 5.2.10 (a): First scene in the concept cartoon (Project P2-10). 5-44
Figure 5.2.10 (b): Second scene of concept cartoon for operation of VAPs. 5-45
Graph 5.2.10 (a): Data knowledge in P2-10. 5-46
Graph 5.2.10 (b): Process knowledge in P2-10. 5-47
Graph 5.2.10 (c): Innovation knowledge in P2-10. 5-47
Graph 5.2.10 (d): Productive and non-productive knowledge in P2-10. 5-48
Figure 5.2.11: Concept diagram of Supermarket Query Device (Project P2-11). 5-49
Graph 5.2.11 (a): Data knowledge in P2-11. 5-50
Graph 5.2.11 (b): Process knowledge in P2-11. 5-51
Graph 5.2.11 (c): Innovation knowledge in P2-11. 5-51
Graph 5.2.11 (d): Productive and non-productive knowledge in P2-11. 5-52
Figure 5.2.12: Concept poster for Personal Protection Device (Project P2-12). 5-53
Graph 5.2.12 (a): Data knowledge in P2-12. 5-54
Graph 5.2.12 (b): Process knowledge in P2-12. 5-54
Graph 5.2.12 (c): Innovation knowledge in P2-12. 5-55
Graph 5.2.12 (d): Productive and non-productive knowledge in P2-12. 5-55
Figure 5.2.13: Component interconnection diagram (Project P2-13). 5-57
Graph 5.2.13 (a): Data knowledge in P2-13. 5-58
Graph 5.2.13 (b): Process knowledge in P2-13. 5-58
Graph 5.2.13 (c): Innovation knowledge in P2-13. 5-59
Graph 5.2.13 (d): Productive and non-productive knowledge in P2-13. 5-59
Figure 5.7.1: Bar chart showing percentage breakdown of productive and non-productive knowledge occurrences per knowledge category for projects. 5-85
Figure 5.7.4 (a): Model of a *productive episode* (top) and corresponding knowledge occurrence graphs (bottom) 5-96
Figure 5.7.4 (b): Model of a *non-productive episode* (top) and corresponding knowledge occurrence graphs (bottom) 5-97
Figure 5.7.7: Pie charts showing contribution of each role. 5-103
Figure 6.1: The role support structure and flow of knowledge around which ESAOA KMS version 2 is designed. 6-8
Figure 6.2: UML model for part of the high level ESAOA knowledge ontology. 6-10
Figure 6.3: Screenshots from ESAOA ontology manager. 6-12
Figure 6.4: Model showing composition of an ESAOA workspace. 6-15

University of Cape Town

Figure 6.5: The three types of ESAOA workspace. 6-16
Figure 6.6: Screenshot of prototype personal workstation installation program. ... 6-19
Figure 6.7: ESAOA version 2 team distribution directory structure. 6-21
Figure 6.8: Scenario showing access to integrated knowledge base using *fclass*. 6-23
Figure 6.9: Screenshot of *fclass* version 2. 6-24
Figure 6.10: UML model of relationships between *fclass*, PEP and related files... 6-28
Figure 6.11: UML diagram showing software design of the fclass program. 6-29
Figure 6.12: Scenario demonstrating interaction between *fclass* and PEP. 6-31
Figure 6.13: Screenshot showing sample HTML output of *fclass*. 6-33
Figure 6.14: Examples of hotspots taken from Project P2-2. 6-34
Figure 6.15: UML diagram showing overview of the HSL tool's software design. . 6-35
Figure 6.16: Role support provided by the WA. 6-40
Figure 6.17: Chain of command for ESAOA KMS roles. 6-41
Figure 6.18: Team members and their knowledge specialisations. 6-43
Figure 6.19: A potential scenario providing a fair division of labour. 6-46
Figure 6.20: Role support structure for ESAOA version 2. 6-48
Figure 6.21: Processes performed and maintained by the CKS. 6-51
Figure 6.22: Processes maintained and carried out by the CKS. 6-52
Figure 6.23: Processes performed and maintained by the TL. 6-53
Figure 6.24: Decision and allocation of roles. 6-54
Figure 6.25: Processes of the CR. 6-55
Figure 6.26: CR training process. 6-56
Figure 6.27: Main processes carried out by the PE. 6-57
Figure 6.28: The PE training process. 6-58
Figure 6.29: Processes involved in the interaction between the PE and IE. 6-59
Figure 6.30: Processes and role interactions concerning the WA. 6-60
Figure 6.31: Processes and artefacts used and managed by the IE. 6-61
Figure 6.32: Training the IE. 6-62

University of Cape Town

List of Tables

Table 1.1: Examples of ESAOA activities for certain implementation tasks 1-7
Table 1.2: Summary of delimitations 1-18
Table 2.1: Commonly used infomediary tools and dimensions of information overload they address [Ho & Tang, 2001; Berghel, 1997]. 2-58
Table 3.1: Overview of the projects studied in Experiment 1 and 2. 3-20
Table 3.2: Project meetings. 3-30
Table 3.3: Schedule of Experiment 1 demonstrations. 3-33
Table 3.4: Schedule of Experiment 2 demonstrations. 3-33
Table 3.5: Except of initial knowledge register from Project P1-2. 3-45
Table 3.6: Except of the knowledge register for Project P2-1. 3-49
Table 3.7: Productive and non-productive knowledge per knowledge type. 3-54
Table 3.8: Proportions of data, process and innovation knowledge produced. 3-55
Table 3.9: Commonly used artefact classification acronyms. 3-61
Table 3.10: Process classifications. 3-62
Table 3.11: Classification acronyms for knowledge atoms. 3-63
Table 4.1: Description of projects for the first experiment. 4-2
Table 4.2: Excerpt from Project P1-1. 4-5
Table 4.3: Demonstration of problem-solution cycles observed in Experiment 1. ... 4-6
Table 4.4: Types of knowledge events. 4-12
Table 4.5: Scenario for hierarchy of data, process and innovation knowledge. 4-13
Table 4.6: Taxonomy of knowledge for embedded system KM. 4-14

Table 4.7: Number of knowledge events in each of the data, process and innovation knowledge categories. 4-22
Table 4.8: The event chains for Project P1-1 (SoSiG). 4-23
Table 4.9: The event chains for Project P1-2 (ANTCON). 4-23
Table 4.10: Number of knowledge events per knowledge category for P1-1. 4-26
Table 4.11: Number of knowledge events per knowledge category for P1-2. 4-26
Table 4.12: Breakdown of knowledge acquisition times for P1-1 (SoSiG)..... 4-27
Table 4.13: Breakdown of knowledge acquisition times for P1-2 (ANTCON)..... 4-27
Table 4.14: Breakdown of knowledge acquisition times for P1-1 (SoSiG)..... 4-34
Table 4.15: Breakdown of knowledge acquisition times for P1-2 (ANTCON)..... 4-37
Table 4.16: Excerpt from knowledge register for Project P1-1 (SoSiG). 4-42
Table 4.17: Knowledge occurrences per knowledge type for P1-1. 4-44
Table 4.18: Productive and non-productive knowledge within knowledge types.. 4-44
Table 4.19: Productive and non-productive time percentages for P1-1..... 4-44
Table 4.20: Differences of results between analysis methods for Project P1-1..... 4-44
Table 4.21: Proportions of data, process and innovation knowledge for P1-1..... 4-45
Table 4.22: Knowledge occurrences per knowledge type for P1-2. 4-48
Table 4.23: Productive and non-productive knowledge within knowledge types.. 4-48
Table 4.24: Breakdown of knowledge acquisition times for P1-2. 4-48
Table 4.25: Differences of results between analysis methods for Project P1-2..... 4-48
Table 4.26: Proportions of data, process and innovation knowledge for P2-2..... 4-49
Table 4.27: Average of productive and non-productive knowledge for knowledge categories for Experiment 1..... 4-51
Table 4.28: Total productive and non-productive knowledge for Experiment 1..... 4-51
Table 4.29: Roles of ESAOA version 1..... 4-61
Table 4.30: Roles of ESAOA version 1..... 4-64
Table 4.31: Excerpt of artefact functional classes..... 4-70
Table 4.32: List of commonly applied role classifications..... 4-73
Table 4.33: ESAOA KMS version 1 distributions. 4-91
Table 4.34: ESAOA communal scripts. 4-92
Table 4.35: ESAOA communal programs..... 4-94
Table 4.36: ESAOA communal templates. 4-96
Table 5.1: List of Experiment 2 Projects. 5-2
Table 5.2.1 (a): Productive and non-productive knowledge per knowledge type... 5-10
Table 5.2.1 (b): Proportions of data, process and innovation knowledge in P2-1.. 5-10
Table 5.2.2 (a): Productive and non-productive knowledge per knowledge type... 5-15
Table 5.2.2 (b): Proportions of data, process and innovation knowledge in P2-2.. 5-15
Table 5.2.3 (a): Productive and non-productive knowledge per knowledge type... 5-19
Table 5.2.3 (b): Proportions of data, process and innovation knowledge in P2-3.. 5-19
Table 5.2.4 (a): Productive and non-productive knowledge per knowledge type... 5-23
Table 5.2.4 (b): Proportions of data, process and innovation knowledge in P2-4.. 5-23
Table 5.2.5 (a): Productive and non-productive knowledge per knowledge type... 5-27
Table 5.2.5 (b): Proportions of data, process and innovation knowledge in P2-5.. 5-27
Table 5.2.6 (a): Productive and non-productive knowledge per knowledge type... 5-31
Table 5.2.6 (b): Proportions of data, process and innovation knowledge in P2-6.. 5-31
Table 5.2.7 (a): Productive and non-productive knowledge per knowledge type... 5-35
Table 5.2.7 (b): Proportions of data, process and innovation knowledge in P2-7.. 5-35
Table 5.2.8 (a): Productive and non-productive knowledge per knowledge type... 5-39
Table 5.2.8 (b): Proportions of data, process and innovation knowledge in P2-8.. 5-39
Table 5.2.9 (a): Productive and non-productive knowledge per knowledge type... 5-43
Table 5.2.9 (b): Proportions of data, process and innovation knowledge in P2-9.. 5-43
Table 5.2.10 (a): Productive and non-productive knowledge per knowledge type. 5-48
Table 5.2.10 (b): Proportions of data, process and innovation knowledge P2-10.. 5-48
Table 5.2.11 (a): Productive and non-productive knowledge per knowledge type. 5-52
Table 5.2.11 (b): Proportions of data, process and innovation knowledge P2-11.. 5-52

University of Cape Town

Table 5.2.12 (a): Productive and non-productive knowledge per knowledge type. 5-56
Table 5.2.12 (b): Proportions of data, process and innovation knowledge P2-12.. 5-56
Table 5.2.13 (a): Productive and non-productive knowledge per knowledge type. 5-60
Table 5.2.13 (b): Proportions of data, process and innovation knowledge P2-13.. 5-60
Table 5.3.1 (a): Knowledge occurrences for each Experiment 2 project. 5-61
Table 5.4.1 (a): Breakdown of creativity ratings per project. 5-63
Table 5.4.1 (b): Breakdown of design ratings per project..... 5-63
Table 5.4.1 (c): Breakdown of artefact ratings per project. 5-64
Table 5.4.1 (d): Totals of the code and design review ratings per project. 5-64
Table 5.4.1 (e): Commonly reported knowledge production methods. 5-66
Table 5.4.2 (a): Demonstration check sheet scores for each project. 5-68
Table 5.4.2 (b): Section scores for each requirements check sheet..... 5-69
Table 5.4.3 (a): Demonstration check sheet scores compared to requirements.... 5-70
Table 5.5.2 (a): Design review averages compared to review panel scores. 5-71
Table 5.5.2 (b): Correlation results. 5-71
Table 5.5.3 (a): Correlations between code and design review scores and productive knowledge occurrences across all projects. 5-72
Table 5.5.3 (b): Correlations between review scores and knowledge occurrences.5-74
Table 5.5.4 (a): Percentage of productive innovation knowledge occurrences compared to requirements check sheet scores. 5-75
Table 5.5.4 (b): Percentage productive innovation knowledge occurrences compared to demonstration check sheet scores..... 5-76
Table 5.5.4 (b): Percentage productive innovation knowledge occurrences compared to requirements check sheet scores. 5-77
Table 5.5.5: Correlations between check sheet scores and categories of knowledge production. 5-77
Table 5.5.6: Correlations between check sheets and proportions of knowledge production. 5-78
Table 5.6.1: Summary of evaluation data. 5-80
Table 5.7.1 (a): Percentage of productive and non-productive knowledge..... 5-85
Table 5.7.3 (a): Experiment 1 averages for productive and non-productive. 5-89
Table 5.7.3 (b): Experiment 2 averages for productive and non-productive. 5-89
Table 5.7.4 (a): Emerging trend indicating relationship between productive innovation knowledge and quality of the prototype. 5-91
Table 5.7.4 (b): Trends emerging across the event chains. 5-92
Table 5.7.7 (a): Excerpt from Project P2-1 knowledge register..... 5-100
Table 5.7.7 (b): Contribution of knowledge forms per role for Project P2-1..... 5-101
Table 5.7.7 (c): Contribution of knowledge forms per role for Project P2-2. 5-101
Table 5.7.7 (d): Contribution of knowledge forms per role for Project P2-10..... 5-102
Table 5.7.8 (a): Excerpt from Project P2-1 knowledge register..... 5-104
Table 5.7.8 (b): Tool versus component knowledge occurrences. 5-105
Table 5.7.9: Separation of role, logistics, and other knowledge. 5-106
Table 6.1: ESAOA version 2 distributions..... 6-18
Table 6.2: A tabular view of a .fci file in an ESAOA personal workspace. 6-26
Table 6.3: A tabular view of a .fcl file in an ESAOA personal workspace. 6-26
Table 6.4: Tabular view of a second version '.fcl' file. 6-31
Table 6.5: Tabular view of a second version '.fos' file. 6-32
Table 6.6: Examples of a .fhl file corresponding to Figure 6.14. 6-34
Table 6.7: Roles of ESAOA KMS version 2. 6-38
Table 6.8: Examples of artefacts and role classification. 6-39
Table 6.9: Team member specialisations. 6-42
Table 6.10: Artefacts of the EMASO KMS. 6-62

University of Cape Town

Chapter 1:

Introduction

In this thesis a specific aspect of embedded system (ES) development, referred to as embedded system artefact organisation and adaptation (ESAOA) is studied. This thesis is an exploratory study that focuses on the development and evaluation of a knowledge management system (KMS) for ESAOA.

This chapter starts by defining terminology related to the broad area that this thesis is situated in, and the specialised terms that are used in this thesis (Section 1.1). The next section (Section 1.2) explains the rationale behind this research project, highlighting challenges of new ES product development and the need to speed up the process of developing these products. The thesis objective is introduced in Section 1.3. The problem statement that leads on from the thesis objective is presented in Section 1.4, and the focus of the thesis is described in Section 1.5. The delimitations of the thesis, including details of the scope and constraints of the study, are elaborated upon in Section 1.6. The final section, Section 1.7, sets out the structure of the thesis and briefly outlines the subsequent progression of chapters.

University of Cape Town

1.1 Terminology and definitions

This section defines the terms that are used in this thesis. The first of these are embedded system (ES) and embedded software development. The differences between ES products and ES prototypes are explained, which is followed by a description of computer engineering and embedded engineering. Next, differences between project tasks and project activities are identified, from which the definition of implementation tasks is derived. This leads into a definition of implementation artefacts, which are used during implementation tasks. Finally, the concept of embedded system artefact organisation and adaptation (ESAOA) activities is defined.

1.1.1 Embedded systems and embedded software development

An ES, alternatively referred to as an embedded computer, is a specialised computer system that is built into a larger system (or product) and is dedicated to performing a specific task within that larger system [Catsoulis, 2002]. The design of an ES can be highly optimised because the operations that the system needs to perform are tightly bounded [Berger, 2002]. As a result, such systems are realised in the form of specialised hardware platforms constructed from microprocessors or microcontrollers, which run embedded software, and are connected to application-specific peripherals. Embedded software is the dedicated software that runs on an ES and that coordinates and controls the hardware to make the system perform useful operations [Labrosse *et al.*, 2008].

1.1.2 ES products and prototypes

In general, a prototype serves as an early product sample that is built to test a concept [Floyd, 1984] or to determine experimentally an effective process through which a product or product range can be produced [Brinkkemper *et al.*, 1996]. For instance, if an ES development company decides to create a new type of product, the company may first develop a prototype of the product to assess a range of issues, such as an effective choice of features, power consumption, and an accurate cost estimate for a production version of the product [Berger, 2002]. Once the prototype has been built (or possibly sooner), the developers may decide that the concept is a good one (and may thus start to create a production version of the product); alternatively the developers may decide that the concept is ineffective and abandon the idea [Ulrich & Eppinger, 1995].

1.1.3 Computer engineering and embedded engineers

ES development has traditionally been divided between hardware design and software design [Berger, 2002]. Hardware design involves tasks such as the creation of schematics, the implementation of circuits, and the analysis and testing of signals [Koopman, 1996; Berger, 2002]. The hardware aspect of ES design is considered to fall within the discipline of electrical engineering [Sangiovanni-Vincentelli & Pinto, 2005; Seviora, 2005]. The software aspect of ES development relates to the field of software engineering, and involves the design and implementation of hardware drivers, signal processing routines and application code [Barr, 1999]. The low-level coding aspects of embedded software development, such as writing device drivers and coding in a hardware description language (e.g., Verilog or VHDL), are commonly referred to as firmware development [Sutter, 2002].

University of Cape Town

The separation of engineering teams between hardware engineers and software engineers remains a common practice; but boundaries between the software and hardware aspects of ES design are becoming blurred [Franke & Purvis, 1991]. For example, system-on-a-chip (SoC) technologies are providing cost effective, commercial-off-the-shelf (COTS) products that can replace entire subsystems previously developed in-house [Rowen & Leibson, 2004]. Hardware description languages, such as VHDL and Verilog, used to program field programmable gate arrays (FPGAs) are replacing circuits previously implemented on comparatively bulky printed circuit boards (PCBs) and offer better performance with more robust packages (e.g., [Sommerville, 2006]). Computer engineering (CE) concerns the construction and maintenance of computing systems, and involves aspects of both electrical engineering and computer science [Coates *et al.*, 1971]. A computer engineer is expected to have skills related to both hardware and software development, such as programming and software engineering abilities, as well as proficiency in reading PCB schematics, datasheets and diagnosing hardware faults [Soldan *et al.*, 2004; Shackelford *et al.*, 2006]. The term 'embedded engineer' or 'ES engineer' refers specifically to a CE who specialises in the development of ES products [Berger, 2002], rather than of other forms of computer systems (e.g., constructing supercomputers and notebook PCs).

1.1.4 Tasks and activities

This thesis adapts the definition of tasks and activities from descriptions used by the Project Management Institute [2004]. In this thesis, a *project* is considered to comprise a collection of tasks. A *task* is in turn accomplished through the completion of a set of activities that are performed by team members[1]. A task has a specific aim, depending on the work performed. For example, in an ES development project, a task may involve coding a device driver for a temperature sensor.

An *activity* is an action carried out by one or more members of a development team (e.g., an embedded software developer in the case of an ES development project). An activity may be task-oriented, in which case it specifically furthers the progress of a task, or peripheral, if the activity is not directly related to a task (this categorisation is derived from a discourse by Hallows [2002], which indicates that activities of an employee tend to be either centred on a certain project, or peripheral to a project). An

[1] Note that the definition of an activity in this thesis is not equivalent to an activity as defined by the Rational Unified Process (RUP) [Kroll & Kruchten, 2003].

University of Cape Town

example of a task-oriented activity, for example one that is related to the task of constructing a temperature sensor driver, is writing lines of C code for the device driver module; an example of a peripheral activity, however, is doing a file backup (i.e., a peripheral activity may be crucial even though it might not further the progress of a project). Figure 1.1 illustrates the differences between projects, tasks, task-oriented activities, and peripheral activities.

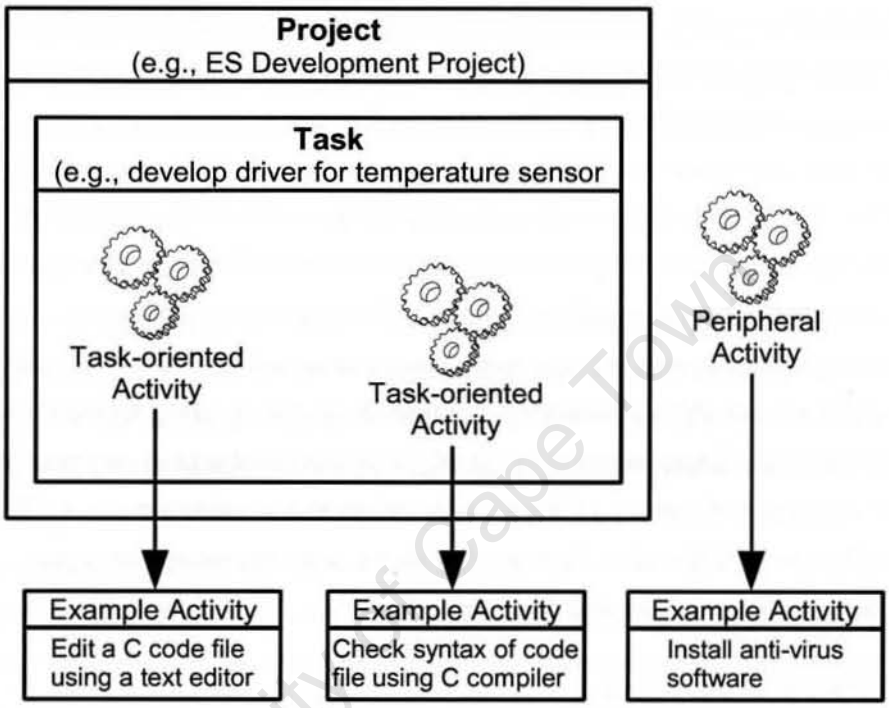


Figure 1.1: The association between projects, tasks and activities.

1.1.5 Implementation tasks

A development project can be divided into different phases, with each phase involving a collection of tasks [Berger, 2002]. Section 2.2 presents an ES development process model (an adapted version of the waterfall model [Royce, 1970]) that divides a project into seven phases, namely: 1) requirements, 2) specification, 3) design, 4) implementation, 5) integration, 6) testing, and 7) maintenance and upgrade. The separation of a project into phases is used in this thesis foremost as an analytical tool to isolate certain tasks and activities in a project. The division of a project into phases is not absolute; for example, tasks and activities are often revisited in other phases [Schach, 2005].

University of Cape Town

The term *implementation tasks* relates to tasks generally performed in the implementation phase of a development project, as per the description of the implementation phase given by Schach [2005].

In this thesis, use of the term *implementation task* refers specifically to tasks in the context of developing embedded software as part of an ES development project (see Section 1.1.4 for examples of implementation tasks that can occur in these projects).

1.1.6 Implementation artefacts, and soft/hard artefact classification

In this thesis, the term *artefact* (or *ES artefact*) refers to a file or object worked on, or used by, developers during a development project (this definition is based on the description of an artefact used in the RUP [Kroll & Kruchten, 2003]). An *implementation artefact* is an artefact used or worked on during an implementation task. The definition of an implementation artefact is intentionally broad; for example, it includes development tools (e.g., compiler and linkers), hardware components (e.g., a microcontroller) and computer files (e.g., PDF datasheets and code modules). Figure 1.2 shows further examples of implementation artefacts.

In this thesis, artefacts are classified as *hard artefacts* or *soft artefacts*. Soft artefacts are defined as digital files stored on a computer, or paper documents (e.g., a printout of a datasheet or soft artefact). Hard artefacts are physical objects that an ES engineer works with in a laboratory, such as electronic components, hardware devices, tools, equipment and the embedded system being constructed. In Figure 1.2, the code file, datasheet and schematic illustrate soft artefacts; whereas the platform, debugging component, and microcontroller demonstrate hard artefacts. The term *design artefact* refers to soft artefacts that are more closely related to the design phase of development (see Section 2.2), such as hardware block diagrams, component diagrams and UML class diagrams [Schach, 2005], which together direct the construction and adaptation of implementation artefacts.

University of Cape Town

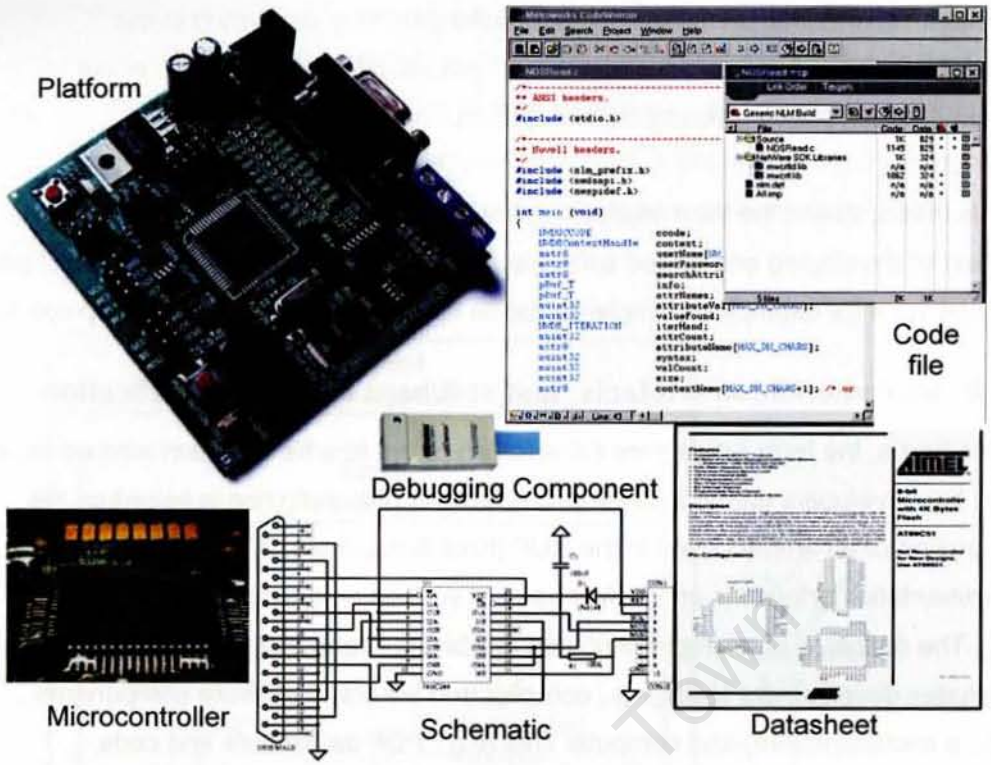


Figure 1.2: Examples of implementation artefacts[2].

1.1.7 ESAOA activities

Embedded system artefact organisation and adaptation (ESAOA) involves the classification and structuring of implementation artefacts, collectively termed acts of *artefact organisation*, and actions of physically manipulating or creating implementation artefacts en route to making the artefacts part of a product or product prototype, which is referred to as *artefact adaptation*.

Since ESAOA activities are defined to relate specifically to implementation artefacts, task-oriented ESAOA activities can generally be considered as part of an implementation task, and therefore relate to the implementation phase of the development project. As stated earlier (see Section 1.1.5), the classification of a project into phases is abstract and not absolute, which implies that a particular ESAOA activity may extend over multiple phases of the development process; similarly, an ESAOA activity may have an impact on multiple phases of a development process.

---

[2] Images obtained from http://www.atmel.com, http://www.freescale.com, http://www.macraigor.com and http://www.piconomic.co.za.

University of Cape Town

ESAOA activities can be separated into two broad categories: 1) artefact organisation (AO) activities, and 2) artefact adaptation (AA) activities. Table 1.1 lists examples of ESAOA activities, indicating the AO and AA classifications, together with the higher-level implementation tasks to which they relate.

Table 1.1: Examples of ESAOA activities for certain implementation tasks.

High-level Implementation task	Artefact Adaptation (AA) activities	Artefact Organisation (AO) activities
Creating start-up code for an embedded software program	Creating a new C file to hold the start-up code.	Making a directory called 'Code' to hold code files.
	Adding lines of code to the new C file.	Saving the new file as 'start.c'.
Writing a device driver for a temperature sensor	Changing the code of the driver so that a different value is written to a hardware register.	Copying an existing device driver module and naming it 'tempsensor.c'.
	Creating a file for use in regression testing of the device driver.	Placing the regression testing file in the directory 'test'

Source: adapted from development activities described by Berger [2002] and Fowler [2007].

1.1.8 Data, Information and Knowledge

Data, information and knowledge are usually understood as forming a hierarchy, commonly referred to as the 'D-I-K hierarchy' (see Section 2.4.1), in which there is a process through which data lead to information, and information becomes knowledge [Groff & Jones, 2003]. Data exist at the lowest level of the hierarchy, and have no significance besides their existence [Davenport & Prusak, 2000]. Bits, numbers and characters are elements of data. Information is formed by giving context and meaning to data. The particular context determines the way in which data are assembled into information. Knowledge exists in the minds of people, and occurs when information is combined with understanding and capability [Groff & Jones, 2003]. Capability is a person's ability to take action. A person's understanding is the way in which that person interprets information. Understanding and capability are mutually dependent: understanding is built through learning new ways to take action, which develops capability. Conversely, by performing actions, made possible through capability, information can be revealed to build understanding. Section 2.4.1 elaborates upon the D-I-K hierarchy by means of a scenario and a discussion of different opinions concerning this view of knowledge.

University of Cape Town

1.1.9 Knowledge management

Knowledge management (KM) takes place across multiple disciplines, and is used in a wide range of applications [Wilson, 2002]. KM should be considered as a movement with specific values, rather than a discipline [Bennet & Bennet, 2004].

There are many interpretations of what KM is. Generally, KM describes processes for collecting data, formulating information and using knowledge [Davenport & Prusak, 1998]. In general, there are two types of knowledge that need to be managed: explicit knowledge and tacit knowledge [Nonaka & Takeuchi, 1995]. Knowledge assets can therefore be tangible (e.g., documents) or intangible (e.g., experience). This thesis is concerned with tangible, rather than intangible, knowledge assets.

As is the case with 'knowledge', KM is explained by different people in different ways. Broadly speaking, there are two contrasting streams of KM [Ortenblad, 2007]. The first is the 'knowledge-focused stream', whereas the second stream is the 'information-focused stream' [McDermott, 1999a]. Both steams study the ways in which knowledge is created, captured, stored, and shared in an organisation [Davenport & Prusak, 1998]. Section 2.4.2.1 further explains these streams and what is meant in this case by an organisation.

This thesis uses a definition of KM that is based on work by Drucker *et al.* [1998], and follows the information-focused stream of KM, namely:

> KM is the way knowledge workers (of which embedded software developers are an example) create, capture, store and share knowledge in an organisation.

Using the above definition, ESAOA KM can be viewed as a method to treat information, artefacts related to the information, and the interaction between people interested in that information, in a context of ESAOA activities performed during the development of embedded software.

1.1.10 Knowledge management systems

A knowledge management system (KMS) encapsulates the way in which people, processes, and artefacts work together to create, capture, store, and share knowledge [Drucker, 1998]. A KMS thus involves the people, processes and artefacts

University of Cape Town

that comprise knowledge work, as well as including how these elements work together to support the creation, capture, storage and dissemination of knowledge [Holsapple, 2003; Maier & Hadrich, 2006]. A KMS is dependent on the type of knowledge work being done, and the context in which this knowledge work is performed. The study of a KMS typically focuses on the ways in which the 'know-how' and 'know-who' of knowledge workers is managed in an organisation to solve problems, make decisions, learn facts, and find the knowledgeable people in certain specialised areas within the organisation (Sections 2.6 and 2.7 elaborate on the research literature concerning KM within engineering fields).

Based on the above description, this thesis defines an ESAOA KMS as follows:

> An ESAOA KMS is an effective system that coordinates the way in which people, processes, and artefacts work together to creature, capture, store and share knowledge, in a context of ESAOA activities performed during the development of embedded software.

University of Cape Town

1.2 Rationale: a KMS for ESAOA activities

Recent studies have shown that the completion of embedded software development projects is frequently late and over budget [Child, 2001; Grenning *et al.*, 2004; Hall *et al.*, 2005; Charette, 2005; Ganssle, 2007]. Embedded software development is often affected by the same problems that affect traditional non-embedded software [Grenning *et al.*, 2004], such as commercial pressures, unclear requirements, unfamiliar technologies, complex projects, miscommunication and makeshift practices – among other difficulties [Charette, 2005; Hall *et al.*, 2005]. In addition to these more traditional software problems, embedded software faces further difficulties, including limited resources, real-time timing constraints, late integration with target hardware and separate environments for program development and program execution [Grenning *et al.*, 2004].

1.2.1 The growing demand for embedded software

The number of projects that involve the development of embedded software is expected to continue growing for many years to come [Graaf *et al.*, 2003; Ganssle, 2007]. The sophistication of these products is expected to increase in the future as they become more varied and ubiquitous [Ganssle, 2007]. For example, the iPhone

has combined a set of comparatively limited predecessors, namely, the cellphone, the iPod palmtop computer and an internet connection device [Apple, 2009]. As the complexity, number and diversity of ES products increases, the companies that produce these products are experiencing difficulties in achieving sufficient product quality while keeping projects on-time and on-budget [Graaf *et al.*, 2003]. In order to improve the quality, timeliness and productivity of embedded software development, companies need to adopt more effective approaches for their specific situations [Ganssle, 1999; Edwards, 2003; Grenning *et al.*, 2004].

1.2.2 The rapid expansion of knowledge

The field of computers, and particularly computer engineering, has been especially affected by the expansion of knowledge [Nienaber & Barnard, 2007]. A major challenge faced by ES engineers, and one which significantly influences the success of a project, is the assimilation and application of new technologies, development tools and methods in development projects [Linn, 2001; Graaf *et al.*, 2003], and avoiding the problems associated with information overload [Kass & Stadnyk, 1992; Lyytinen & Robey, 1999]. The exponential increases in device counts and computer performance are expected to continue for another decade and possibly longer [Kanellos, 2003]. Whether another type of technology will succeed silicon and continue this exponential trend is less clear, but Kurzweil [2001] observes that the history of computing demonstrates that, when the growth of one technology levels off, another technology emerges to continue the exponential trend. A similar exponential increase is occurring in communication system bandwidth, both wired and wireless, and this is sometimes called Moore's law for communication bandwidth [Cherry, 2004]. Consequently, embedded software is predicted to become more powerful, more complex and more interconnected [Jerraya & Wolf 2005; Jerraya, 2004; Jantsch & Tenhunen, 2003]. ES applications are expected to expand into a growing number of new areas, and as the number and power of embedded computers grows, an increasing number of novel applications are likely to emerge [Committee on Networked Systems of Embedded Computers, 2001].

ES developers thus face the challenges of rapid growth in new technologies and market demands, and the consequential need to obtain technical knowledge and create efficient development strategies so that products are aligned to market trends, and that projects remain on-time and on-budget [Graaf *et al.*, 2003].

University of Cape Town

1.2.3 Knowledge management as a potential means to facilitate embedded software development

Curtis *et al.* [1988] performed a survey of large software development companies in 1988, in which it was shown that software engineers spent significant amounts of time learning and experimenting with new technologies for use in projects. Embedded software has further demands for technical knowledge, such as acquiring knowledge of specialised operational contexts and application domains [Sangiovanni-Vincentelli & Pinto, 2005].

The role of programmability (and software adaptability and specialisation) has expanded further, placing greater demands on embedded software developers to assimilate and apply knowledge related to new technologies, such as system-on-a-chip (SoC) components [Sgroi *et al.*, 2001], software frameworks [Fayad *et al.*, 1999], and intellectual property (IP) blocks [Keutzer, 2002]. Embedded software engineers are thus faced with the compounding challenges of meeting demands for new and increasingly complex products, while obtaining the knowledge needed to make use of the new technologies and identifying appropriate development methods to develop software for these products [Berbers, 1999; Douglass, 2000; Berger, 2002].

Recent studies of ES projects indicate that the application of new technologies, while adhering to projects' time and budget constraints, remains a challenge for software engineers [Komi-Sirviö *et al.*, 2002; Graaf *et al.*, 2003; *Ko et al.*, 2007b]. There are many strategies which address this problem, such as: more comprehensive integrated development environments (e.g., MPLAB [Microchip, 2008]), easier to use embedded operating systems [Baskiyar & Meghanathan, 2005], simulator advancements (e.g., the Crossware ARM simulator [Crossware, 2009]) and model-integrated solutions [Karsai *et al.,* 2003], to name a few. While these advancements can significantly speed up the process of development, these solutions are not necessarily available to all developers (e.g., due to high licensing costs or incompatible architectures); moreover, these tools also take time to learn.

Knowledge and the practice of effective knowledge management (KM) methods have been acknowledged as essential to successful development projects by many prominent authors, such as Davenport [1998; 2002], Drucker [1998], Nonaka & Takeuchi [1995], Sveiby [1997], and Senge [1990]. These authors have also recognised that KM, like knowledge itself, is partly dependent on the context in which

University of Cape Town

it is used, as well as on the knowledge artefacts (or objects) with which it is used [Knorr-Cetina, 1997; 1999; Arias & Fischer, 2000; Knorr-Cetina & Brugger, 2002]. For example, a specialised form of development project needs a specialised form of KM, which typically involves the process of adapting a selection of existing KM approaches and establishing new practices [Groff & Jones, 2003; Firestone & McElroy, 2005]. Improvements to the KM practices used by software engineers during the acquisition and use of technical knowledge has been identified as a means to increase the success rate of these software development projects [Dingsøyr & Conradi, 2002; Komi-Sirviö *et al.*, 2002; Rus & Lindvall, 2002].

1.2.4 The need for an ESAOA KMS

Based on the current literature, KM research initiatives specific to embedded software development are less common than those related to other forms of software development[3]. The literature shows that KM studies related to embedded software development are generally focused at a high level of project management, related more to broad issues of development and its impact on the products (e.g., [Hahn & Subramani, 2000; Davenport, 2002; Dingsøyr & Conradi, 2002; Rus & Lindvall, 2002]), or collaboration and organisational learning techniques (see Section 2.8.4 for details). Organisational activities occur at the macro, meso, and micro level [House *et al.*, 1995]. The literature predominantly covers KM at the macro level of software development, which comprises processes that operate at a strategic level. There is also a growing literature on the micro level of KM, such as expert networks, web portals, document and content management tools [Lindvall *et al.*, 2001]. However, there is little literature concerning meso level KM of embedded software implementation activities. Meso level tasks involve the integration of macro and micro processes [Rousseau & House, 1994], such as approaches that facilitate the way in which embedded software developers experiment with code, organise data and files, and learn how to modify components use a range of development tools. These meso level tasks can account for a significant portion of the time an embedded software developer spends on a project (as discussed in Section 1.2.3). ESAOA activities are at the meso level of the product development process, and may therefore have a potentially significant effect on the progress of development. For these reasons, the researcher decided to investigate the KM of implementation phase development activities, with the intention of addressing this research gap in the field of KM for embedded software development.

[3] Based on a comparison of search results using Google Scholar (see Appendix E1), the combined result for embedded software was 4.2% of those for more general software.

University of Cape Town

1.3 Thesis Objective

This thesis builds on existing work in the field of KM for software development [Rus *et al.,* 2001; Trimble, 2000; Rus & Lindvall, 2002; Dingsøyr & Conradi, 2002]. As the preceding discussion has identified, embedded software is highly knowledge intensive [Ganssle, 1999; Ball, 2002] and dependent on a variety of complex knowledge artefacts, such as new types of development tools and components that result from the new technologies [Kettunen, 2003].

The broad objective of this thesis is an explorative investigation of KM within a specific area of embedded software development, namely certain forms of activity that are mainly performed during the implementation phase of these projects. These activities are referred to as embedded system artefact organisation and adaptation (ESAOA) activities. ESAOA activities involve organising (i.e., classifying and structuring) implementation artefacts and the adaptation of these artefacts during a development project (see Section 1.1.7).

The specific objective of this thesis is the construction, evaluation and evolution of an experimental KMS, referred to as the ESAOA KMS, with the intention of determining an effective structure for the implementation of such a KMS that will facilitate knowledge production to promote successful completion of ES implementation tasks. The ESAOA KMS is intended for use in ESAOA activities within the context of new projects that involve prototyping novel ES products. The ESAOA KMS is expected to incorporate KM strategies that are both more visible and more systematically applied during a project, than is the case for an *ad hoc* KMS that evolves naturally during a project (see Figure 1.3). The ESAOA KMS also needs to incorporate an analysis system to measure the performance of KM operations performed by users of the KMS. The main research activities in this study consequently include studying *ad hoc* KM strategies used by novice ES engineers, and refining these initial methods to create a more visible and refined KMS applied consistently by subsequent groups of novice ES engineers. See Chapter 3 for details.

University of Cape Town

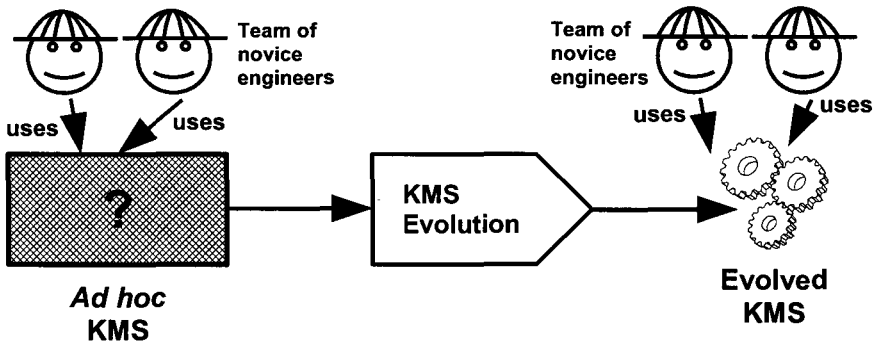


Figure 1.3: Thesis objective – evolving a KMS for ESAOA activities.

1.4 Problem Statement

This thesis argues that new projects that involve the development of novel ES products can be facilitated by the application of a specialised KMS, namely the ESAOA KMS, applied within the context of ESAOA activities. These ESAOA activities are performed by embedded software developers mainly in the implementation phase of a project (see Section 1.1.5).

The investigation is restricted to the specific context of *novice engineers* (see Section 3.6.4) working on newly initiated ES projects that concern the development of ES prototypes. Section 1.6 provides further details on the delimitations of this study.

This thesis uses the definition that knowledge is understood to reside within the mind of an individual [Polanyi, 1958; Grant, 1996; Davenport & Prusak, 1998] (see Section 1.1.8). Based on this definition, knowledge itself is thus difficult to measure, because it is defined as existing exclusively within a person's mind [Polanyi, 1958]. Consequently, the emphasis of this problem statement, and the thesis as a whole, is not on knowledge itself, but rather on knowledge-based activities, the development of an ESAOA KMS, and the effect such a KMS would have on the creation of products.

From this research project, the following overarching research question, developed after an investigation of the literature and a *preliminary study*[4] (see Section 4.2), was used to guide this research:

[4] Since this is an exploratory study, the research question and analysis methods were developed after an initial investigation of ES development practices, which was the original objective of the preliminary study. Thus, the same Experiment 1 data was used in both the preliminary study and in the initial data analysis – see Chapter 4).

University of Cape Town

Research question:

What is an effective structure for the ESAOA KMS (i.e., the roles, activities, artefacts, etc.) that will contribute to the successful completion of ES implementation tasks?

The research design developed to address this research question is the topic of Chapter 3.

1.5 Focus

This thesis focuses on ESAOA activities performed by embedded software developers during new product development projects. The focus is further refined in the following three ways:

- New product development: studying projects that involve the development of new products;
- Product prototyping: investigating projects that aim to produce a product prototype as part of a proof-of-concept project; and
- Component integration: examining ESAOA activities related to writing, or learning how to write, component integration code.

The subsections that follow elaborate upon this focus.

1.5.1 Focusing on new product development

In a study of European ES development companies performed by Graaf *et al.* [2003], it was found that the companies generally started new projects that were based on previous projects. Thus new projects tended to reuse resources (such as requirements specifications and development tools) from previous projects. While companies that have developed ES projects previously are likely to reuse previously developed artefacts and tools, as shown by Graaf *et al.* [2003], this thesis focuses on new product development in which a development team starts a new project to produce a new product that is not based on an upgraded version of a previous product. Developers working in either the context of building from scratch or reusing previous artefacts are likely to encounter similar issues; however, teams that build on previous projects have many advantages, such as the use of systems, procedures artefacts and tools left behind from previous developers – an aspect highlighted by Leonard-Barton [1992] in a study of multiple development projects. In this thesis, the focus is on teams comprising first-time developers who need to develop project

University of Cape Town

resources from scratch and who do not have access to previous projects or project resources that they can reuse[5].

1.5.2 Focusing on product prototyping during a proof-of-concept

The objectives of development products can vary. For example, the aim of production projects is to produce a quantity of marketable projects; whereas the purpose of research and development (R&D) projects relates more to experimentation, establishing development methods [Brinkkemper *et al.*, 1996], and exploration of concepts [Lewis, 2006]. A proof-of-concept project involves the possibly incomplete creation of a product concept; these projects aim to demonstrate the feasibility of a product idea, or to verify that an aspect of the concept is feasible [Erdogmus, 2002]. This thesis focuses on proof-of-concept projects in which an ES prototype is built. A proof-of-concept project (more so than a larger scale R&D project) generally has a specific concept to test and a corresponding plan for an experiment to perform [Erdogmus, 2002] (whereas part of an R&D project may involve conceptualising a concept to test). An ES product prototype built as part of a proof-of-concept project may be functionally limited (in contrast to a marketable version of the product) and intended only to test certain design strategies [Floyd, 1984] (Section 1.1.2 discusses ES prototypes).

1.5.3 Focusing on ESAOA activities related to component integration

The implementation phase of product development involves the integration of a component into an incomplete product using development tools, or the application of development tools to make adjustments to the way in which a component is integrated into a product, or part of a product [Schach, 2005]. The term 'component' refers to the parts from which a product is built (a component can be considered to be a special form of implementation artefact). Development tools include software or hardware tools that embedded software developers use in projects; examples include web browsers, scripting tools, cross-compilers and remote debuggers [Sutter, 2002].

ES product development can be viewed as a process of forming a product from a set of hardware and software components [Marwedel, 2003]. A component can be a hardware component (e.g., a microcontroller) or software component (e.g., a code module). In this thesis, the term *component integration* refers to a process by which a

[5] The teams have access to resources on the Internet in the public domain.

University of Cape Town

software developer writes or adapts code to connect components. This study focuses on ESAOA activities related to component integration.

While component integration may overlap both the implementation phase and the integration phase of a project, the focus remains on activities related to implementation tasks, more specifically activities in which the software developer modifies code components[6]. Component integration studied in this thesis includes connecting software components with other software components, and connecting software components to hardware components[7] (e.g., creating or modifying a device driver). The connection of hardware components to other hardware components does not fall within the scope of this study.

1.6 Delimitations

The focus of this study, as elaborated upon in Section 1.5, is on KM for ESAOA activities performed by software developers in the context of implementation tasks performed in new proof-of-concept development projects in which an ES product prototype is constructed (Figure 1.4 visually models the focus and delimitations of this study).

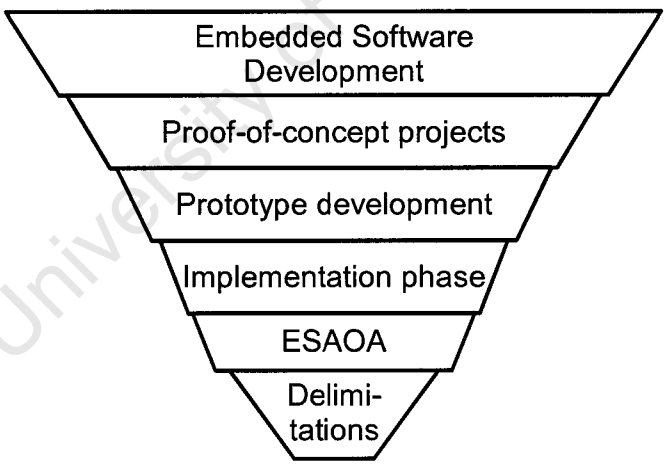


Figure 1.4: Visualization of the thesis focus and delimitations.

Although the focus narrows down the research area, a wide variety of possible projects still fit the research focus. For example, such projects could be done entirely by an individual developer, alternatively by a highly proficient team of professional

[6] In this text, the integration phase can be considered as the integration of *completed* product components, and to a lesser extent the writing of code that connects components.
[7] This is generally referred to as hardware/software interfacing [Patterson & Hennessy, 2005].

University of Cape Town

engineers, or possibly by a decentralised team, among other possibilities. The delimitations outlined below indicate the specific type of development projects investigated in this study. Although these delimitations restrict the generalisation of the results produced, the study can still provide potentially useful insights, or suggest strategies that may be beneficial to the wider area of the research focus, and contribute to the knowledge in this field.

The six main aspects of the delimitation are outlined in Table 1.2, and are refined in the subsections that follow.

Table 1.2: Summary of delimitations

Aspect	Delimitation imposed
Activities studied	Task-oriented ESAOA activities only
Level of developers	Novice engineers
Team size and structure	Small teams, of two to three members each
Number of case studies	Two case studies in Experiment 1, thirteen in Experiment 2
Time-frame	Experiment 1 projects studied for three months; Experiment 2 project studied for eight months
Products developed	All prototypes built using the same hardware platform and selection of development tools

University of Cape Town

1.6.1 Task-oriented ESAOA activities

This study is focused on ESAOA activities in which component integration is performed, as explained in Section 1.5.3. Only task-oriented ESAOA activities are investigated (i.e., peripheral activities were ignored). The emphasis is on ESAOA activities used by developers while *attempting*[8] to learn about, or performing, component integration. These forms of component integration are restricted to:

- Incorporating a software artefact (e.g., a code file) into an existing application;
- Modifying code so that one code module connects to another; and
- Coding a software module to make it communicate with a hardware device.

Only activities in a single project are studied, the study does not investigate how activities in one project relate to those in concurrent, past or future project).

[8] The emphasis is on 'attempting' because all acts of component integration are not necessarily immediately successful; many failed attempts may be made before a successful strategy is discovered.

1.6.2 Level of developers

The study focuses on novice engineers, who have completed two years of a university computer engineering programme (further detail for this delimitation is given in Section 3.6.4).

1.6.3 Team size and composition

A developer's knowledge of implementation tasks results from his/her underlying skills that are essential to the type of development concerned [Jackson & Caspi, 2005; Grimheden & Törngren, 2005]. For example, in ES development, fundamental software development skills include the developer's ability to write source code in a particular programming language, and an understanding of how the source code is turned into an executable object. Members of project teams in knowledge-creating companies are often assigned certain roles, such as: a team leader, a knowledge owner, or a research specialist [Abell & Oxbrow, 1999]. Each team member can perform more than one role. Having more than one member in a team allows the potential for capturing explicit inter-person communication (e.g., email or discussion points recorded in meetings) between individuals filling specific roles (the generation of such explicit knowledge would not be as natural or automatic in the case of one-member teams). However, large teams (e.g., five or more people) can complicate the study, such as logistic problems of getting all team members to attend meetings [Milton, 2005]. Consequently, teams in this study are limited to two or three members to avoid unnecessarily complicated team structures, thus keep the research focused.

1.6.4 Number of experiments and case studies

The research design involves two experiments. The first experiment (Experiment 1) was a pilot study comprising two projects, with each project performed by two developers. Experiment 1 was an initial investigation in the chosen research field, which led to the design of ESAOA KMS version 1 (see Chapter 4). The second experiment (Experiment 2) was a larger study that involved thirteen projects, each performed by a team of three members using ESAOA KMS version 1 (see Chapter 5). Chapter 3 provides further detail on the design of these two experiments.

1.6.5 Time-frame for case studies

A requirement for this research project was to complete the acquisition of data from case studies within a two-year period. The time period for capturing data from projects was chosen to be three months for Experiment 1, and eight months for Experiment 2. The rationale for these time periods is given in Section 3.2.2.2.

University of Cape Town

1.6.6 Products developed

All projects in the study involve the construction of ES prototypes (as described in the research focus in Section 1.5). The prototypes were all built using the same evaluation board (the CSB337 [Cogent Computers, 2005]) and the same set of development tools. The teams acquired and interfaced additional external hardware and software as required. Section 3.2.2 provides detail on this delimitation.

1.7 Thesis Structure

This thesis is structured around seven chapters. The progression of the thesis, and the focus of each chapter, is as follows:

1. *Chapter 1* introduces the research area, the rationale for this research and the focus of the thesis, together with contextual and background information.
2. *Chapter 2* overviews current literature on KM, and focuses in on the research literature on KM in software engineering and related fields. A meta-analysis of the literature identifies KM-related issues, KM strategies and KM tools applied, and the impact these have on product development processes.
3. *Chapter 3* outlines the research design and research methods developed for the study and evolution of the ESAOA KMS.
4. *Chapter 4* presents the results of the first experiment, in which an *ad hoc* KMS was applied. The results showed that, when using an *ad hoc* KMS, ES developers spend more non-productive than productive time during ESAOA activities [Winberg & Schach, 2007]. There was also a greater occurrence of non-productive knowledge events than productive knowledge events. This chapter also explains the first KMS prototype (ESAOA KMS version 1).
5. *Chapter 5* presents the results of the second experiment, which used an experimental research methodology that involved testing ESAOA KMS version 1. The results indicate that the application of ESAOA KMS version 1 caused more productive knowledge events than non-productive knowledge events than did the first experiment.
6. *Chapter 6* develops the structure of ESAOA KMS version 2 based on the findings from the first and second experiments.
7. *Chapter 7* concludes the thesis, presenting recommendations and areas for further research.

University of Cape Town

1.8 Summary

This thesis studies a specific aspect of ES development, referred to as ESAOA. It is motivated by the growing demands for embedded software and the need for improvements to ES development projects so that more of these projects can be completed on-time and on-budget [Child, 2001; Grenning *et al.*, 2004; Hall *et al.*, 2005; Charette, 2005; Ganssle, 2007] (see Section 1.2). KM is a potential means of addressing these problems (see Section 1.2.3). The specific objective of this thesis is the construction, evaluation and evolution of an experimental KMS, referred to as the ESAOA KMS. This KMS is intended for use in ESAOA activities within the context of new projects that involve prototyping ES products. This thesis centres on how ES projects are facilitated by the application of the ESAOA KMS. This research study is focused on new product development that involves proof-of-concept projects performed by novice engineers. Further delimitations are applied to narrow the study (see Section 1.6). The next chapter (Chapter 2) provides a literature review that overviews the current literature and builds towards the research design (Chapter 3).

University of Cape Town

University of Cape Town

Chapter 2:

Literature review: Knowledge management and embedded system engineering

This chapter begins, in Section 2.1, by describing the focus of the literature review and the methodology used in its construction. A funnelled approach is then followed, in which Section 2.2 reviews fundamental terminology and theories related to embedded system (ES) theories, embedded software development, and knowledge management (KM). Section 2.3 elaborates on the thesis rationale outlined in Chapter 1 – highlighting the inefficiencies of embedded software development and KM methods as a means of addressing the problems of information overload and the management of technical knowledge. Section 2.4 discusses general theory and terminology related to KM, clarifying how these terms should be interpreted in the context of this thesis. Section 2.5 presents a topology of KM that maps out and defines major categories of KM based on current research reported in the literature. Section 2.6 details the concept of a knowledge management system (KMS), highlighting the difficulties and principles according to which these types of systems are designed and deployed. Section 2.7 concerns roles that people take on when involved with a KMS. In Section 2.8, the literature review hones in on research performed in the disciplines of electrical engineering and computer science, reviewing state-of-the-art studies that are related to KM as a potential means of reducing information overload and improving knowledge acquisition. Section 2.9 draws on the earlier sections of this chapter to develop the theoretical framework that is used in the research design (see Chapter 3). Section 2.10 ends with a brief summary and conclusion of this chapter.

University of Cape Town

2.1 Methodology of the literature review

The literature reviewed was separated into six parts, with each part focusing on a specific topic related to the positioning of this thesis within a general field of research. Sections 2.2 to 2.8 correspond to the topics reviewed, namely:

1. ES development processes (Section 2.2);
2. Inefficiencies of ES development (Section 2.3);
3. General knowledge management concepts and terminology (Section 2.4);
4. Typology of KM strategies (Section 2.5);
5. Knowledge management systems (Section 2.6);
6. Roles of people involved with a KMS (Section 2.7); and
7. Use of KM in embedded software development (Section 2.8).

The literature reviewed for topics 1, 3, 5 and 6 above was based predominantly on books, as these are fundamental concepts. The literature investigated for topics 2, 4 and 7 included mainly journal articles, conference proceedings, and reports; the focus was on publications after 2000.

Topic 7, which narrows down the focus on studies that deal with KM in technical product development contexts, is the most detailed part of the literature review. This part is most closely related to the specific area of this thesis. The literature chosen for this part was required to be empirically based and to be as recent as possible.

For each part of the literature review, the information searches that were done captured many articles describing empirical research. Most of these were of an evaluative nature that concerned the implementation and effects of using KM in contexts of technical product development. The conference papers and journal articles generally directed the researcher towards book-length studies, book databases (e.g., IEEE Press), technical reports, and Masters and Doctoral dissertations (these findings further motivated parts of the literature review to focus on these types of sources).

Extensive internet searches were performed using the ACM portal, Emerald, Engineering Village, the IEEE portal, Proquest, Science Direct, Scopus, Springerlink, Taylor and Francis, and Web-of-Knowledge (ISI). Keywords used in these searches included terms such as 'embedded system', 'embedded software', 'knowledge management', 'development procedures', 'organisational learning', 'implementation' and 'techniques'. Searches of the following journal databases were also done: *Administrative Science Quarterly, AI Communications, Automated Software Engineering, Communications of the ACM, Computer, Computer-aided Design, Concurrent Engineering, Decision Support Systems, Empirical Software Engineering, European Management Journal, Expert Systems with Applications, Information and*

University of Cape Town

Software Technology, IEEE Transactions on Automation Science and Engineering, IEEE Transactions on Knowledge and Data Engineering, IEEE Transactions on Engineering Management, IEEE Transactions on Software Engineering, International Journal of Software Engineering and Knowledge Engineering, International Journal on Software Tools for Technology Transfer, Journal of Engineering and Technology Management, Journal of Management Information Systems, Journal of Systems and Software, MIS Quarterly, Organisational Science Journal, Software Engineering Journal, Software Quality Journal, The Sloan Management Review, and *Research in Engineering Design.*

Conference and workshop databases searched included the following: Proceedings of the IEEE Conference on Control Applications, Proceedings of the Joint Conferences on Knowledge-based Software Engineering, Proceedings of the International Conferences on Product-focused Software Process Improvement (PROFES), Proceedings of the International Conferences on Software Engineering Advances (ICSEA), Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE), Proceedings of the International Conferences on Software Product Lines, Proceedings of the IEEE International Conferences and Workshops on the Engineering of Computer-based Systems, International workshop on Learning Software Organizations, Proceedings of the IEEE International Conferences on Services Computing (SCC), Proceedings of the International Conference on System Sciences, Proceedings of the International Workshops on Principles of Software Evolution, Proceedings of the IEE International Conference and workshop on the Engineering of Computer-based Systems (ECBS), Proceedings of the IEEE Annual Symposium on Reliability and Maintainability (RAMS), Proceedings of the Annual NASA Software Engineering Workshop, Proceedings of the International Conference on Knowledge Acquisition, Proceedings of the International Conference on Modelling and Management, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Proceedings of the International Conference on Management of Information and Communication Technology, and Proceedings of the International Conference on Software Engineering (ICSE).

A review of the articles that met the inclusion criteria was performed, focusing on the issues identified for KM, the KM strategies or systems adopted, the tools used for KM, and the effect of KM strategies and tools. There were very few studies that focused on specialised, or adapted, forms of KM for use by ES engineers as a

University of Cape Town

means of defining, guiding and improving the way in which developers address knowledge-related difficulties associated with ES product development, although there were studies of KM in related fields such as software engineering and information and communication technologies (e.g., [Abrahamson *et al.*, 2003; Lindvall *et al.*, 2004]). The existing KMS literature, even when it is located in technical contexts, tends to address project management rather than KM (as defined in Section 2.4.2), and also tends to focus more on business processes rather than development; however, many of the strategies and tools developed for collaboration, knowledge sharing, and other knowledge-related activities can be adapted to KM needs (as is shown in Section 2.8.2).

2.2 The ES development process

At a high level, the design process of an ES development project can be divided into seven main phases (as is shown by Berger [2002] and by Mäntyniemi *et al.* [2004]). Figure 2.1 shows an adapted view of the waterfall model [Schach, 2005], which reflects an amalgamation of the ES development models described by distinguished authors in the field, namely models by Labrosse *et al.* [2008], Fowler [2007], and Berger [2002]. The model shown in Figure 2.1 is not identical to the original version of the waterfall model [Royce, 1970], since a testing phase has been added in accordance with the emphasis on testing by these authors. Although the phases presented by these authors are not identical, the authors' descriptions of their models are generally consistent with the model shown in Figure 2.1, and as elaborated upon in the text below. For example, Berger [2002] uses the name 'specification' to collect activities of the first phase, whereas Labrosse *et al.* [2008] use the name 'requirements'; in contrast, Fowler [2007] chooses to name the first phase 'requirements and specifications', and he describes an ES development model that is a hybrid waterfall and spiral model. This thesis uses a variant of the waterfall model chiefly as a means of clarifying the position of this study.

As Figure 2.1 shows, an ES project generally starts by gathering requirements for the product to be developed. This 'requirements' phase usually involves defining and documenting the features and functionality of the product [Labrosse *et al.*, 2008]. In this phase, requirements are commonly documented in a rapid and informal manner [Schach, 2005].

University of Cape Town

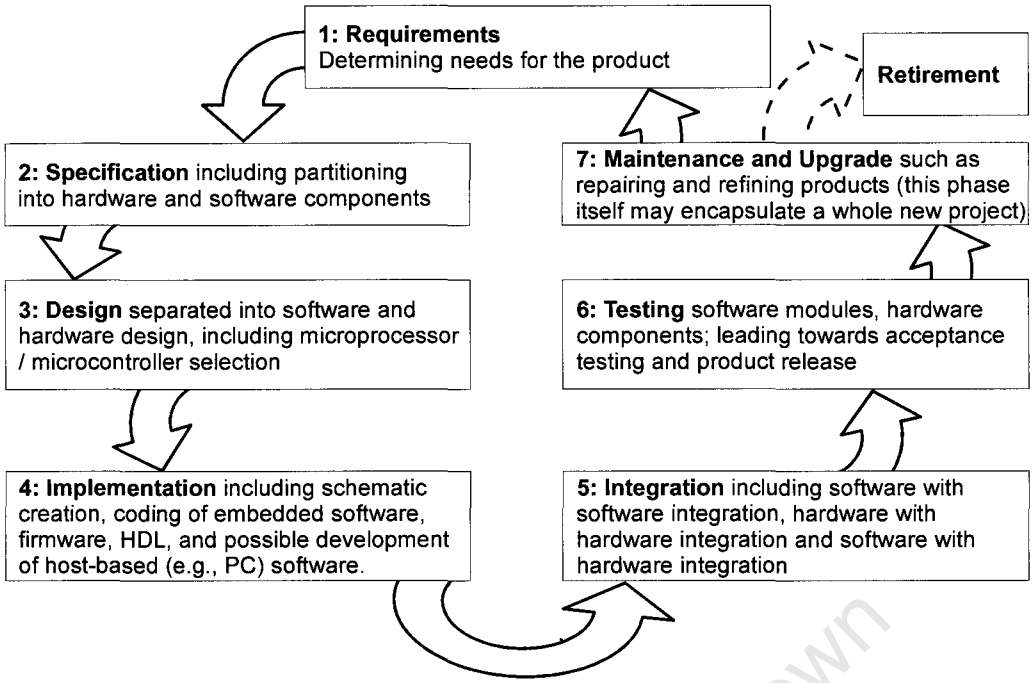


Figure 2.1: Embedded system lifecycle model (an adaptation and amalgamation of models by Schach [2005], Labrosse *et al.* [2008], Fowler [2007] and Berger [2002]).

The specification phase can be seen as building on the requirements phase, in which formal documentation is produced that explicitly, and precisely, describes the functionality of the product, often including a description of the specific inputs and outputs of the system, as well as of the constraints on the product operation [Schach, 2005]. The specification phase may also include the development of a particular system architecture for the ES to be built [Labrosse *et al.*, 2008].

The design phase involves refining and developing the structure of the product, essentially working at an abstract level to determine how the product is going to provide the functionality required [Schach, 2005]. In terms of ES design, this is often divided between software and hardware design. The hardware design aspect includes activities such as developing schematics for printed circuit boards and the selection of hardware components (e.g., displays, buttons and sensors). The software design aspect may include UML modelling, and deciding which boot loader and operating system to use. The selection of microcontrollers or microprocessors to use in a product is an important decision, one that should arguably be made as early as possible, since it has a far-reaching impact on the development process (i.e., in terms of choosing design tools and the system architecture) [Berger, 2002]. Accordingly, the design phase is sometimes split into two parts; for example,

University of Cape Town

Labrosse *et al.* [2008] indicate that processor selection should take place before the design phase starts.

The implementation phase can be separated into hardware aspects and software aspects [Labrosse *et al.*, 2008]. Software implementation involves writing code for the embedded software, including application code, driver modules, boot loaders, configuration files, and scripts – all of which may be needed to create and install executable programs that the product will use [Fowler, 2007]. Typically, in order to implement embedded software, the developers need to learn how to use a 'cross-compiler tool chain' [Ganssle, 2007]. A tool chain is a set of tools used to develop software; these tools are often applied in sequence so that the output of one tool feeds into the input of the next [Wiktionary, 2008]. A cross-compiler tool chain is a tool chain capable of creating executable code for a platform that is different to the one on which the tool chain is run.

The hardware aspect includes activities such as writing HDL code and fabricating printed circuit boards [Shetler, 1996; Riesgo *et al.*, 1999]. As mentioned in Section 1.5.3, this thesis focuses on implementation phase activities, with some overlap of integration issues.

The integration phase occurs at three levels: 1) the integration of software with software (e.g., connecting code modules together); 2) the integration of hardware with hardware (e.g., physically connecting a microcontroller to an external peripheral); and 3) the integration of software with hardware (e.g., making software work with hardware) [Labrosse *et al.*, 2008].

Testing generally happens as an ongoing process (e.g., compiling and testing code as it is being developed). The testing phase can range from quick and simple tasks (e.g., compiling and running a code module on a PC) to more elaborate and time-consuming undertakings (e.g., a factory acceptance test [Fowler, 2007]).

The maintenance and upgrade phase occurs after the product has been accepted by the client [Schach, 2005]; studies have established that this phase commonly involves the most effort and engineering expense for a development company [Sommerville, 2006]. The terms 'upgrade' and 'maintenance' tend to be used interchangeably in the literature; however, the term 'maintenance' generally relates to fixing an existing product [Fowler, 2007], whereas 'upgrade' tends to refer to the

University of Cape Town

development and release of a new version of an existing product [Berger, 2002]. The team that performs the maintenance and upgrade of a project often differs from the team that originally developed the product, which is one of the difficulties inherent in this phase of the development lifecycle [Berger, 2002].

2.3 Inefficiencies of ES development

Recent studies have shown that embedded software development projects are frequently late and over budget [Child, 2001; Grenning *et al.*, 2004; Hall *et al.*, 2005; Charette, 2005; Ganssle, 2007]. The number of embedded software projects is also expected to continue growing [Graaf *et al.*, 2003], and the complexity of these products is likely to increase further [Ganssle, 2007]. These trends were highlighted in Section 1.2 as motivation for the need for more efficient approaches to embedded software development.

Many factors contribute to the inefficiencies of ES development. This section focuses on five of the major causes that, based on the recent literature, may be made more efficient by using appropriate KM techniques. The causes focused on are: 1) general software engineering difficulties; 2) complex and lengthy learning processes; 3) the value and temporality of intellectual capital; 4) decentralised development, speed of obsolescence and availability of new technology; and 5) embedded software maintenance issues. These points are elaborated upon below.

2.3.1 General software engineering difficulties

The inefficiency of embedded software projects are caused by many of the difficulties that are found in non-embedded software development [Grenning *et al.*, 2004], for example, dealing with vague requirements, makeshift practices and unfamiliar technologies [Charette, 2005; Hall *et al.*, 2005]. However, embedded software developers also need to address additional challenges, which are not usually faced by non-embedded software developers, such as working with resource limited platforms, and using separate environments for software development and program execution [Grenning *et al.*, 2004].

2.3.2 Complex and lengthy learning processes

Added to the issues above and a likely contributing factor to the problems such as time and budget overruns, are the difficulties with regard to learning and knowledge acquisition found in embedded software development. For instance, an engineer

University of Cape Town

generally does not know the particular implementation methods to use in a project at the start of a project [Kitchenham *et al.*, 1995; Ganssle, 1999]. Rather, the implementation methods tend to develop as the engineer's knowledge of the target hardware, development tools and other development artefacts grows, as the project progresses. This knowledge often develops from time-consuming activities, such as reading datasheets, finding examples of code, discussing solutions with colleagues, and experimenting with code [Ganssle, 1999; Labrosse *et al.*, 2008].

2.3.3 The value and temporality of intellectual capital

Intellectual capital (IC) is the most valuable asset of a software development organisation [Rus & Lindvall, 2002], as is the case for many other technology-based enterprises [Sveiby, 1997]. A major difficulty for a software development firm is that the individual engineers do not necessarily stay at the same firm; when an engineer leaves the firm, a portion of that firm's knowledge leaves as well [Rus & Lindvall, 2002]. Knowledge management is seen as a means of assisting with these types of problems.

2.3.4 Decentralised development, speed of obsolescence and availability of new technology

Software development practices are changing rapidly [Herbsleb *et al.*, 2001; Rus & Lindvall, 2002]. For instance, work performed on a project is being done by engineers working on different phases simultaneously, as well as by people working in different parts of the world [Von Krogh *et al.*, 2003]. In these situations, developers have additional difficulties, such as locating and sharing technical knowledge, which provides further motivation for more effective KM in software development [Rus & Lindvall, 2002].

2.3.5 Embedded software maintenance issues

Factoring maintenance into the development process of an ES product is important, because changes to a product and its embedded software, after it has been delivered and installed, are often a necessity [Lindvall *et al.*, 2003]. There are many reasons for the high costs and low productivity of implementing changes in any type of software, especially if appropriate allowances for maintenance considerations have not been made; common problems, based on findings by Lindvall *et al.* [2003], include the following:

University of Cape Town

1. The personnel that maintain the system are often not the ones who originally developed the system.
2. In some cases, maintenance is performed by developers who are relatively inexperienced and unfamiliar with the application domain.
3. Maintenance tends to have a poor image among software engineers; as a result, it is often seen as a less advanced process that is allocated to junior staff.
4. The software being maintained is often poorly documented or the provided documentation is inconsistent with the code used.
5. In some case, a separate team, which does not include the original designers, produces the documentation.
6. The structure of the modified software tends to degrade, as further changes are made to it (making further changes increasingly more difficult).
7. Changes are often not carried out in a consistent and guided manner.

Many of the difficulties described above could be facilitated by appropriate KM techniques that account for maintenance needs. Especially valuable would be methods that help maintenance workers to learn previously established development methods, and to avoid making the same mistakes as those made by the original developers while formulating these methods [Dingsøyr & Conradi, 2002].

University of Cape Town

2.4 Knowledge Management Terminology

This section focuses on defining important terms related to KM. Many of these terms are not unanimously agreed upon in the literature, as evidenced below. These terms are defined according to their application in this thesis, highlighting interpretations of the terms where relevant. This section builds outwards from fundamental definitions of data, information and knowledge, towards the increasingly complex notions of KM and knowledge flows that lead to a typology of KM approaches (in Section 2.5).

2.4.1 The Data, Information and Knowledge (D-I-K) Hierarchy

As outlined in Section 1.1.8, data, information and knowledge can be seen as a hierarchical process, through which data lead to information, and information becomes knowledge. This hierarchy is commonly termed the D-I-K hierarchy [Patrick, 2008; Rennolls & Al-Shawabkeh, 2008; Tian *et al.*, 2009]. This notion of how

knowledge is formed is used in this thesis to elaborate on issues of KM and the acquisition of knowledge.

Data exist at the lowest level of the hierarchy, and have no significance besides its existence [Davenport, 2002]. Bits, numbers and characters are elements of data. Information is formed by giving context and meaning to data. A particular context determines the way in which data are assembled into information. Knowledge exists in the minds of people, and occurs when information is combined with understanding and capability [Groff & Jones, 2003]. Capability is a person's ability to take action. A person's understanding is the way in which that person interprets information [Madhavan & Grover, 1998]. Understanding and capability are mutually dependent: Understanding is built through learning new ways to take action [Kogut & Zander, 1992], which develops capability; conversely, by performing actions, which are made possible through capability, information builds understanding.

2.4.1.1 Data, information and knowledge scenario

Figure 2.2 shows a scenario in which the differences between data, information and knowledge are illustrated (the scenario is an adaptation of one by Groff and Jones [2003]). In this scenario, data exist in the form of twelve numbers. These numbers relate to a bank account, with account number 1234567, and transactions affecting this account. Some of the transactions are withdrawals, and are thus classified as debits, while others are deposits, and thus classified as credits. These remaining numbers reflect the balance in the bank account after a transaction has occurred. Since the transactions follow a certain sequence, they are ordered according to the time at which they occurred. This procedure of classifying and organising data elements gives the data context, and turns it into information. The specific process involved depends on what the information is likely to be used for. In this case, the bank statement belongs to an individual, named John, who plans to use the information as part of a procedure for withdrawing cash from his bank account. For this objective, John needs specific levels of understanding and capabilities to interpret the information provided on his bank statement. He needs to understand how to read a bank statement, and he also needs to know the procedure of withdrawing cash from an ATM. He also needs the capability to withdraw cash from an ATM (e.g., having access to an ATM and being in possession of his cash card).

University of Cape Town

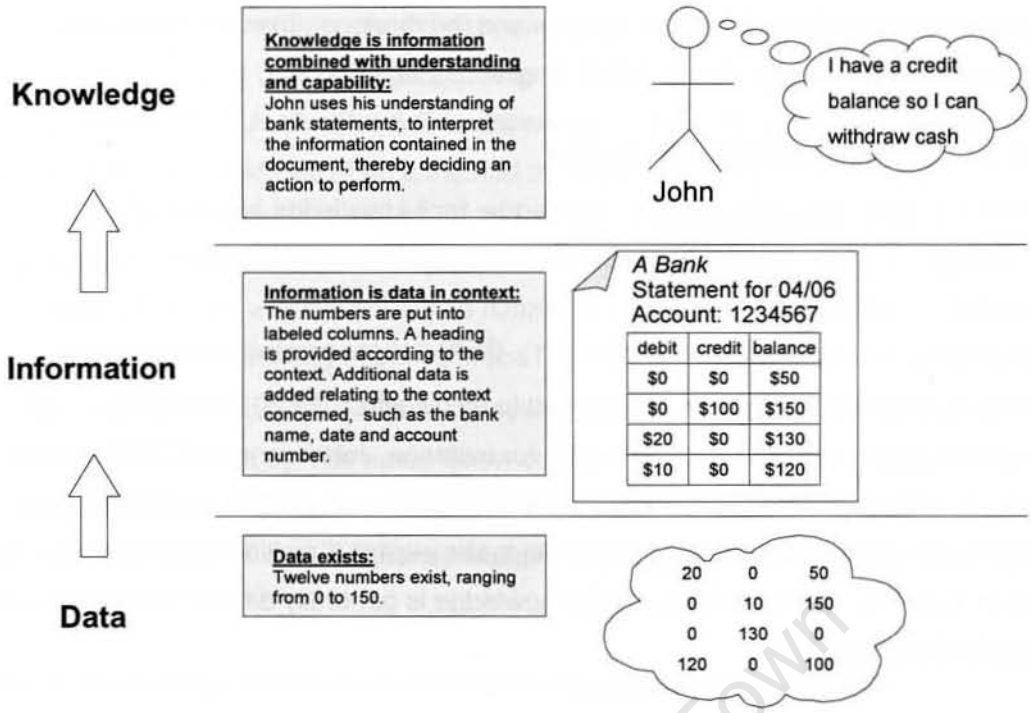


Figure 2.2: The data, information, and knowledge hierarchy, with scenario illustrating how data in the form of numbers become information, which leads to actions (adapted from Groff & Jones [2003]).

University of Cape Town

2.4.1.2 Knowledge acquisition and limitations of the D-I-K hierarchy

Although the hierarchy shown in Figure 2.2 indicates that knowledge is built from information, it must be noted that knowledge is by no means derived directly from the availability of information, nor is it derived from information alone [Patrick, 2008]. Knowledge is gained through a complex interaction between ideas, concepts and thought patterns, which may be coupled with information, prior experience, personal needs, and many other factors [Bennet & Bennet, 2004]. Accordingly, knowledge can be considered as a phenomenon that emerges through complex patterns, and is dependent on factors such as an individual's objectives, and the context concerned, in addition to the existence of information. It should also be remembered that the D-I-K hierarchy can be challenged (as shown by Bellinger [2000]).

Data consist of unsystematised information. There are forms of data, however, that are extremely complex and that require complex information systems for purposes of data management [Espinosa *et al.*, 2007]. Similarly, there are forms of data that are fairly simple. The link between data, information, and knowledge is not necessarily always hierarchical; nevertheless, it can be said that the more complex the data is,

the more complex the information system and the more complex the knowledge forms will be [Galliers & Newell, 2001; English, 1999].

2.4.1.3 Tacit and explicit knowledge

There are two major categories of knowledge: tacit knowledge and explicit knowledge. Tacit knowledge is personal knowledge that is embedded in individual experience, relating to intangible factors such as the individual's personal values, perceptions and beliefs [Polanyi, 1958]. Tacit knowledge has been described as being more about 'know-how' as opposed to 'know-what' (facts), 'know-why' (science) and 'know-who' (networking) – for instance, Johnson *et al.* [2002] discuss that it is easy to write down rules for playing tennis (know-what) and who the good players are (know-who); but it is difficult to make explicit the skilful behaviour of tennis pros (i.e., the know-how). Tacit knowledge is generally difficult to transfer [Argote & Ingram, 2000].

Explicit knowledge is codified tacit knowledge; or more precisely, explicit knowledge is tacit knowledge expressed in a formal language, such as having been spoken or documented [Nonaka & Takeuchi, 1995]. Explicit knowledge can be much easier to transfer than tacit knowledge [Argote & Ingram, 2000]. Sveiby [2001] estimates that possibly 1 percent, or less, of all knowledge can be made explicit. However, research has not shown that explicit knowledge is *always* easier to transfer than tacit knowledge. For instance, there are certain situations where tacit knowledge is easily transferred directly from one individual to another without using explicit knowledge; in such situations the communicating individuals may be close together and have similar experience levels [Dhanaraj *et al.*, 2004; Eraut, 2000].

2.4.1.4 A definition of knowledge and where knowledge resides

As this section has shown, knowledge is an abstract concept, which is expressed in the way it is used. An analysis of recent literature [Bennet & Bennet, 2004; Davenport & Prusak, 2000; Sveiby, 2001] shows a common theme for defining knowledge as information combined with capability and understanding.

A number of authors indicate that knowledge lives in the mind of people [e.g., Patriotta, 2004; Polanyi, 1958; Groff & Jones, 2003], which may imply that 'explicit knowledge' refers to a form of information since it can be stored outside a person's mind. Many authors do not add this caveat, in particular Rumizen [2002], Davenport & Prusak [2000], and Senge [1990]. It may be debatable whether or not knowledge

University of Cape Town

exists only in the minds of people; for example, an expert system mimics the way in which a human expert makes decisions based on a diagnosis and prior experience [Grundspenkis, 2007; Apshvalka & Grundspenkis, 2005]. If the implementation of an expert system is considered to be a method of codifying knowledge, then this example indicates that the codification of knowledge can lead to more than just information.

2.4.2 Knowledge management (KM)

As mentioned in Section 1.1.9, KM is a movement taking place across multiple disciplines [Wilson, 2002]. The name 'knowledge management' is therefore better considered a label rather than a descriptive title for the movement [Hahn & Subramani, 2000] and is consequently explained by different people in many different ways.

2.4.2.1 Knowledge-focused vs. information-focused streams of KM

Broadly speaking, there are two contrasting streams of KM, referred to as the 'knowledge-focused' stream and the 'information-focused' stream. Both streams involve the way in which knowledge is created, captured, stored, and shared in an organisation, with the intention of leveraging this knowledge inwards within an organisation (e.g., helping developers to understand common difficulties experienced by other teams), and outwards to people outside the organisation (e.g., assisting customers in solving problems) [Rus *et al.*, 2001; Wiig, 1997]. In these definitions, the term *organisation* can refer to a variety of different collaborations in which people are involved; it is not limited to corporate concerns, in that an organisation could also refer to a team of individuals not necessarily employed by the same company, but working towards a common purpose [McDermott, 1999b].

The knowledge stream focuses on leveraging knowledge that occurs in different forms, such as the form of knowledge encountered in meetings compared to the form encountered in solving problems [Sveiby, 2001; Nonaka, 1994].

The information stream, by comparison, is focused on methods for treating information, and the interaction of people interested in that information [McDermott, 1999a]. This form of KM involves capturing explicit information and the way in which the captured information is stored, categorised, and then recovered and interpreted to create, recall, or apply knowledge in certain contexts [Groff & Jones, 2003; Sveiby, 2001; Grant, 1996].

University of Cape Town

2.4.2.2 The overall goal of KM

The goal of KM is to make the enterprise act as intelligently as possible to secure its viability and overall success [Wiig, 1997]. Based on the definition of KM by Davenport & Prusak [1998] (see Section 1.1.9), KM can be seen as enhancing processes of sharing, distributing, creating, capturing and understanding a company's knowledge [Davenport *et al.*, 1998]. Some important aspects of KM are to survey, develop, maintain and secure the intellectual and knowledge resources of the enterprise, to determine the knowledge and expertise required to perform work tasks, to organise knowledge, to make the requisite knowledge available, to 'package' it, to 'distribute' it to relevant points of action, and to provide the 'knowledge architecture' so that the enterprise's facilities, procedures, guidelines, standards, examples, and practices facilitate and support its activities [Wiig, 1995].

2.4.3 Knowledge Processes

Radding [1998] describes the concept of a 'knowledge process', a mostly unseen relation of actions that takes place in the use and creation of knowledge. Radding [1998] describes a knowledge process as consisting of four steps: 1) capture, 2) storage, 3) processing and 4) communication. Figure 2.3 illustrates this progression of steps, accentuating the way in which these aspects tend to become increasingly complex. The points below elaborate upon each step.

1. *Capture*: The organisation (or individual) captures explicit and tacit knowledge in the form of data or higher-level information, termed 'raw knowledge'.
2. *Storage*: The captured raw knowledge (data and information) must be stored in a place, such as a data warehouse, where it can be managed, secured and made accessible to others.
3. *Processing*: Raw knowledge is transformed into valuable business knowledge during the processing step. Processing may involve sorting, filtering, organising, analysing, comparing, correlating, mining or a number of different techniques. It may involve a little more than just labelling the knowledge so that others can easily find it when they need it, or it may entail sophisticated, complex, statistical analysis to uncover hidden relationships and insights. It is here that human intuitiveness and experience come into play in making decisions, and it is here that the 'meaning' contained in that knowledge is determined.
4. *Communications*: In order for knowledge to be truly valuable, it must be shared with others. Communications can be active or passive. Knowledge

University of Cape Town

can be transmitted via information systems or passed on through personal interaction. Alternatively, it can simply be placed in an accessible storage receptacle, ready and accessible by users when they need it.

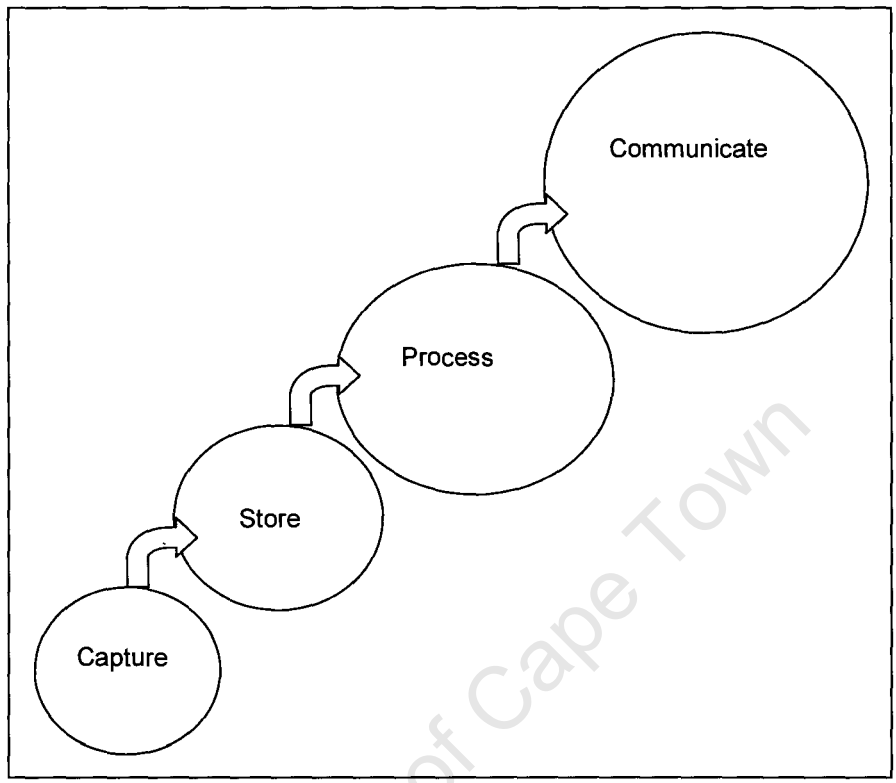


Figure 2.3: The knowledge process adapted from Radding [1998].

2.4.4 Knowledge Flows

Based on the knowledge supplier/consumer concept for KM, knowledge can be modelled as flowing from supplier to consumer [Milton, 2005]. In this model, knowledge can flow either directly between people, or indirectly via a knowledge base. Figure 2.4 illustrates these two forms of knowledge flow.

Knowledge flows directly between people when they communicate in person, such as in discussions during meetings, by telephone or by email. Knowledge flows indirectly between people through a more elaborate process whereby the knowledge supplier codifies key elements of explicit knowledge, organises this information in a knowledge base, and then publishes the information so that it can be accessed by knowledge consumers.

University of Cape Town

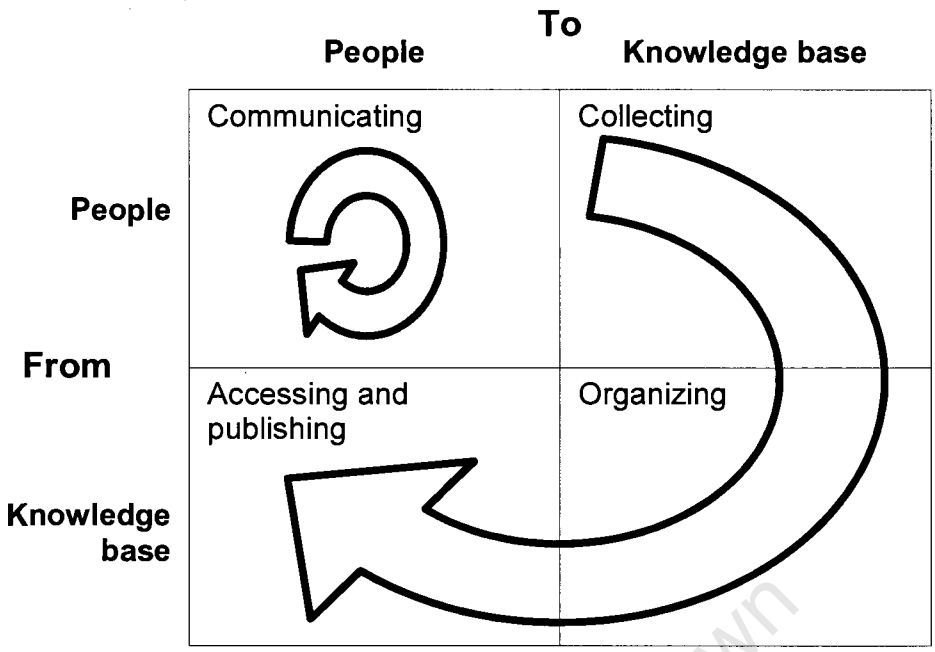


Figure 2.4: Flow of knowledge from supplier to consumer (diagram adapted from Milton [2005]). Knowledge flows either directly between people, or via a knowledge base.

2.4.5 Knowledge Forms

The classification of knowledge into multiple 'knowledge forms' is a technique commonly used by scholars in the KM field [e.g., Polanyi, 1958; Allee, 1997; Galliers & Newell, 2001; Alavi & Leidner, 2001]. KM authors often use this method of classification to focus on the nature of knowledge that emerges from KM processes [Galliers & Newell, 2001]. KM authors divide knowledge in many different ways depending on the context of study; for example a study of KM in project management may differentiate between technical and strategic knowledge forms [Galliers & Newell, 2001]. Commonly used knowledge classifications are those of 'tacit knowledge' and 'explicit knowledge' [Polanyi, 1958; Spender, 1996] (see Section 2.4.1.3).

2.5 A typology of KM

A broad range of literature related to KM has been produced, including journal articles, papers, and books. Most are concerned with the more managerial aspects of KM [Davenport & Prusak, 1998] and the rewards associated with being a knowledge-based or learning organisation [Leonard, 1999]. The effective implementation of a sound KMS and the transformation of corporations into 'knowledge-based organisations' are seen as a mandatory condition for companies to succeed in the

University of Cape Town

knowledge economy [Leonard, 1999]. However, becoming a knowledge-based organised requires appropriate planning [Edvinsson & Malone, 1997].

The literature on KM derives predominantly from the fields of business science [Alavi & Leidner, 1999], management [Davenport & Prusak, 1998; Martensson, 2000; Sveiby, 1997; Teece, 1998], marketing [Davenport & Klahr, 1998], and information technology [Guthrie & Petty, 1999; Newell *et al*, 1998]. Specialist forms of KM, such as the management of technical innovation [Leonard, 1999; Nonaka & Takeeuchi, 1995], are a growing concern to researchers in the field.

The broadness of the field and the wide range of topics covered make a review of the literature a complex undertaking. Each new piece of reviewed literature presents a view of KM from that author's particular interest and perspective. For this reason, Binney's [2001] meta-analysis of KM applications is drawn on to give a brief overview of the general KM applications described in the literature, before addressing the literature on KM within technical product development.

Binney [2001] clusters KM applications around common ideas, which include: creating new knowledge; improving processes and methods; understanding patterns in vast amount of data; tapping expertise in organisations; and developing employee capabilities and competencies. Binney [2001] then separates the KM applications into six main categories, which are referred to as transactional KM, analytical KM, management of knowledge assets, process-based KM, developmental KM, and innovation management. Alternate typologies have been provided, such as Earl [2001]; however, the intention of this section is to provide the reader with a broadened view on KM issues, which Binney [2001] provides.

Each type of KM described in Binney's [2001] meta-analysis is explained in the subsections that follow. Figure 2.5 provides a summary of the meta-analysis.

University of Cape Town

	Transactional	Analytical	Asset management	Process	Developmental	Innovation and creation
KM applications	▪ Case-based reasoning (CBR) ▪ Help desk applications ▪ Customer Service applications ▪ Order entry applications ▪ Service agent Support applications	▪ Data warehouse ▪ Data mining ▪ Business intelligence ▪ Managem/t information systems ▪ Decision support systems ▪ Customer relationship managem/t (CRM) ▪ Competitive intelligence	▪ Intellectual property ▪ Document managem/t ▪ Knowledge valuation ▪ Knowledge repositories ▪ Content managem/t	▪TQM ▪ Bench marking ▪ Best practices ▪ Quality managem/t ▪ Business process re-engineering ▪ Process improvem/t ▪ Process automation ▪ Lessons learned ▪ Methodology ▪ SEI/CMM ISO9000, Six Sigma	▪ Skills development ▪ Staff competencies ▪ Learning ▪ Teaching ▪ Training	▪ Communities ▪ Collaboration ▪ Discussion forums ▪ Networking ▪ Virtual teams ▪ Research and development ▪ Multi-disciplinary teams
Enabling technologies	▪ Expert systems ▪ Cognitive technologies ▪ Semantic networks ▪ Rule-based expert systems ▪ Probability networks ▪ Rule induction, decision trees ▪ Geospatial information systems ▪ Portals, internet, intranets, extranets	▪ Intelligent agents ▪ Web crawlers ▪ Relational & object DBMS ▪ Neural computing ▪ Push technologies ▪ Data analysis and reporting tools	▪ Document managem/t tools ▪ Search engines ▪ Knowledge maps ▪ Library systems	▪ Workflow managem/t ▪ Process modelling tools	▪ Computer based training ▪ Online training	▪ Groupware ▪ Email ▪ Chat rooms ▪ Video conference ▪ Search engines ▪ Voice mail ▪ Bulletin boards ▪ Push technologies ▪ Simulation technologies

Figure 2.5: The KM spectrum (adapted from Binney [2001]).

University of Cape Town

2.5.1 Transactional KM

Transactional KM involves the application of technologies in the course of completing a transaction or a unit or work, such as entering an order or handling a customer query or problem. Davenport and Klahr [1998] describe this type of KM as 'case-based reasoning' in a customer service application, where knowledge is 'packaged' for the user in the course of interacting with a system and can be accessed through help desks, customer service, order entry and other field support applications. In transactional KM systems, users can choose what to do with the knowledge presented; but its access methods and presentation forms are usually not optional.

2.5.2 Analytical KM

Analytical KM interprets or creates new knowledge from significant amounts of material or from highly disparate sources [Binney, 2001]. In analytical KM applications, large amounts of data are used to derive trends and patterns. This involves turning data into information that can become knowledge if a person makes use of it. Analytical KM applications tend to focus on customer-related information that assists marketing or product development [Yoon *et al.*, 1999]. Competitive intelligence applications, another example of analytical KM, are used by companies and government agencies to analyse and understand their marketplace and assess competitive activity [O'Dell *et al.*, 2003; Fuld, 1994]. The most common method used in these applications is scenario planning [Cushman *et al*, 1999] – used if one needs to provide quick answers to complex questions; an example of this would be an engineer solving a question, such as "what would I need to know about hardware interfaces in order to develop a cellular telephone that can communicate with a PC?"

2.5.3 Management of knowledge assets

The management of knowledge assets involves the management of explicit knowledge that has been codified in some way [Guthrie and Petty, 1999], an example being the management of intellectual property (IP) and the processes surrounding the identification, exploitation and protection of IP [Teece, 1998]. This type of KM is analogous to a library, with the knowledge assets being catalogued in various ways and made available for unstructured access and use. Knowledge assets are often created as a by-product of 'doing business' and are kept for future uses, these uses often being unknown at the time the assets are created, captured and stored. What differentiates asset management from analytical systems is that the assets are often more complex [O'Dell *et al.*, 2003]; they may also require some level of intervention in order to codify them. For example, the capture of project or product

University of Cape Town

development history typically requires user intervention, in addition to certain types of prior experiences and a working product.

2.5.4 Process-based KM

Process-based KM includes the codification and improvement of processes, work practices, procedures and problem-solving methods. Process-based KM is often an outgrowth of other disciplines such as total quality management (TQM) and process reengineering. The knowledge assets produced in this category are also known as 'engineered assets' in that they often involve working with specialists to document best practices in standard formats [Henniger, 1997b]. Process knowledge assets are often improved through internal (or in-house) lessons, formalised processes, publicised best practices, and benchmarking [Feltus, 1995; Hill, 1999; O'Dell & Grayson, 1999].

2.5.5 Developmental KM

Developmental KM applications focus on increasing the competencies of an organisation's knowledge workers. This is also referred to as investing in human capital [Edvinsson & Malone, 1997]. These applications cover the transfer of explicit knowledge via training sessions (e.g., training ES engineers on a new compiler tool chain), or the planned development of tacit knowledge through developmental interventions [Elliott, 1999] (e.g., arranging to capture and systematise informal knowledge during a project). In addition to traditional training in 'explicit knowledge', developmental KM relates to products, methods and technologies, with emerging emphases on developing a 'learning organisation' [Senge, 1990] and 'communities of practice' [Wenger *et al.*, 2002] to enable the exchange of ideas and experiences.

2.5.6 Innovation management

Innovation-based KM applications focus on providing an environment in which knowledge workers can collaborate in the creation of new knowledge. While there is still a necessity for a certain amount of individual innovation, innovation increasingly comes from interdisciplinarity and teamwork. This category of KM is summarised by Nonaka & Konno [1998], who claim that knowledge is manageable only insofar as leaders in an organisation are able to accept and nurture knowledge creation.

Nonaka *et al.* [2001a] uses the term *ba*, a Japanese word, to express the concept of an environment in which knowledge workers of various disciplines can come together

University of Cape Town

to create new knowledge. The role of top management is to provide *ba* for knowledge creation, and to manage 'knowledge emergence' [Nonaka & Konno 1998].

The most common application referenced in the literature concerning innovation management is the creation of new products or company capabilities [Binney, 2001].

2.6 Knowledge management systems (KMSs)

Broadly speaking, a KMS refers to a system for managing knowledge within an organisation, which supports the creation, capture, storage and dissemination of knowledge among members of the organisation or a broader, related community [Alavi & Leidner, 2001; Dalkir, 2005]. A KMS is often a component of a broader KM initiative that may in time be part of a higher-level and organisation-wide initiative involving various forms of upgrade and improvement, such as overhauling technical support infrastructures [Alavi & Leidner, 2001]. Maier & Hadrich [2006] emphasise that the term 'KMS' is used ambiguously in the literature and that the notion of a KMS generally refers to an abstraction of enabling technologies for effective KM.

According to the current literature concerning KMS development (such as Holsapple [2003], Groff and Jones [2003], and Maier and Hadrich [2006]), the implementation of a KMS generally comprises three parts. Based on Groff and Jones [2003], these parts are more in the form of phases, which may be revisited as the KM initiative progresses; the phases are:

1. The initial step is to develop an overall strategy for a KM initiative, which identifies the organisation's goals and how to achieve them.
2. Next, processes and activities that facilitate KM are implemented, as well as methods for collecting and distributing knowledge across the organisation.
3. The processes and activities decided in the second step are then enhanced through the development of tools and other artefacts (collectively referred to as the *KMS framework* in this thesis) to support these KM processes.

In software engineering contexts, a KMS is likely to have as its ultimate goal the reduction of software development costs, ideally together with improvement in software quality and reduction in the workload of software engineers [Dingsøyr & Conradi, 2002]. The management, collection and distribution of knowledge involved in such an initiative is likely to be done by the project managers and software

University of Cape Town

developers themselves (considering the complexity of this knowledge), or it could be done by a separate team (see 'knowledge brokers' in Section 2.7.9). An example of a tool to support this form of KM, where operational information or knowledge can be found by different practitioner groups of a company (e.g., developers, project managers, quality management) usually tend to be in the form of corporate intranet systems where explicit knowledge is represented and stored in databases, web-pages and other types of computer files [Dingsøyr & Conradi, 2002].

2.6.1 The two principle uses of a KMS

Two different uses of a KMS can be identified [Dingsøyr & Conradi, 2002; Hansen *et. al.*, 2001], namely:

1. *Codification:* the systematisation and storage of information that represents knowledge use by an organisation, in order to share this knowledge more efficiently among people in the organisation; and
2. *Personalisation:* supporting the flow of information in a company by storing information about knowledge sources in a company.

The codification strategy does not fit all types of knowledge [Dingsøyr & Conradi, 2002]. In situations where knowledge is extremely context-dependent, and where the context is difficult to transfer, it can be dangerous to reuse knowledge without analysing it critically first [Jørgensen & Sjøberg, 2000]. An additional strategy (apart from the two mentioned previously) is to support the growth of knowledge and the creation of new knowledge by arranging for innovation through special learning environments [Kessels, 2001] or expert networks [Davenport *et al.*, 1998].

2.6.2 Growth of a KMS

Knowledge is context dependent, which leads to KM being dependent on the ways in which specific knowledge workers in an organisation perform knowledge work. Different organisations have different types of knowledge workers who have dissimilar KM needs that change over time [Dignum, 2006]. For this reason, a KMS usually needs to be custom designed for the specific organisation concerned, with the design being sufficiently flexible to respond to changes in the organisation. Consequently, a KMS should not be considered as a once-off development project, but rather as a continuous development effort that grows an organisation's KMS, as the organisation evolves [Groff & Jones, 2003].

University of Cape Town

It is difficult to ascertain the KM needs of an organisation and then to integrate a new KMS into the organisation in a short time period. Typically, an organisation's KMS is incrementally modified, gradually moving the organisation away from the use of *ad hoc* KM methods towards more formalised strategies that are consistent between knowledge workers (as is described by Easterby-Smith & Lyles [2005] in terms of the implementation and refinement of KM practices).

The development of a formalised (or *visible*) KMS can be seen as paralleling the customary seven phases of development [Schach, 2005]. However, take note that a KMS is not a software tool, but rather an interconnection of people, organisation practices, artefacts and other aspects that may be supported by various software tools [Davies, 1998].

2.6.3 Establishment and evolution of a KMS

The establishment and maintenance of a KMS generally involves a sequence of phases, which may have to be repeated; as mentioned earlier (at the start of Section 2.6), there are usually three phases: 1) developing the strategy; 2) deciding on and implementing the KM processes; and 3) enhancing the KMS through refinement and development of specialised artefacts and tools (i.e., establishing a supporting framework) [Groff & Jones, 2003].

In this thesis, Groff and Jones' [2003] three phases for implementing a KMS are expanded into a seven-phase model describing a general approach to initial development and then incremental, long-term refinement of the KMS. This incremental, long-term refinement is referred as *KMS evolution* in this thesis (see Section 3.1.2). This generalised model is based on an integration of several published methodologies, including Nonaka's "Knowledge Spiral" [Nonaka *et al.*, 2000, pg. 9], Allee's "Organisational Knowledge Management Model" [Allee, 1997, pg. 48] and Milton's "Knoco Ltd 12-component framework" [Milton, 2005, pg. 10]. This generalised model of KMS evolution is outlined below:

1. *Phase 1: Assessment of readiness*: The organisation makes an initial assessment of its readiness regarding cultural issues, processes and technology. The purpose of this phase is to identify which perspectives on KM are critical for the particular business concerned and its context.

University of Cape Town

2. *Phase 2: Knowledge audit*: A KM initiative should have defined boundaries. The single most common reason for problems in the KM field is the failure to focus on a manageable problem. The organisation should identify which organisational, personal and technical knowledge is most important. Potential benefits should also be identified at this stage, because these will provide measures of success.

3. *Phase 3: Current initiatives*: The purpose of this phase is to map initiatives that are already underway or planned, which have a bearing on the KM initiative as defined in Phase 2. This will enable the various initiatives to be harmonised and will assist the process of gaining support from the appropriate stakeholders.

4. *Phase 4: People and guidelines*: At this stage, it is necessary to ensure that all the basics of good information management are in place before trying to construct a KM system. Incorporating the principles of good information management is fundamental to the development of a KMS. It is important to take a process perspective that crosses internal boundaries within the organisation. A plan to deal with the cultural and human change issues should be created at this stage. This phase includes identifying which knowledge workers are in the greatest need for more formalised KM methods. It is also essential for the chosen KMS to be compatible with the organisation's practices and culture.

5. *Phase 5: Design/mapping*: The purpose of this phase is to maintain the momentum of the project by mapping the knowledge requirements and building a model of the KM design. This phase also involves deciding on which knowledge workers are going to be users of the KMS, and deciding which types of knowledge work have the greatest need for a formalised KMS. The KM needs for knowledge workers involved in this work should be identified and the ways in which these individuals currently perform KM (even *ad hoc* KM) should be investigated. This involves identifying people involved in the KMS, processes by which these people manage knowledge, and the artefacts (such as software tools, files or equipment) which are used in the processes. Coordinated KM strategies would then be designed and gradually introduced into the organisation, leading to the organisation using an increasingly more formalised and visible KMS.

University of Cape Town

6. *Phase 6: Technology*: At this stage, the requirements for the KMS are clear. The organisation now needs to think about the right platforms, technologies and communications infrastructure (collectively referred to as the *KMS framework* in this thesis – see Section 2.6.6) to deliver the solution.

7. *Phase 7: Review of benefits*: The purpose of this phase is to review progress in order to ensure that the business benefits identified have been delivered and to identify actions for further improvement of the system.

An additional advantage to a KMS initiative is that the individuals involved often obtain a detailed understanding of the underlying KM strategies and fundamental techniques related to the knowledge work concerned, which may be of value even if the resultant KMS is short-lived [Dingsøyr, & Conradi, 2002]. There are a variety of disadvantages associated with a KMS initiative, of which the major drawbacks include the amount of time required to study knowledge work [Jørgensen & Sjøberg, 2000] and the cost of purchasing or implementing customised tools for the KMS [Herrmann *et al.*, 2004].

2.6.4 Structure of a generic KMS

This section briefly outlines the structure of a generic KMS. As was the case for *KMS evolution* in the previous section, the generic structure of a KMS described below is an integrated view of strategies described by influential theorists, specifically Nonaka *et al.* [2001b], Allee [1997], Van der Spek & Spijkervet [1997] and Milton [2005].

The structure of a KMS can be divided into six interrelated aspects, namely: roles, groups, desires, work, workflows and artefacts [Van der Spek & Spijkervet, 1997]. Each aspect, which provides a different view of the KMS, is outlined below:

1. *Role:* describes the behaviour, responsibilities, and needs of a certain type of person involved with the system. Each *role* desires a certain objective.

2. *Group:* a collection of people, which can be a combination of the same or different roles.

3. *Desire:* a body of knowledge that a role wants to learn and/or to share; for example, a knowledge worker may desire to learn how to use a certain piece of equipment.

University of Cape Town

4. *Work:* a type of work performed by a certain role to achieve a *desire* (such as dialogue, web searches, writing, etc.).

5. *Workflow:* describes when and in what sequence activities are performed to achieve a certain *objective*.

6. *Artefact:* physical resource (e.g., books and equipment) or digital resource (e.g., software tools, documents and other files) used in activities.

The KMS aspects described above are highly interrelated; these aspects are used to describe the design of the KMS by describing slices of the system that abstract the details of the other aspects. When defining a new KMS, an aspect to start with is that of the roles, because desires, activities and the other aspects follow naturally from the act of describing a role. This sequence thus provides only a rough guideline to the way in which a KMS can be expressed and then progressively refined to describe its structure and operation in more detail [Van Zolingen *et al.*, 2001].

2.6.5 Visibility of a KMS

The terms 'hidden KMS' and 'visible KMS' are used in this thesis to distinguish between, respectively, the initial KMS and the resulting KMS of a KM initiative (as would result from performing the phases discussed in Section 2.6.3, for instance). These terms are further elaborated upon below.

Prior to the establishment of a KMS, as might be the case before the first phase of establishing a KMS [Groff & Jones, 2003], knowledge workers within the organisation may be using a variety of inconsistent and poorly defined *ad hoc* KM methods (such as described by Alavi and Leidner [1999]). The term 'hidden KMS' or 'invisible KMS' is used in this thesis to refer to this type of situation, where the KM methods of an organisation are not purposely studied and refined[1].

In the course of developing an explicit KMS, as per following the phases discussed in Section 2.6.3, the organisation generally moves from the hidden KMS that uses *ad hoc* KM methods, towards a more 'visible KMS' where the KM methods are more explicit, and thus more visible, and the methods used consistently among the knowledge workers.

---

[1] Considering the generally loose definition of KMS given in Section 2.6, it may be that any organisation that performed KM also has some form of KMS, even though this KMS may not be documented or somehow articulated by the individuals within the organisation.

University of Cape Town

2.6.6 Framework of a KMS

The term *KMS framework* is used in relation to a *visible* KMS (i.e., a KMS that has reached the sixth 'technology' phase of *evolution* described in Section 2.6.3). The framework of a KMS refers to the supporting elements (e.g., software tools and other support technologies) of the KMS and the documentation (e.g., role and process descriptions) that makes it *visible*.

2.7 Roles of people involved with KM

The main ingredients of successful KM in an organisation is the willingness of people to share knowledge [Snowden, 1998] and effective leadership to establish and direct the KM practices [Pan & Scarbrough, 1999].

Common roles that exist in a KMS include the chief knowledge officer (CKO), KMS user, change agent, knowledge engineer, knowledge steward, knowledge analyst, knowledge worker, and knowledge broker. This section overviews these, and other, commonly occurring KM roles used by organisations involved with technical product development. These roles are described according to their use by prominent authorities in the field, including Davenport and Prusak [1998], Drucker *et al.* [1998], Groff and Jones [2003], Holsapple [2003], McDermott [1999a], Milton [2005], Nonaka & Takeeuchi [1995], and Sveiby [2001].

As noted before, a KMS is needs to be adapted for the specific type of knowledge work and organisation concerned; consequential, a particular KMS may include only a selection of the roles described here, or it may include a variety of additional roles. However, the roles of CKO, knowledge worker, and knowledge engineer are likely to exist in virtually any KMS, given the definition of these roles (see Section 2.7).

University of Cape Town

2.7.1 Knowledge suppliers and knowledge consumers

Milton provides an abstracted view of a KMS by viewing an organisation as consisting of two types of people: knowledge suppliers and knowledge consumers[2] [Milton, 2005]. Many of the roles described later can be viewed from the perspective of either supplying or consuming knowledge.

---

[2] Milton [2005] uses the term "knowledge user" instead of "knowledge consumer"; but the term "knowledge consumer" is used here to avoid confusion with the term "KMS user".

Milton's [2005] concept is principally based on knowledge consumers drawing expertise from knowledge suppliers. From this perspective, knowledge suppliers create knowledge through experience (performing actions and interpreting the results) and from reflecting on prior experiences to derive guidelines, theories, rules and heuristics. Knowledge consumers, in contrast, consult knowledge suppliers to assist in creating new knowledge, or to leverage other people's knowledge in order to accomplish tasks more quickly and with fewer mistakes. An individual switches from the role of knowledge consumer to that of knowledge supplier, as he or she changes from personally performing knowledge work to assisting others in accomplishing such work [Milton, 2005].

2.7.2 Chief Knowledge Officer (CKO)

The CKO is in charge of the KMS, and is therefore responsible for the establishment and continued operation of the KMS [Bonner, 2000]. The main tasks of the CKO are to coordinate users of the KMS and to ensure that each aspect of the KMS is functioning smoothly. The responsibilities of the CKO include finding executives to join the steering committee, overseeing the acquisition of knowledge engineers to build the KMS, and ensuring that the various roles of the KMS are allocated [Holsapple, 2003].

Before an organisation delves into the time-consuming task of determining detailed requirements for a new KMS, the organisation first needs to decide whom to designate as the CKO, viz. the person that will lead the KM initiative. The CKO is usually a senior corporate executive who has detailed experience in many of the tasks performed by the users of the planned KMS [Rumizen, 2002]. The CKO needs to be available to spend large amounts of time on KM tasks and on administering the KMS; he or she should also have up-to-date personal experience of tasks performed by users of the KMS [Davenport *et al.*, 1998].

2.7.3 KMS user

A KMS user either draws knowledge from an organisation using the KMS (i.e., acting as a knowledge consumer), or is actively involved in adding to that organisation's knowledge through participation in the KMS (i.e, acting as a knowledge producer) [Milton, 2005]. Developers of the KMS (i.e., the knowledge engineers) are also considered as users of a KMS because, by building the system itself, they are contributing to the knowledge of the organisation.

University of Cape Town

The most common user of a KMS should be the knowledge worker (Section 2.7.4). If the knowledge worker is not the most frequent user of the KMS, then the cost of maintaining the KMS is unlikely to be recovered from its use [Simard *et al.*, 2007]. A simple statistic that a knowledge analyst can use to determine the feasibility of a KMS is to compare how often knowledge workers use the system with how often knowledge engineers add to the system [O'Brien, 2006]. If knowledge engineers add to the system more than knowledge workers use the system, then there is clearly a fault in the workings of the KMS [O'Brien, 2006].

2.7.4 Knowledge worker

Knowledge workers are educated persons who use their education and experience to achieve their objectives [Drucker, 2000; Groff & Jones, 2003]. Knowledge workers predominately use their minds more than their hands to accomplish work [Drucker *et al.*, 1998]. The term 'knowledge work' is used to generalise the type of work done by a knowledge worker. ES engineers can be considered a type of knowledge worker, considering the complicated products they work on, the range of concurrent and interrelated technical activities typically performed in these projects [Kettunen, 2003], and their general dependence on education, experience, and mental problem-solving and innovation abilities. Concrete examples of KM activities performed by ES engineers include researching operations of microprocessors and peripherals to write device drivers, and studying based software (produced by a prior in-house project or externally) for reuse in a new project [Kettunen, 2003].

2.7.5 Change agent

A change agent is a person or a team (e.g., a consultancy firm) that is responsible for planning and implementing changes in an organisation [Vinter, 2005]. The CKO is often the principal change agent in a KMS, being responsible for planning how the organisation needs to change to adopt the KMS [Abell & Oxbrow, 1999]. But this undertaking can be difficult, time-consuming, and fraught with political and cultural problems. For this reason, the responsibility of changing the organisation's culture to accept a new KMS is sometimes delegated to a change agent [Groff & Jones, 2003].

2.7.6 Knowledge engineer

The main task of the knowledge engineer is to make explicit knowledge usable [Borghoff & Pareschi, 1998]. Knowledge engineers essentially design and implement the support infrastructure for a KMS, updating or pruning the system while it is in use.

University of Cape Town

They also add resources and documentation to the knowledge base. Important characteristics of a knowledge engineer are to maintain consistency in the resources developed, and to ensure that these resources are easy to use [Grant, 1996].

Activities performed by a knowledge engineer (adapted from Grant [1996] and Cortada & Woods [1999]) include:

1. *Documentation*, such as writing down procedures explaining how to use features of the KMS.
2. *Classifying and associating* documents, programs and other resources.
3. *Programming*, such as developing executable macros and programs to automate routine tasks performed by knowledge workers. This approach can be used to capture elaborate and frequently performed procedures in a highly repeatable manner, without the user of these programs having to read through detailed information and perform manual operations.
4. *Transcribing* rough notes and recordings of procedures carried out by knowledge workers for inclusion into the knowledge base.

The types of resources developed by a knowledge engineer depend both on his or her capabilities, and on the needs of the knowledge users addressed. The knowledge engineer may also work with information technologists who maintain file servers, databases and other IT systems. A knowledge engineer thus generally benefits from an understanding of the computer systems and IT resources available to the organisation [Cortada & Woods, 1999].

The knowledge engineer may have a second role as a knowledge worker within the organisation. Depending on the complexity and scale of the KMS concerned, individuals may be responsible for several roles [Maier, 2004].

2.7.7 Knowledge steward

The knowledge steward has two chief responsibilities: capturing and codifying tacit knowledge, and facilitating the use of the KMS [Holsapple, 2003]. Knowledge stewards make tacit knowledge explicit by conducting interviews with knowledge workers, or observing (and recording) knowledge workers in action [Tsui, 2002]. Knowledge stewards may transform their recordings into more meaningful documents before handing them over to the knowledge engineer for adoption into the

University of Cape Town

KMS; or the knowledge steward may work closely with the knowledge engineer towards this purpose [Holsapple, 2003].

Knowledge stewards are not always themselves knowledge engineers because each role requires different skills; for example, a knowledge steward needs expertise in conducting interviews and unobtrusively observing knowledge workers in action, whereas a knowledge engineer requires skills for developing resources for the KMS. The knowledge steward could also be seen as an assistant to the knowledge engineer, assisting the knowledge engineer in time-consuming activities, such as conducting interviews, observing knowledge workers, and helping users on a one-on-one basis [Bergeron, 2003].

2.7.8 Knowledge analyst

The role of the knowledge analyst is similar to that of a system analyst, in that the knowledge analyst chiefly studies the KMS, looking at its broader effects, such as correlating changes in the KMS to changes in the company's profits [Capshaw, 1999]. The knowledge analyst generally uses statistics to discover means of making the KMS more efficient, often disseminating findings in person to relevant parties [Bergeron, 2003].

2.7.9 Knowledge broker

The duty of a knowledge broker is to help knowledge workers connect to knowledge that is not their own [Piaseki, 2005]. This generally means helping knowledge workers to find other people from whom they can learn. Knowledge brokers usually have large and diverse social networks, and know the strengths of the specific individuals in their social networks [Maier & Remus, 2003].

Knowledge brokers are sometimes involved in the act of communication between the knowledge workers they connect. For instance, the knowledge broker may be called on to act as a translator or mediator to facilitate communication between people who speak different languages. This can include situations in which the communicating parties are using languages foreign to one another, or where they have different dialects and forms of jargon in the same tongue [Cortada & Woods, 1999].

University of Cape Town

2.8 KM in technical product development

The research literature related to the use of KM in the technical and software product development context is drawn on in this section of the literature review. The findings discussed in this section have been separated into three parts:

1. Managing development teams;
2. KM tools for managing individual and team knowledge;
3. Dealing with information overload.

The literature on KM in contexts of technical product development and innovation includes several studies concerning the factors that affect product development team performance, as presented by authors such as Aurum *et al.* [2003], Dingsøyr & Conradi [2002], Kettunen [2001; 2003], Lynn *et al.* [2000], Lindvall *et al.* [2001], Lindvall & Rus [2002], Rus *et al.* [2002], Törngren *et al.* [2007] and Ulrich & Eppinger [1995].

Literature focused specifically on the implementation phase of embedded software development projects was limited. Although there is potentially significant value that can come from applying KM techniques in embedded software development, literature searches on this issue show a general lack of consistent investigations and results in this field, a result that also seen by Rus *et al.* [2001]. The most relevant and constant publications found on this topic include Aurum *et al.* [2003], Dingsøyr and Conradi [2002], Lindvall and Rus [2002], and Kettunen [2001; 2003].

The research-based KM literature within technical product development contexts mostly addresses the issues of theoretical foundations, practical techniques, software tools, and applications and practical experiences of engineers in technical contexts such as software engineering. The application of KM in technical contexts has tended to focus on the management of systems, such as transport systems [e.g., Herrmann *et al.*, 2004], ICT systems [e.g., Carlsen *et al.*, 1999; Fischer & Schneider, 1984], requirements engineering [e.g., Kirikova & Grundspenkis, 2000; Sommerville, 2005], and the development of KM software for various 'soft skills' applications (e.g., user-feedback on technical products [Hoffmann *et al*, 1999]). Much of the literature is also more focused on issues such as management of knowledge to assist with other forms of management [e.g., Langer *et al.*, 2006; Gao *et al.*, 2005; De Meyer *et al.*, 2002], methods of improving organisational or team communication [e.g., Counsell *et*

University of Cape Town

al., 2005; Dionne *et al.*, 2004], performance analysis of engineering teams [e.g., Kettunen, 2003; Wandeler *et al.*, 2006], and strategies and tools to manage technical product development (but with less emphasis on management of the knowledge produced by these developers) [e.g., Cordeiro *et al.*, 2007; Gopalswamy *et al.*, 2004; Jepsen *et al.*, 2007; Kommeren & Parvianien, 2007; Eppinger *et al.*, 1997].

2.8.1 Managing development teams and their knowledge

A product development team is often forced to manage with incomplete information and lacking technical knowledge during parts of a project. Additionally, frequent changes to a product and its related information are more inherent in the nature of development, rather than an exception [De Meyer *et al*, 2002]. Project teams do not work in isolation; and studies of KM in larger organisational contexts emphasise the need to understand the interactions between individual, team, project, company and commercial environment. This necessitates considerations, not only with regard to individual product development projects, but also in relation to multiple projects performed in an organisation and their interplay over time. A complex interaction exists between project teams, the company that performs them, the market at which resultant products are aimed, and other macro environment issues [Ulrich & Eppinger, 1995]. Multi-site considerations are likely to cause additional complexities in large and globally dispersed organisations [Desouza & Evaristo, 2003].

A number of common solutions are applied to the management of knowledge-based teams. Based on a broad view of the literature, these strategies have been grouped into six main categories, namely: 1) steering committees [Holsapple, 2003], 2) communities of practice [Wenger, 1998], 3) team learning [Bűchel & Raub, 2002], 4) team knowledge sharing [Kettunen, 2003; Louridas, 2006], 5) distributed teams [Langer *et al.*, 2006], and 6) sub-contracting [Kommeren & Parviainen, 2007]. A summary of each category follows.

2.8.1.1 KM steering committee

A governing body is generally required to establish and direct a KMS in a large organisation. This governing body is commonly referred to as the KM steering committee and is chaired by the CKO [Holsapple, 2003].

2.8.1.2 Communities of practice

A Community of Practice (COP) is often a component of a KMS [Wenger *et al.*, 2002]. A COP can essentially be viewed as a voluntary group of people who interact

University of Cape Town

regularly to learn from one another. Members of a COP benefit from gaining deeper insights and understanding of problems by being able to exchange ideas and help one another, which can lead to valuable benefits and new innovations [Brown & Duguid, 1991]. The COP coordinator facilitates the operation of a COP, such as scheduling times and venues for meetings, and informing people in the organisation of the existence of the COP and of when and where the meetings are held.

The COP coordinator or 'champion' is typically a knowledge worker and a member of the COP [Smith & McKeen, 2003]. The role of COP coordinator can be a rotating one where the member acting as COP coordinator hands over coordination tasks to another member after some time. The COP coordinator does more than scheduling meetings and encouraging attendance, however; the most important skills of the COP coordinator are to assist the COP to develop as a community (rather than as a regulated work group), and to chair meetings so that all members can participate fairly [McDermott, 1999a].

2.8.1.3 Team learning

Team learning has been recognised as an important success factor for product development and innovation [Lynn *et al.*, 2000]. The project team should negotiate a shared vision and common objectives to improve success of this strategy [Lynn *et al*, 2000]. Various forms of knowledge networks usually occur in team learning within a context of R&D or innovative product development [Bűchel & Raub, 2002].

2.8.1.4 Team knowledge sharing

A product development team needs to master many forms of technical knowledge to be able to develop a product successfully; but not every member of the team has to know everything [Kettunen, 2003]. Inter-team knowledge transfer is an important enabler of accelerated product development. Inter-project learning also facilitates larger-scale productivity improvements within technical development contexts [Kettunen, 2003].

Tacit knowledge sharing takes place in interactions between people, but enabling conditions must be satisfied in order for such knowledge-sharing interactions to take place [Nonaka & Takeuchi, 1995]. For example, the emerging Agile software development methods encourage intensive communication with less formal documentation [Abrahamson et al., 2003; Lindvall *et al.*, 2004]. Some types of knowledge are not easy to disseminate on paper alone and need additional face-to-

University of Cape Town

face communication. When product and process knowledge is to be shared, team proximity is often an issue [Kettunen, 2003].

2.8.1.5 Distributed teams

Distributed product development projects are becoming an increasingly common phenomenon, which is showing potential benefits such as improvements in time-to-market efficiency and access to greater and less costly human resources [Langer *et al*, 2006]. In a paper describing the experience of over 10 years of distributed development at Philips, derived from about 200 projects, Kommeren and Parviainen [2007] identify a number of lessons learned from multi-site development. In particular, they point out that explicit agreements and ways of working should be defined – with the following areas needing the most attention: 1) team coordination and communication, 2) requirements capture and architecture design, 3) integration, and 4) configuration management.

2.8.1.6 Sub-contracting

The main lesson learned from subcontracting technical product development is the need for explicit attention and ways of working with respect to selection of suppliers, specification of the work to be subcontracted, and establishment and content of the contract [Kommeren & Parviainen, 2007].

2.8.2 KM tools for managing individual and team knowledge

The research literature evaluates several strategies, applications and tools for managing individual and team knowledge. These are summarised below.

2.8.2.1 Training workshops

The CKO is responsible for arranging training workshops to bring people into the KMS and to train these new users of the system. In the case of a large KMS, separate training workshops may be arranged for each user group (usually a department), and these may be presented by the knowledge engineer allocated to that user group [Lynn *et al.*, 2000]. The CKO may delegate training workshops to experienced knowledge stewards, but usually knowledge stewards are involved with training on a one-on-one basis, which is often combined with observing knowledge work [Bűchel & Raub, 2002].

2.8.2.2 Yellow Pages

An organisation may produce a 'yellow pages' catalogue [Dingsør, 2003] which lists people in the organisation and their areas of expertise. This catalogue contains email

University of Cape Town

addresses, wikis, blogs, homepages and the like. A 'yellow pages' is used by knowledge workers as a means to find colleagues who can provide the expertise needed to accomplish certain forms of knowledge work [Cortada & Woods, 1999].

2.8.2.3 Performance analysis

Performance analysis is usually used as a project management tool, and involves measuring the performance of a team's work [e.g., Kettinger *et al.*, 1997], rather than the team's knowledge. However, several simulation-based tools have been developed for the analysis of knowledge production and transfer, such as the one used in the development of embedded real-time systems [Wandeler *et al.*, 2006].

Simulation-based KM tools are able to analyse knowledge-based performance early in the life cycle of product development. Wandeler *et al.* [2006] developed a simulation approach based on real-time calculus. They believe this approach to be an efficient way of evaluating knowledge performance due to its high level of abstraction, which also makes the technique suitable for early design exploration.

2.8.2.4 Responsibility charts

Responsibility charts are popular tools for project planning in general [Turner, 2009]. For example, Andersen [1996] has proposed project responsibility charts for the systematic identification of project milestone responsibilities. Anderson's ideas can be adapted for KM in ES product development by defining the producers and consumers of the key information and mapping them together as a chart.

Figure 2.6 illustrates a project knowledge sharing chart. This tabular method makes the knowledge artefact dependencies of each team member clear (in a large project, there would be many connections linking to individuals outside the project team) [Kettunen, 2003]. Both tangible and intangible knowledge items are included, because not all useful information is in a tangible form. For example, previous project experiences may be useful but largely intangible knowledge.

University of Cape Town

	Roles				
	Software project internal			Software project external	
Knowledge artefacts	Project manager	Designer	System specifier	Hardware manager	Quality manager
Previous projects history	User	n/a	n/a	n/a	Provider
Software specification A	Author	Reader	n/a	n/a	n/a
System specification B	Reader	n/a	Responsible	Contributor	n/a
Hardware data sheet	n/a	Reader	Reviewer	Responsible	n/a
ASIC hardware behaviour	n/a	User	n/a	Provider	n/a
Standard operating procedure	Reader	Reader	n/a	n/a	Responsible
User's Guide	n/a	Author	n/a	Reviewer	Reviewer
Test process experience	Provider	User	n/a	n/a	n/a

Figure 2.6: ES project knowledge sharing chart (adapted from Kettunen [2003]).

University of Cape Town

2.8.2.5 Status tracking

Charts similar to responsibility charts (see Figure 2.6) for managing team knowledge have been developed to describe when particular types of information are needed in a project (Romano *et al.* [2002], for instance, discuss techniques for collaborative project work).

2.8.2.6 AI tools

Artificial intelligence approaches have proposed agent-based frameworks for modelling organisational and personal knowledge from two conceptual models: the first describes the intelligent enterprise memory, the second models an intelligent organisation's KMS [Grundspenkis, 2007; Ellis & Wainer, 2002; Knapik & Johnson,1998].

2.8.2.7 Shared buffers

Team-based knowledge sharing can be enhanced thought the use of 'shared buffers' [Gao *et al.*, 2005].

2.8.3 Managing information in technical development projects

KM strategies and technologies to assist or improve product development teams to find appropriate data for their projects can be divided into knowledge acquisition [Birk *et al.*, 1999] and knowledge reuse [Tautz & Althoff, 1997] approaches, and various combinations thereof [Birk & Tautz, 1998]. Communication gaps and missing information have been recognised as typical causes of project failure in large-scale project work [Kettunen, 2001]. What is needed then is a KM tool to ensure that critical information is both produced by the right persons at the right time, and utilised by all the relevant parties. Broad issues that need to be addressed in the design and implementation of these tools relate to the management of information and strategies for finding information during product development. The next two subsections (2.8.3.1 and 2.8.3.2) explore these issues more closely.

2.8.3.1 Issues in information management

Based on the literature, there are three broad categories of information management related to technical product development, namely:

1. Managing product requirements and specifications;
2. Management of documentation; and
3. Improving information storage and search techniques.

Each category above will be explored further.

Requirements and specifications

A seminal investigation was conducted by Curtis *et al.* [1988]; it showed that a lack and insufficient spread of domain knowledge and requirements-related gaps are major difficulties in the development of technical products. Kettunen [2001] similarly found that problems relating to requirements and specifications were among the main causes of trouble for telecommunications equipment development. Incomplete software requirements and specifications of the system are generally troublesome for embedded software projects [Kettunen, 2003]. In the case of ES development, it is important that the software developers have sufficient knowledge of the ES to be produced and its hardware behaviour in order to work efficiently on the software part of product design [Kettunen, 2003].

University of Cape Town

Documentation

In technical product development environments, there is often a trade-off between the completeness of documentation and the effort required to develop and maintain these documents [Schach, 2005]. The key is to find a practical balance so that the risk caused by partial or incomplete information is justified by the resource expenditure; earlier literature on risk-driven specification acknowledges this strategy [e.g., Boehm, 1988]. The more recent Agile software development approach favours methods that avoid producing possibly intermediate or valueless documentation that is likely to be made obsolete or redundant [Kettunen, 2003].

Improving data storage and search techniques

Being familiar with search engines and search functions is clearly important when working in knowledge-rich contexts [Kitamura *et al.*, 2006]. Osiov *et al.* [2006] state that 'linguistic knowledge' is required for search relevance improvement. A good understanding of these issues and of the effective use of keywords and directory structure layouts improves both search strategies and organisation of data [Capra et al., 2007].

2.8.3.2 Tools for information management

Development projects need methods by which team members can find relevant and usable data efficiently, as well as strategies to capture valuable data without excessive manual intervention [Kettunen, 2003]. Common methods to achieve these objectives include: ontologies for creating and maintaining data records [e.g., Kitamura *et al.*, 2005], planning templates [e.g., Kettunen & Laanti, 2005], computer-aided software engineering [e.g., Wood & Agogino, 1996], web-based AI tools [e.g., Grundspenkis & Kirikova, 2005], and product data management tools [e.g., Eppinger *et al.*, 1994]. These approaches are discussed below.

Ontologies for the creation and maintenance of data records

Engineers often have difficulty in reusing technical documents because these documents tend to be written in an *ad hoc* manner, often using a technical and possibly non-standard vocabulary developed through the course of various projects performed by the engineers concerned; these documents may also be context- or project-specific [Kitamura *et al.*, 2005]. Important aspects to consider in improving the usability of technical documents include consistency in terms, names and acronyms used in the documents and a consistent strategy for locating and identifying files and directories that contain documentation [Patil *et al.*, 2005]. In the information sciences

University of Cape Town

and computer science fields, the term *ontology* refers to 'a specification of a representational vocabulary for a shared domain of discourse' [Gruber, 1993]; such an 'ontology' can assist developers in the reuse of and sharing of technical knowledge [Stojanovic *et al.,* 2002]. In an investigation of ways to resolve these difficulties, Kitamura *et al.* [2005] developed an ontological framework for exchanging product development knowledge, for which the ontology can be used to systemise information records to improve their reuse.

Computer-aided software engineering (CASE)

In embedded software development projects, the software developers must understand not only the general operation of the target hardware but also the overall functionality of the combined hardware/software system [Ball, 2002]. Hardware/software co-development (including co-specification and co-design) attempts to build this kind of shared system-level knowledge [Chiodo *et al.*, 1994]. There have been various approaches to create CASE tool environments for such developments, where these tools can be used to offload some of the technical know-how and manual tasks, for which the software engineer would otherwise be responsible [Heikkinen, 1997].

Web-based AI tools

Rodgers *et al.* [1999] describe a design support system known as WebCADET that uses distributed Web-based AI tools. The system can provide support for designers when searching for design knowledge. WebCADET uses the 'AI as text' approach, where a knowledge base system can be seen as a medium to facilitate the communication of design knowledge between designers.

Planning templates

Kettunen [2003] describes a planning template that assists in the identification and provision of the necessary knowledge for the product development team (Figure 2.7 shows an example of this).

University of Cape Town

Planning template
Customer–supplier process cat (CUS): Acquisition Supply *Who are our customers (external and internal)?* Requirements elicitation *What do the customers really want from us?* *Who is responsible for the elicitation of the customer requirements?* Operation Engineering process cat (ENG): System requirements analysis and design *Where do I get my system requirements?* *How do I know the software architecture (and system design)?* Software requirements analysis *Which items (documents) comprise my software requirements package?* *How are the requirements managed (changes)?* Software design *What design methods and tools do I use?* *How do I change the component/subsystem external interfaces?* *Where can I find the hardware data sheets (if any)?* Software construction *What compilers etc. tools do I use?* *What implementation rules do I have to obey (e.g. coding standards)?* Software integration *What kind of integration and testing should I do?* Software testing

Figure 2.7: Embedded software project KM planning template (adapted from Kettunen [2003]).

The planning template shown in Figure 2.7 lists software product development process areas based on the ISO/IEC 15504 Reference Model. The accompanying questions (in italics) are intended to help managers and designers identify the practical information needs in particular areas of development. Each member of the project team would be required to fill in the template from his or her point of view. Each response helps the manager to consider the actual needs of that person. Overall, this KM method based on planning templates was found to be useful mainly

University of Cape Town

in the early phases of a project [Kettunen, 2003], in which it may also be used for identifying staffing and training needs.

Product data management tools

Product data management tools offer an integrated approach to combine all the information of complex products consisting of various subsystems and components into a consistently managed and accessible system [Zha & Du, 2006; Zha & Du, 2005; Feldmann, 1999; Lindeman & Moore, 1994].

Luqi *et al.* [2004] developed a documentation-driven development (DDD) approach for the management of data in complex real-time systems. This approach can enhance the integration of computer aided software development activities, which encompass the entire life cycle of a project. DDD provides a mechanism to monitor and quickly respond to changes in requirements, and it provides a friendly communication and collaboration environment to enable different stakeholders to be easily involved in development processes, thereby significantly improving the agility of software development for complex real-time systems. DDD is planned to support automated software generation based on a computational model and some relevant techniques. DDD includes two main parts: a documentation management system and a process measurement system.

2.8.4 Managing knowledge of technical development processes

This subsection begins by reviewing three general types of development process knowledge described in the research literature. Issues related to process management are then discussed, which leads into a review of tools for managing this knowledge in the context of technical development projects (issues or examples concerning ES development are included where applicable).

2.8.4.1 Development process knowledge: the input, in-situ and output knowledge types

The literature identifies three main types of engineering design process knowledge that need to be managed in technical product development contexts (based on findings from Pena-Mora *et al.* [1993; 1995], Zha *et al.* [2002] and Zha and Du [2006]). These types of knowledge are outlined below:

1. *Input knowledge:* the knowledge that developers bring with them, which relates to the specific engineering or design processes to be performed;

University of Cape Town

2. *In-situ knowledge:* the knowledge related to engineering processes that developers acquire during the development project; and

3. *Output knowledge:* the knowledge that results from a project.

2.8.4.2 Input, in-situ and output knowledge in embedded software development projects

Input knowledge relates to existing knowledge and its representation, including design knowledge in handbooks, datasheets and other (often downloadable) documentation, together with the design 'know-how' (see Section 2.4.1.3) that is likely to exists in a ES development organisation. Output knowledge comprises not only concepts and expertise related to the product being developed, but also competencies in project-related and process-related issues, including specialised knowledge about the organisation [Zha *et al.,* 2002].

The importance of input knowledge has been recognised [Pena-Mora et al., 1993; 1995] but there have been fewer research endeavours focused on in-situ or post-project output knowledge in development contexts [Zha *et al.*, 2002; Zha & Du, 2006]. Zha *et al.* [2002] suggest that in-situ or 'on-the-job' product design knowledge can also be categorised according to off-line and on-line knowledge, where the former refers to knowledge acquired in mid-stream (and usually involves leaving the job at hand), and the latter refers to new design knowledge created while working on a design task.

The production of output knowledge involves complex sets of information, as is the case for input knowledge [Zha *et al.*, 2002]. In practice, a portion of design knowledge remains outside the formal project documentation (e.g., product manuals and reference texts), and this would include informal documentation (e.g., private notes in a logbook) as well as that which is 'undocumented' [White, 2005]. In ES development this loss of knowledge tends to happen, for example, when an ES engineer's understanding of a specific hardware platform has been learned during the development process and later forgotten, needing to be relearned, during maintenance [Molnar & Nandhakumar, 2007].

During a project, new processes and tools may be developed, from which the project team can gain new personal experiences and skills [Simard *et al.*, 2007]. In the case of ES projects, developers typically formulate a particular set of techniques (e.g.,

University of Cape Town

implementation methods) to develop systems using a specific selection of hardware and tools [Graaf *et al.*, 2003]. These implementation methods involve sequences of actions or patterns that relate to the way in which development tools are used to create and modify specific parts of a product [Gamma *et al.*, 1997]. This knowledge of development techniques is valuable to the organisation, especially when it comes to using or adjusting processes and tools during operations [Simard *et al.*, 2007].

2.8.4.3 Approaches to software engineering processes improvement

Many strategies for improving software engineering processes have been developed, and these can aid in the retention and representation of software engineering knowledge. Examples of these methodologies include (based on Fuggetta [2000], Schach [2005] and Sommerville [2006]):

1. Structured techniques, including using structured analyses of past experiences to make informed design decisions;
2. Fourth generation programming languages (4GL) [Martland *et al.*, 1986];
3. Computer-aided software engineering tools [e.g., IBM, 2009];
4. Formal methods, including formal specification and verification of software [e.g., Edwards *et al.*, 1997];
5. Cleanroom methodologies, particularly methods to reduce software defects [e.g., Prowell *et al.*, 1999];
6. Process models that provide descriptions of software engineering techniques and problem-solving strategies [e.g., Brinkkemper, 1996]; and
7. Object-oriented technology to identify objects in the problem to be solved, and to use those in generating software solutions [Jacobson *et al.*, 1999].

University of Cape Town

The methodologies listed above are closely related to processes of creating software. The next section focuses on techniques for managing knowledge related to software development, or using a formal KMS for managing software-related knowledge.

2.8.4.4 Issues in software processes KM

In a meta-analysis of the KM research literature in software engineering, Dingsøyr & Conradi [2002] found several descriptions of a KMS; however, most studies did not deal with how KMS worked in the process of implementation in the organisations where they are deployed. Based on Dingsøyr and Conradi [2002] results, KM technologies show potentially beneficial results. However, there are relatively few

scientific articles that evaluate how these different methods actually work [Glass, 1999]; thus further research is needed.

Based on findings from the current literature [e.g., Rus & Lindvall, 2002; Balasubramanian *et al.,* 2005; Zhou *et al.*, 2007], the management of ES development processes tends to involve one or more of the following issues:

1. Managing complexity;
2. Selecting effective process models;
3. Using platform-based approaches;
4. Selecting an appropriate interfacing strategy;
5. Evolving the product;
6. Maintaining the product;
7. Planning for uncertainty or risk;
8. Concurrently developing and co-designing the product; and
9. Identifying approaches for achieving product variety.

The issues listed above are elaborated upon in the sections that follow, in which references are made to relevant research and other publications.

Management of complexity

The recent literature on KM in technical product development, including ES development, has focussed on approaches of dealing with the complexity in these contexts [e.g., Zhou *et al.*, 2007]. In a software engineering context, KM tends to focus on learning, such as capturing and reusing experience [Rus & Lindvall, 2002]. Common KM problems in software development are misunderstandings and imperfect communication caused by out-of-date documents, incomplete terminology definitions, undocumented information and unclear instructions [Skuce, 1995].

Reducing the complexity of ES product development through improved engineering methods (such as techniques mentioned in Section 2.8.4.3) is a recurring theme in the recent literature. Complexity reduction usually involves standardisation [Object Management Group, 2002], modelling [Karsai *et al.*, 2003], scenarios and process reference models [Larsson *et al.*, 2007]

Balasubramanian *et al.* [2005] find that the problem of learning overload commonly occurs in ES development teams, which is often caused by using many different

University of Cape Town

platforms and architecture-dependent tools. Baksi *et al.* [2001], working on embedded software for the MILAN project, discuss model-based integration of reusable components as a potential means to avoid similar types of learning overload. Likewise, Greenfield *et al.* [2007] recommend a methodology based on patterns, models, frameworks, and tools to reduce learning times. Gopalswamy *et al.* [2004] emphasise the need for work product variation management tools to handle the increasingly enlarging scope and use of model-based control systems.

Selection of a process model

An important KM decision has to do with the selection of a software process model [Kettunen & Laanti, 2005]. An appropriate process model can help developers to cope with challenges and complexity of a project; whereas an unsuitable choice of process model can add to the difficulties of development. A process model used for one project is not necessarily appropriate for another project [Kroll & Kruchten, 2003]. Some project-related problems can be traced to the process models used [Kettunen & Laati, 2005]. Kettunen & Laati [2005], for instance, recommend that a development organisation does a comparison of known software process models, including Agile methods, to build a process model selection frame that can be used as a systematic guide for choosing the process model for a project.

Platform-based approaches

A platform-based approach for technical product development allows companies to eliminate redundancies, efficiently utilise its resources and provide products for a wider market [Seth, 2007]. This approach centres on developing and sharing key components and technologies among products. By creating a common software platform, this concept can be applied to ES software development in which software modules and applications can be shared across products within a product family [Seth, 2007]. Existing products could be made into platforms to serve as a foundation on which new products are developed – but there are limitations to this approach, such as microcontroller architecture incompatibilities [Mäkäräinen, 2000].

Interfacing strategy

Embedded systems concern a variety of interfaces: hardware/hardware interfaces (or physical connections), hardware/software interfaces (such as device drivers), and software/software interfaces (such as the methods with which an application program communicates with the operating system) [Berger, 2002]. Software developers are, to some extent, limited to hardware interfacing choices made in the product design

University of Cape Town

phase [Kettunen, 2003]. Hardware/software partitioning and platform-based design issues could be incorporated into a KMS following an incremental approach, as researched by Mäkäräinen [2000] in the case of incorporating embedded software change management support into development practices.

Product evolution

A technical product is seldom created entirely from scratch; it usually reuses a variety of artefacts and components – some of which may have been developed in-house, while others have been sourced externally. In such a situation, the software team is likely to have access to an initial (possibly disorganised) base software version at the start of a new project [Kettunen, 2003]. In such a situation, the software team may still need to learn how to incorporate or modify the existing artefacts for use in a new project. In so doing, the team needs to obtain enough detail about the artefacts and existing design decisions before they can start modifying and extending them. Forsberg *et al.* [2000] describe techniques to facilitate such tasks, for example, by building a terminology base.

Product maintenance

Process knowledge includes an understanding of software 'fragility', which relates to the tendency of a particular piece of software to be unstable, unreliable or entirely non-functional at some required level of operation. Fragility is manifested in various ways, such as unreliability during times of high demand, lack of security, performance lapses, computation failures and upgrading difficulties [Joy & Kennedy, 1999]. Embedded software is often developed from, or dependent on, legacy code, sometimes with minimal or no modifications being allowed to parts of the code due to compatibility requirements, short development lead-time, and budget constraints among other factors [Ko *et al.*, 2007b]. If legacy code is repeatedly revised without considering issues of maintenance, the code gradually becomes harder to maintain, eventually needing to be entirely reengineered to make it maintainable [Ko *et al.*, 2007b]. Existing reengineering research on ES tends to focus more on hardware issues than embedded software issues [Ko *et al.*, 2007b]. Improvement to the maintainability of embedded software for 'corrective maintenance' includes other reengineering approaches, such as: process reengineering, reengineering views, and reengineering infrastructure [Aman *et al.*, 2006]. Freeman and Schach [2004] point out that the maintainability of object-oriented software products is partly dependent on the maintainability of the inheritance hierarchy concerned.

University of Cape Town

Planning for uncertainty

Technical product development is characterised by uncertainty and frequent changes, which often occurs in complex ES development projects [Mellis, 1998]. In turbulent environments, flexibility of the product development process (i.e., its ability to accommodate frequent and last-minute changes) is a key factor for success [Mellis, 1998; Mikkonen & Pruuden, 2001]. Cordeiro *et al.* [2007] describe an Agile development methodology that combines agile principles with organisational patterns as a potentially efficient means to build embedded real-time systems that have stringent constraints.

Concurrent development

Speed is an important success factor for software product development [Smith & Reinertsen, 1998]. But this can be difficult to achieve; for instance, even a small bug can take a long time to correct. This is one of many reasons that make it difficult to determine accurately how long it will take to complete a piece of software [Dingsøyr and Conradi, 2002].

One way to accelerate product development is to compress development workflows by performing normally sequential *work* to some extent in a parallel (overlapping) manner, or even to skip some intermediate steps [Heikkinen,1997]. A difficulty with such concurrent development, however, is that some work is dependent on the completion of previous work, or cannot be accomplished with incomplete information that would otherwise be provided from a previous, dependant work. In other words, managing concurrent ES engineering tasks, and these kinds of interdependencies, can be difficult to perceive and achieve [Heikkinen, 1997].

Achieving product variety

Manufacturers of technical products are under pressure to produce a number of product series with an increasing number of variants, while simultaneously decreasing development costs low and time-to-market [Graaf *et al.*, 2003]. Jepsen *et al.* [2007] investigated minimally invasive methods for migrating products developed by a company to a software product lines approach, as a means to overcome these types of challenges. Larsson *et al.* [2007] describe a KMS that helps to reduce risks and that works in similar situations.

University of Cape Town

2.8.4.5 Tools for managing knowledge of software development processes

Generally speaking, an engineer does not have immediate access to (or know) implementation methods at the start of a project, except possibly in trivial cases where the engineer has experience with the specific platform and tools chosen for use in a project [Kitchenham *et al.*, 1995; Ganssle, 1999]. Implementation methods develop as the engineer's knowledge of implementation techniques, target hardware, development tools and other development artefacts grows. The development of implementation methods in ES involves a significant amount of knowledge work, which often involves time-consuming activities, such as reading datasheets, finding example code, discussing solutions with colleagues, and experimenting with code [Ganssle, 1999; Labrosse *et al.*, 2008].

A review of tools commonly used for managing knowledge of software development processes is provided below, including: 1) matrices for capturing and storing data, 2) representational models for modelling knowledge flows, 3) the concept of experience factories, and 4) case-based reasoning.

Matrices for capturing and storing data

A variety of matrices have been used to capture and store data related to knowledge about engineering processes. For example, a variety of 'Design Structure Matrices' (DSM) were developed for managing team-based knowledge, component-based knowledge, activity-based knowledge, and parameter-based knowledge [Zha & Du, 2006]. A DSM is a compact matrix representation of knowledge related to a development project. The matrix contains a list of all constituent subsystems and activities and the corresponding information exchange and dependency patterns. A DSM can be used by knowledge workers to find information pieces (or parameters) that assist at the start of a certain design activity, and show what information generated by the activity is likely to feed into it (i.e. which tasks within the matrix use the output information). The DSM can prove useful in managing complex projects, highlighting issues such as information needs, requirements and task sequencing [Eppinger *et al.*, 1997]. A web-based prototype system for modelling development process using a multi-tiered DSM was developed at MIT [Browning, 1999].

Representational models

Many representation approaches exist for modelling knowledge flows in engineering processes. These methods often use a high level of abstraction to categorise existing

University of Cape Town

knowledge and related experiences into a series of design principles and constraints. TRIZ is an example of this approach [Altshuller, 2004]. An alternate method is to partition design knowledge according to certain case descriptions; some case-based design tools use this approach [Wood & Agogino, 1996].

Researchers at the Engineering Design Centre at Lancaster University established a knowledge representation methodology and knowledge base vocabulary for mechatronic systems (used in their Schemebuilder tool) based on the theory of domains, design principles and computer modelling [Counsell *et al.*, 1999]. Blessing [1993] proposes the process-based support system (PROSUS) based on a model of the design process rather than the product. Another focal research area is using ontologies for product representation (e.g. Patil *et al.*, 2005).

The experience factory

Reusing experiences in the life cycle experience of processes and products for technical product development is often referred to as having an 'Experience Factory' [Houdek *et al.*, 1998; Basili *et al.*, 1994]. Using this approach, experience can be collected from different development projects and 'packaged' in an experience base (this 'packaging' process entails generalising, tailoring and formalising experience around reuse). A similar system, the CODE prototype, is a KM system that serves as a medium for knowledge capture and transfer, in which 'packaging' is also used to make knowledge from prior experiences more easily available [Skuce, 1995]. Examples of experience packages (based on Skuce [1995]) include: product packages (information about the lifecycle of a product and lessons learned); process packages (information on how to perform a process and reuse it); relationship packages (for analysis); tool packages (for use of a tool and related experiences); management packages (reference information for project managers); and data packages (containing data relevant to a technical product or its activities, such as a project databases [e.g., Dingsøyr & Conradi, 2002]).

Case-based reasoning

Case-based reasoning (CBR) is an approach that could facilitate technical decision-making during software development projects [Henniger, 1997a]. Many companies have used similar systems for retaining and retrieving experiences [Aamodt & Plaza, 1994]. Althoff *et al.* [1999] report on the benefits of this technology for experimental software engineering. CBR has also been used successfully in building learning organisations [Althoff *et al.*, 1999]. Wangenheim *et al.* [1998], who evaluated several

University of Cape Town

CBR approaches for experience reuse, found that CBR could be an effective means for reusing experience from software engineering. CBR generally uses a form of underlying experience database (or knowledge base); Broomé & Runeson [1999] reported on the important technical requirements for such a database. Bergmann *et al.* [2003] researched the feasibility of combining ideas of an experience factory with a CBR system.

A number of companies, such as Computas [Carlsen *et al.*, 1999], Hewlett Packard India [Bhave & Narendra, 2000], the COIN EF system in use at the Fraunhofer IESE [Tautz *et al.*, 2000], and a variety of Norwegian organisations [Conradi & Dingsøyr, 2000] have implemented a case-based KMS (or similar organisational learning approaches) for experience reuse, which have shown positive results. Another example of a CBR system, which was developed at a university, is the BORE system for problem-solving experiences [Henninger & Schlabach, 2001].

2.8.5 Managing innovation in technical product development

Innovation shifts the KM paradigm to one of 'new knowledge production', in other words, knowledge of a new way of doing something [McKeown, 2008]. Knowledge of an innovation or new technique for accomplishing something may involve collaboration and sharing of knowledge [Fischer, 2001], but the emphasis is on developing something new – possibly a new method to solve an existing problem, or a new technique to address a new issue [Fischer, 2001].

A principal question regarding the effective production of innovative knowledge is how this knowledge, and its associated innovation processes, is managed. Some authors suggest that innovation cannot be effectively managed [e.g., Kitchenham, 1998].

As mentioned in Section 2.4.2, KM involves the way in which knowledge is created, captured, stored, and shared, with the intention of leveraging this knowledge within an organisation [Rus *et al.*, 2001; Wiig, 1997]. Specific issues related to KM of innovation are discussed below:

1. *Technology*: Companies determine how to develop their information and communication technology (ICT) to facilitate innovation, and how their ICT can be customised for this kind of knowledge. Dhont [2003] uses the term 'playful technology' to describe technologies that enable innovation.

University of Cape Town

2. *People*: Companies need suitable training, recruiting and selection procedures to suit and encourage innovation; Nonaka & Takeeuchi [1995] highlight further issues of combination, socialisation, internalisation and externalisation of knowledge.

3. *Intra-organisation*: Supporting the innovative capacity of a company, with processes designed to foster 'creative competence development' is a core aspect of work performed [Miesing *et al.*, 2007].

4. *Inter-organisation*: Contacts are managed with customers and suppliers. Companies need to have an open mind with regard to the problems and needs of their customers and suppliers [Harvey & Speier, 2000].

The speed, competitiveness and overall success of companies are beginning to rely increasingly on the innovation and creativity of the company [Carneiro, 2000; Malhotra, 2007]. However, approaches to KM can confound innovation with an excessive amount of information and procedures [Dhont, 2003]. In such situations, KM may be seen as a centralised 'database' with a goal of collecting information about innovation. Majchrzak *et al.* [2003] point out that information dates quickly and that the emphasis should rather be on building and managing innovative capacity, which is likely to have greater value to a corporation. KM for innovation should focus on helping innovators to generate new ideas, to transform such ideas into working products, and to ensure that these capabilities are shared and strengthened among the creative teams of the organisation [Majchrzak *et al.*, 2000].

Gibbons *et al.* [1994] maintain that a new mode of knowledge production is emerging, which has profound implications for both the competitiveness and sustainability of product development. Gibbons *et al.* [1994] draw a distinction between two modes of knowledge production referred to as Mode 1 and Mode 2. Mode 1 knowledge relates to problems set and solved in a largely academic context (e.g., universities), whereas Mode 2 knowledge is carried out in a 'context of application' (e.g., in an industry context, such as a factory) [Gibbons *et al.,* 1994]. In the context of new product development, an innovative organisation is likely to use more Mode 2-type development and problem solving processes [Gibbons *et al.*, 1994]. Such an approach is likely to 1) constantly generate new ideas, 2) provide the

University of Cape Town

capacity to change new ideas into products, and 3) ensure that knowledge is disseminated to all the knowledge workers concerned [French & Bell, 1990].

2.8.5.1 Management of innovation issues in product development

Broadly speaking, forms of innovation in the context of technical product development can be divided into three main types: 1) process improvement models, 2) implementation of new technologies, and 3) process re-engineering [Vinter, 2005].

Techniques related to KM for innovation that have been identified as areas to develop further, and that currently show a significant level of interest (in terms of recent publications) include the following: 1) balancing creativity against systems [Counsell *et al.*, 2005], 2) methods for efficient technology transfer [Wandeler *et al.*, 2006], and 3) creative problem solving [Van Zolingen *et al.*, 2001]. These research areas, and the seminal work related to them, are described below.

Creativity vs. 'systems'

In their study of prototyping in five companies, Counsell *et al.* [2005] reported on the following main issues: 1) non-adherence to standard prototyping guidelines was common; 2) developers often engaged in 'sketchy' change request procedures; 3) there was frequent concern about time and cost deadlines not being met; and 4) developer experience was found to be an essential requirement for innovation. Clearly, a balance is required: an extremely formal KMS puts too much stress on knowledge workers who will then not be committed to the goals of KM, whereas an overly relaxed and informal approach may deny individuals the support they should get from the system.

In a paper reporting on the findings of a case study that explored micro level factors surrounding the processes of creativity and process management in a creative ISO certified organisation, Molnar and Nandhakumar [2007] argue that structured KM provides a framework for organisations. However, they also observed diverse opinions and attitudes towards KM inside organisations. Additionally, this framework results not only in positive effects but also in certain constraints for organisations. These constraints in turn influence the processes and innovations of organisations. The creative potential helps to overcome the given constraints of a structured process. Consequently, the paper claims that within a creative business environment it is essential to maintain these creative potentials [Molnar & Nandhakumar, 2007].

University of Cape Town

Technology transfer

The accelerating development of technology transfer and knowledge diffusion activities can have a profound impact on technology oriented companies and their potential for innovation, also supporting their need for this work to be carried out more rapidly to meet market demands [Zander & Kogut, 1995; Ernst & Kim, 2002; Pérez & Sanchez, 2003].

Creative problem solving

Innovation is strongly linked to problem solving [Van Zolingen *et al.*, 2001] and this is particularly evident in ES product development, for example, developing fast and reliable digital signal processing code that runs on resource-limited hardware [White, 2005].

2.8.5.2 Tools for managing innovation in product development

A number of tools have been developed to assist the process of innovation. These tools include methods to overcome mental inertia [Souchkov *et al.*, 2005], strategies to improve personal creative skills [Tierney & Farmer, 2002], heuristics that avoid psychological inertia during problem solving [Nakagawa *et al.*, 2002], reengineering techniques, and guidelines for novel product conceptualisation and development [e.g., Blosiu, 1999; Altshuller, 1998]. A selection of tools that appear in the literature and that relate to the management of innovation knowledge is reported on below.

TRIZ

Perhaps the most widely used tool in the context of technology mapping for the transfer of technology is TRIZ (an abbreviation from the Russian for 'theory of inventive problem solving') [Altshuller, 2004]. TRIZ includes the following parts:

1. *Laws and trends of the technology evolution.* This part of TRIZ studies and formulates general trends of engineering system evolution.
2. *Problem solving techniques.* The techniques aim at building a problem model and producing recommendations on how to solve the problem.
3. *Principles for the elimination of technical contradictions.*
4. *Inventive standards,* which solve inventive problems by representing them in terms of substance-field interactions and applying generic patterns for interaction transformations.

University of Cape Town

5. *Pointers to effects*. This part of TRIZ focuses on studying how to use the knowledge of natural sciences (physics, chemistry, geometry) in the inventive process.
6. *Algorithm of inventive problem solving.* An integrated technique aimed at solving the most difficult inventive problems that contain physical contradictions.
7. *Collections of selected patents*. This part contains patent descriptions drawn from diverse engineering domains. The patents are structured according to inventive principles used to eliminate one or other types of contradiction.

Reverse engineering

Reverse engineering (RE) is the process of discovering the technological principles of a device, object or system through analysis of its structure, function and operation [Muller *et al.*, 2000]. RE often involves taking apart a product, such as an electronic product or software program[3], and then performing a detailed analysis on its operation. Usually this strategy involves making a new product that provides many (if not all) of the same features as the original product, but without performing a verbatim copy of the original product [Chikofsky & Cross, 1990]. The key to innovation in RE is to recombine the knowledge and routines in efficient ways according to the current situation [Ståhle & Grönroos, 2000].

A number of different RE approaches are used, including Function Analysis (a modified version of traditional Value Engineering Analysis with the focus on functional decomposition and analysis of design products and technologies) and Ideal Modelling (of function-based redesign, also known as 'trimming'), which aims to perfect the product design and to formulate new problems (i.e., ones for which innovative new solutions are needed) [Warden, 1992].

Technology mapping

Technology mapping is a high-level planning tool for supporting the planning and control of technology management, and for supporting national and sector 'foresight' initiatives [Coombs & Hull, 1998]. However, existing models of technology transfer have been investigated separately with regard to the methods, stakeholders, and the

---

[3] In terms of a software program, this may be working with the original source code (which may lack documentation) or compiled machine code.

University of Cape Town

elements of how the transferors and transferees assimilate, adopt, develop, and transfer technology in various organisational settings [Simonin, 1999; Argote & Ingram, 2000]. Chan and Yu [2004] investigated a structured approach, which makes use of knowledge networks, for analysing the technological merit of innovative products and their impact on a value chain.

Cummings and Teng [2004] claim that a company should recognise that the development of knowledge repositories, portals connecting with external databases, and alerts about recent developments in the field will be of limited value for knowledge transfer. They recommend that companies should supplement these activities with mechanisms for connecting people to people through web conferences, communities of practice, discussion boards, chat rooms, instant messaging, and other forms of expertise identifiers and expert collaboration, all of which are useful for knowledge to be transferred effectively.

Tools for supporting emerging innovative processes

Business processes, especially those in knowledge intensive environments, often emerge organically, rather than following predictable and predefined steps [Covin & Slevin, 1989; Sabat, 2007]. Supporting emergent processes is one of the key issues for collaborative knowledge sharing. Luqi *et al.* [2004] introduce WorkPath, which is a component-based workspace meta-model used to support emergent processes. Key elements that construct workspace and WorkPath include knowledge worker roles, actions, artefacts, workspaces and reference relation methods [Luqi *et al.*, 2004].

2.8.6 Dealing with information overload

The term 'information overload' refers to situations in which a person is oversupplied with information, or when the amount of information exceeds the cognitive capability of a person [Ho & Tang, 2001]. Too much collection, sharing and distribution of information can lead to information overload and obstruction of work [Bouthillier & Shearer, 2002].

Dealing with information as a commodity does not necessitate knowledge production [Castells, 1996]. As the scenario in Section 2.4.1 shows, some kind of processing and interpretation, which depends on the prior competencies and experience of the individual concerned, are needed to make effective use of information. Information can be comparatively easy to obtain compared to the effort involved in making use of it [Oluic-Vukovic, 2001]. This is particularly true for information found on the internet.

University of Cape Town

However, if a resource is available on the internet, this is not necessarily any guarantee of its importance, accuracy, utility or value [Berghel, 1997].

This section briefly outlines major issues related to information overload associated with the application and development of technology, namely:

1. Dimensions of information overload; and
2. Reducing information using infomediary tools

2.8.6.1 Dimensions of information overload

Ho and Tang [2001] argue that there are three dimensions to information overload: 1) information quantity, 2) information format, and 3) information quality. Information quality relates to the amount of information for a particular activity (e.g., results returned for a web search). Information format can relate to the way in which information is presented to the user (e.g., the layout of a text document and the use of tables and images in the document), and to the way in which the information is stored (e.g., the document in the form of a Microsoft Word document or a Latex file; the type of executable file format of a program; and the image file format used to store an image). Information quality is a more subjective measure of these three dimensions, concerning issues such as the usefulness of the information provided and the difficulty involved in interpreting the document that contains the needed information.

2.8.6.2 Addressing information overloading with infomediary tools

An infomediary tool is designed to cut down on the problem of information overload by acting in a similar way to that of an 'intermediary' or a 'filter' by attempting to reduce the amount of information presented to a user, and to impart the information in a manner that is more usable (for a particular purpose) than the original information sources in their 'raw' states (e.g., web pages or online articles) [Ordanini & Pol, 2001]. For example, a comparison infomediary tool analyses a set of documents and presents a summary of the differences found between them [Ho & Tang, 2001].

Ho and Tang [2001] investigated a variety of commonly used infomediary tools and identified the information overloading dimensions each approach seemed to address to reduce the problems of such information overload. Table 2.1 is an adaptation of

University of Cape Town

the results produced by Ho and Tang [2001, pg. 94]; each row corresponds to an informediary model and the information overload dimensions it addresses.

Table 2.1: Commonly used infomediary tools and dimensions of information overload they address [Ho & Tang, 2001; Berghel, 1997].

Infomediary tool	Description	Dimensions of information overload addressed
Portal	Typically a web service providing searching and yellow pages facilities, often classifying web-based information according to predefined subjects.	Assists in reducing *information quantity*.
Virtual community	A community of people that share common interests and techniques through e-mail, chatting, newsgroups, or discussion boards.	Assists in resolving *information quantity* by focusing on one specific topics of interest. Voting and ranking features helps with *information quality*.
Transaction aggregator	Electronic marketplaces that enable customers to connect with each other and purchase products in the same place (or on the same website).	These services (e.g., Amazon.com) help to reduce time in finding information about products and buying the product, addressing *information quantity* and *quality*.
Syndication	Collects and packages digital information from many sources that use potentially dissimilar information formats or representation standards.	Addresses issues of *information quantity* and *information format*.
Personalisation	Helps in making (from a potentially large set of options) a selection of information that is of most value to an individual.	Mainly addresses *information quantity*, e.g. automatically navigating multiple web sites to manage a person's information.
Comparison	Searches in real time across many websites (e.g. retailer sites) giving comparative information. For example, showing different prices and features for a certain make of car sold by online retailers.	Assists in reducing *information quantity* and *information quality*.

University of Cape Town

2.9 Conceptual framework for researching a KMS

The most important part of implementing a KMS is the strategising and establishment of effective guidelines and infrastructure that knowledge workers will use to carry out KM activities in the organisation (this was elaborated in Sections 2.4 and 2.6). The overarching purpose for the KMS is typically to make the organisation concerned operate more intelligently in order to secure its continual viability and success. These guiding principals are emphasised by Edvinsson & Malone [1997] (see Section 2.6) and is corroborated by many other researchers, such as work by Karadsheh *et al.* [2009] in which they formulated a generalised theoretical framework for KM of organisational processes. Karadsheh *et al.* [2009] showed that the overall objective

of a KMS is also commonly viewed as an approach to implementing KM strategies that somehow lead to knowledge workers saving time, effort or costs, and preferably a combination of all three.

The objective of the ESAOA KMS is guided by the KM literature guidelines as discussed above, which are predominately related to KM at the organisational level, or *macro level* as defined by [House *et al.*, 1995] (see Section 1.2.4). However, the ESAOA KMS focuses on *meso level* processes. For this reason, the above-mentioned broad objective of a KMS needs to be narrowed down to the appropriate level of KM work that the ESAOA KMS will be used with. In the case of the ESAOA KMS, the KM work of focus is ESAOA activities that are performed by novice engineers to produce knowledge needed to complete implementation tasks during ES prototyping projects. Thus, the objective of the ESAOA KMS is to facilitate knowledge production during ESAOA activities, to promote successful completion of ES implementation tasks.

Since ESAOA activities focus on changing and classifying artefacts that may be further adapted by the same or different individuals at a later stage, the ESAOA KMS design has been chosen to follow the *personalisation* trait of KM [Hansen, 1999] rather than the *codification* trait (i.e., having a focus on facilitating the flow of information about knowledge sources, as apposed to how knowledge is captured – see definitions in Section 2.6.1). Furthermore, the ESAOA KMS will follow the *information-focused stream* of KM, comprising KM methods that focus on the way knowledge workers treat information and the interaction between people interested in sharing information [McDermott, 1999a] (see Section 2.4.2).

Researchers, such Wiig [1997], Davenport [2002] and others, caution that the establishment of a KMS is not straightforward; it could take months or years to complete (as discussed in Section 2.6). Consequently, these researchers commonly recommend an incremental and iterative approach to the KMS development. In addition, as discussed by Alavi & Leidner [1999], an understanding of the knowledge work that the KMS will be used for, is needed prior to imposing the new KM strategies into an organisation. Methods to measure the effectiveness of the KM techniques implemented and their influence on the organisation is also required (see Section 2.6.3). These measurements could be done independently, such as in the form of a 'knowledge audit' [Groff & Jones, 2003] applied to knowledge artefacts that are produced, or by using other techniques; depending on the organisation and

University of Cape Town

knowledge work involved, these techniques could include surveys [Gold *et al.*, 2001], employee performance evaluations and product quality inspections [Alavi & Leidner, 2001] among other methods.

Following the above recommendation of expert researchers, the ESAOA KMS will be developed incrementally. Accordingly, a study will first be performed to gain a clear understanding of the specific knowledge work involved, together with establishing techniques to measure KM for the context involved. Next, appropriate KM methods will be planned and the ESAOA KMS constructed. This will be followed by putting the new KMS into operation, and gathering data to determine in which ways, and to what extent, the KMS benefits knowledge workers. Further revisions to the KMS can then be performed to incrementally develop increasingly improved versions of the KMS.

The strategy described above was used in planning the research design for this thesis (presented in Chapter 3). The research design has been implemented in the form of an experimental methodology involving two experiments. The first experiment corresponds to the study of knowledge work, whereby data will be obtained from developers using their own *ad hoc* KM methods (i.e., they will not using a KMS). KM measurement techniques will then be developed based on findings from the experiment and through the use of relevant works from the KM literature. Next, the first version of the ESAOA KMS will be developed, and then tested during the second experiment. Future refinements to the KMS will then be formed.

The next chapter takes the fundamental framework described in this section and refines it into the comprehensive research methodology used for this thesis.

University of Cape Town

2.10 Summary and conclusion

This chapter outlined KM needs, strategies and tools that are of relevance to ES projects in product development environments, if not directly in the field of ES product development (for which this specific form of literature is limited [Rus *et al.*, 2001]). The literature indicates that managing knowledge in technical product development is dependent on the related systems, infrastructures, and other context-specific sources and sites where knowledge is developed and where learning takes place (see Section 2.8). The literature suggests that there are four main considerations when developing a KMS, in particular:

1. Knowledge workers and knowledge-based teams;
2. Data and information finding tools and systems;
3. Knowledge-based engineering processes; and
4. Innovation, knowledge production and creative problem solving.

Each of these knowledge-based aspects concerning product development requires different forms of KM and KM tools (discussed in Section 2.8.2). Within each of these areas, there is likely to be explicit as well as tacit knowledge, and tangible as well as intangible knowledge assets – all of which require appropriate forms of KM.

The literature review has demonstrated that there are many gaps in the research literature on the implementation of KM systems and tools in ES engineering (as elaborated upon in Section 2.1) and particularly in terms of KM for technical development (as seen in Section 2.8). This gap in the literature confirms the need for more studies focused on KM for ES product development.

This chapter has shown that managing intellectual capital (IC), intellectual property (IP) and innovation is becoming increasingly important in the ES field (see Section 2.3), not least because of the knowledge-intensive nature of this work. In short, and as can be generally seen in this literature review, KM for ES product development should be an integral part of the activities for efficient ES product development organisations working in today's turbulent business environments.

University of Cape Town

University of Cape Town

Chapter 3:

Researching embedded system artefact organisation and adaptation (ESAOA) knowledge

This chapter explains the research design and methods for *evolving* a knowledge management system (KMS) for embedded system artefact organisation and adaptation (ESAOA) knowledge.

Section 3.1 describes the key concepts and specialised terms used in this chapter. Thereafter, the research objective, which was outlined in Section 1.3, is recapped and elaborated on in Section 3.2. The main research problems associated with studies concerning the research objective, and the ways in which these problems are addressed in this thesis, are discussed in Section 3.3. The problem statement, together with its associated sub-problems is then described in more detail in Section 3.4. The experimental research design for the ESAOA KMS is presented in Section 3.5. The selection criteria for the experiments, such as the development projects, sites and participants, are outlined in Section 3.6, whereas ethical concerns are addressed in Section 3.7. Data collection methods are described in Sections 3.8, and figures are provided to show samples of collected data. The data analysis strategy is given in Section 3.9. The data synthesis procedures are explained in Section 3.10, which involved merging and generalising the detailed results from the data analysis process to obtain a more abstract view of the KM processes. Section 3.11 presents the ESAOA conceptual modelling language that is used in the data synthesis phase. Section 3.12 concerns the artefact and prototype assessment that was performed at the end of the experiment and provided a means to compare the KM results to the product prototype and artefacts produced by the teams. The final section, Section 3.13, explains how the research findings are arranged in the subsequent chapters.

University of Cape Town

3.1 Key concepts

A KMS is the way in which people, processes and artefacts work together for the creation, capture, storage and sharing of knowledge [Drucker, 1998]. The actual design of such a KMS depends on the type of knowledge work being done, and the context within which it is performed (see Section 2.6). The study of a KMS typically focuses on the way in which the 'know-how' and 'know-who' of knowledge workers is managed in an organisation to solve problems, make decisions, learn facts, and find the most knowledgeable people in certain specialised areas within the organisation. Chapter 2, for instance, has already elaborated on KM and the contexts of ES product development.

3.1.1 ESAOA knowledge

The term *ESAOA knowledge* is defined as knowledge that is needed to carry out ESAOA activities (see also Section 4.2.5). ESAOA knowledge is a form of implementation knowledge produced and used during the implementation phase [Schach, 2005] of ES development projects. It is technical knowledge that is related to the specific types of implementation activities, which are the concern of this thesis (see Section 2.2 for details concerning the implementation phase of ES development). As was described earlier, implementation is the process by which a developer transforms a product design into a physical product [Schach, 2005]. Implementation knowledge is defined in this study as knowledge that developers use during the implementation phase of a project in which they are actively involved in building a product.

Generally, implementation knowledge encompasses the extremely wide range of competencies that a developer uses to perform implementation tasks. For example, in the case of ES development, implementation knowledge includes the engineer's knowledge of back-end tools used in developing software (e.g., the ability to use text editors and web browsers), and techniques for programming-in-the-many (i.e., strategies for working in a team of developers) [Bendix, 1993; Medvidovic & Mikic-Rakic, 2003; Schach, 2005]. This thesis concerns implementation knowledge that relates specifically to the engineer's understanding of, and his/her ability to select and use, development tools and components to perform implementation activities. Development tools include the compilers, linkers, oscilloscopes, multimeters, and other software programs and equipment that ES engineers use to develop ES. The components used to construct an ES comprise both electronic components (e.g.,

University of Cape Town

microchips, switches and displays) and software components (e.g., embedded operating systems and code modules).

Although implementation knowledge is generally associated with the implementation phase of a development project, such knowledge can be produced and applied at any point in a project. It therefore encompasses a broad expanse of knowledge. ESAOA knowledge, which is the focus of this thesis, consequently refers to a smaller subset of implementation knowledge, specifically implementation knowledge that is used to carry out ESAOA activities (i.e., knowing how to organise and adapt ES artefacts).

3.1.2 Towards a study of directed KMS evolution

Developing a KMS for a certain context of knowledge work is not a simple undertaking that can be accomplished in a few hours (as can be seen in initiatives studied by researchers such as Wiig [1997], Alavi & Leidner [1999] and Davenport [2002], and further discussed in Section 2.6). Rather, a KMS tends to be developed over time for the specific context in which it is to be used [Hu & Chen, 2002]. Such a context often has an elaborate arrangement of dependent factors, involving the actual knowledge work, the specific organisation involved, and its working environment. Furthermore, knowledge work also changes over time in response to a variety of influences, which results in the KMS used by knowledge workers being in constant flux [Carneiro, 2000; Bergeron, 2003].

The term *KMS evolution* is used in this thesis to encapsulate the process by which a team of developers moves from initially using an *ad hoc* KMS towards using a newer and more *evolved* KMS. As explained in Section 2.6.2, the process of evolving a KMS involves gradually changing an existing KMS towards an increasingly more evolved system, in which the system becomes more visible, more consistently applied across different projects and individuals, and generally better understood by the people who use it. KMS evolution can be considered as an ongoing process that starts when a certain group of knowledge workers begins to collaborate, and that ends when the group stops work and separates.

3.1.3 Directed KMS evolution

For the sake of clarity, the concept of *directed KMS evolution* refers to KMS evolution that is consciously studied and controlled, in contrast to a KMS that evolves naturally as an unobserved phenomenon. Customised KM tools and personnel assigned to

University of Cape Town

specialised KM roles are considered to be part of a more *refined*, or more highly evolved, KMS (as per the sixth phase of evolving a KMS defined in Section 2.6.3). Figure 3.1 illustrates directed KMS evolution, showing that an *ad hoc* KMS is purposefully monitored and changed to produce a refined KMS. The term 'directed KMS evolution' thus concisely describes the focus of this chapter, which is to describe a methodology for directed evaluation of an ESAOA KMS.

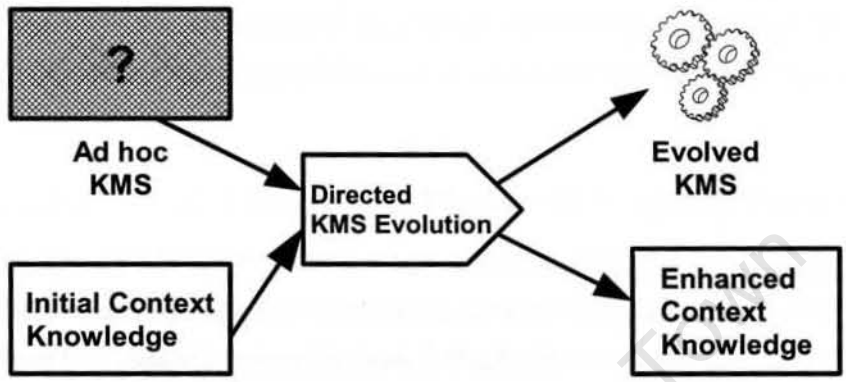


Figure 3.1: Directed KMS evolution.

3.2 Research objective: A KMS for ESAOA activities

In this thesis, knowledge is defined to reside exclusively within a person's mind (see Section 2.4.1) [Groff & Jones, 2003], which makes it difficult to measure. For this reason, the emphasis of the research objective is not on knowledge itself, but rather on the KM methods and *ESAOA activities* (defined in Section 1.1.7) that relate to a KMS. The research objective of this thesis can thus be expressed as follows:

> **Research Objective**
> The objective of this thesis is the construction, evaluation and evolution of an experimental KMS, which is referred to as the ESAOA KMS.

The ESAOA KMS is intended for use in ESAOA activities within the context of new projects that involve prototyping novel ES products (see Section 1.5 for details). The aim of the ESAOA KMS is to facilitate the progress from the use of an *ad hoc* KMS towards a more highly evolved KMS in which KM strategies are made visible and applied systematically (see thesis objective introduced in Section 1.3).

The subsections below further motivate and refine the specific objective of this thesis together with the scope and delimitations imposed on this work.

University of Cape Town

3.2.1 Specific objective: Moving from an *ad hoc* to a formalised KMS

Experts on software methodology, such as Brinkkemper *et al.* [1996], Kroll & Kruchten [2003] and Jacobson *et al.* [1999], indicate that processes need a suitable level of 'ceremony' for the work and working environment involved. A high level of ceremony implies processes have "comprehensive supporting documentation and traceability maintenance among artefacts" [Kroll & Kruchten, 2003, pg. 50]. In contrast, a low level of ceremony implies less supporting documentation and little formalism in the working procedure. A formal KMS involves certain costs and manual overheads to carry out KM methods while knowledge work is performed [O'Dell *et al.*, 2003].

As is the case with the representation of software methodologies, a KMS also needs a suitable level of ceremony in order to make it expressive and usable. Likewise, the methodology for evolving a KMS for ESAOA knowledge needs to be expressed and evaluated at an appropriate level of detail, while imposing a minimal amount of additional manual overheads for the knowledge workers concerned: in other words, representing the KMS with either too much, or too little detail can result in the system being ineffective and not applied consistently amongst knowledge workers [Kettunen & Laanti, 2005].

An additional challenge, in terms of instituting a formal KMS, is that the KM overheads of an *ad hoc* KMS may appear to be less than those of a more formalised KMS; such an opinion may be due to the perception that the KM methods of an *ad hoc* KMS are invisible, while those of a formalised KMS are more visible and therefore appear to impose more overheads [Lynn, Reilly & Akgűn, 2000]. As a result, engineers may feel their own *ad hoc* methods are better than the formalised methods imposed on them. However, KMS can provide longer term benefits that benefit the project later, or during product maintenance [Alavi & Leidner, 2001].

In consideration of the above issues, the construction, evaluation and evolution of the experimental ESAOA KMS to be constructed needs to involve suitable levels of ceremony and process descriptions, while simultaneously avoiding excessive overheads, to ensure its usability for consistent application among developers.

University of Cape Town

3.2.2 Scope and delimitation: ESAOA during component integration

The boundaries of a KM initiative need to be established so that the effort expended in studying, formalising, and adjusting the system is restricted; this can be accomplished by defining the scope of the initiative [Groff & Jones, 2003]. Such a definition of scope includes, among others, describing the objectives of the KM initiatives (or work related to those initiatives), the timeframes, and the number of people studied.

Specific delimitations identified for this study are outlined in the subsections that follow. The broad areas of delimitation were guided by Groff & Jones [2003].

3.2.2.1 Delimitation of tasks

An important delimitation chosen in this study is the restriction on the types of knowledge work to be studied. For example, when studying the KMS used by software engineers, a scope limitation may be to focus on a specific type of engineering task, and to exclude administrative, marketing, and more basic project-related tasks, which together may accumulate a volume of data that cannot be handled within the limited time of such a study.

Due to the broad extent of implementation knowledge that is produced in a project, this thesis is restricted to a subset of implementation knowledge that is generated during the development of ES products: in particular, it focuses on implementation tasks in which engineers learn how to use development tools and components. The emphasis is on managing ESAOA knowledge, which is closely associated with an engineer's knowledge of the development tools and product components used to construct an ES. More specifically, the research design focuses on how developers organise and adapt implementation artefacts to create, capture, store and share knowledge of product components, as well as on the use of development tools to implement a product.

In order to focus on ESAOA knowledge, a research strategy is needed to isolate this knowledge and its related activities from other forms of knowledge and development work that occurs during projects. Such a separation is an analytical tool because knowledge itself is an abstract phenomenon that can be viewed as the complex interaction of an individual's history of experiences [German & Hindle, 2006]. Experience can be obtained in many contexts, such as reading, talking, and performing experiments in a laboratory; for many forms of knowledge work, the

University of Cape Town

experiences that give rise to knowledge are not limited to the context in which knowledge work is performed. Based on these insights, it is important to recognise that the separation of knowledge into knowledge categories is an analytical strategy (other analytical strategies include methods such as performance analysis [Kettinger et al., 1997] and examination of knowledge objects [Knorr-Cetina, 1997]).

ESAOA knowledge is isolated from other forms of implementation knowledge using the definitions below. The first definition delineates the concept of routine development tasks. The second definition describes implementation knowledge in relation to these routine development tasks. The third definition explains artefact organisation and adaptation (AOA) knowledge based on the first two definitions. Thereafter, a concise definition for ESAOA knowledge, a specialised form of AOA knowledge, is given.

> *Definition 3.1: Routine development tasks* are low-level practices common to projects in a certain field of development. An example of a routine development task in the field of software development, for instance, is writing code for a program procedure using a text editor.

> *Definition 3.2*: *Implementation knowledge* is used by a developer to incorporate a component into an incomplete product using development tools, or to use development tools to adjust the ways in which a component is built into a product (or part of a product). An example of implementation knowledge would involve knowing the sequence of operating system services to call in a function to configure an interrupt service routine for a particular microcontroller. Implementation knowledge builds on the broader and more fundamental knowledge used in routine development tasks.

> *Definition 3.3:* Artefact organisation and adaptation (AOA) knowledge concerns an engineer's understanding and ability to organise (i.e., classify and structure) implementation artefacts, as well as the engineer's understanding of and capability to adapt these artefacts during the implementation of a product. Examples of AOA knowledge include understanding the logic behind the structure of a project directory, and knowing how a particular artefact was adapted to incorporate it into the product being built.

University of Cape Town

From the definition of AOA knowledge above, a more refined version of AOA knowledge restricted specifically to the implementation phase of an ES can be created. This knowledge form is the primary area of interest of this study.

> *Definition 3.4*: Embedded system artefact organisation and adaptation knowledge (or *ESAOA knowledge*) is a specialised form of AOA knowledge that concerns an ES engineer's understanding and ability to organise and adapt implementation artefacts during the implementation phase of embedded system projects. ESAOA knowledge essentially focuses on knowing how ESAOA activities are performed.

Examples of ESAOA *activities* were given in Section 1.1.7, which mentioned activities of "creating a new C file to hold the start-up code", "making a directory called 'Code' to hold code files" and "saving the new file as 'start.c'". Examples of ESAOA *knowledge,* based on the aforementioned activities, would include comprehending the logic behind naming the file 'start.c', reasons for placing the file in the subdirectory named 'Code', and understanding of how code in the 'start.c' was implemented or adapted for the particular ES platform used.

3.2.2.2 Delimitation of time

An optimal timeframe for the study had to be developed. The first experiment delimited the time to approximately three months. This was found to be an insufficient time for complex ES product development. The timeframe for the second experiment was thus extended to eight months. This was found to be sufficient for the first iteration, or prototype building (see Chapter 5).

3.2.2.3 Delimitation of team size

An engineer's knowledge of implementation tasks results from his/her underlying skills that are essential to the field of development concerned [Caspi *et al.*, 2005; Grimheden & Törngren, 2005]. For example, in ES development, fundamental software development skills include the engineer's ability to write source code in a particular programming language, as well as his understanding of how the source code is turned into an executable object. The size of the ES development teams was restricted as a means to simplify the study and possibly to reduce the number of development techniques used in the project. Consequently, Experiment 1 comprised teams of two member each (in order reduce issues of complexity related to team

University of Cape Town

dynamics, since the experiment was also a form of pilot study to establish the initial KMS), whereas Experiment 2 comprised teams of three members each (in order to investigate projects involving more complex team dynamics).

3.3 Research problems

A KMS exists for any form of knowledge work. A KMS ranges from being entirely informal, invisible, and poorly understood, to being formalised, visible, and well understood. A KMS also varies from being useful to the knowledge workers who use it, to being an impractical system that is more of a hindrance than an aid to knowledge work. The nature of a KMS raises a number of issues that need to be addressed in developing appropriate research methods for studying KM. These issues are elaborated on in the sub-sections that follow, highlighting significant challenges that are likely to occur while researching KMS evolution.

3.3.1 Associative memory, time-limited knowledge, and repeated learning

Product development projects, and in particular complex projects such as ES development projects, draw on a significant amount of knowledge, a large portion of which is produced while working on the project [Cross, 1994]. The research design and data capturing methods need to consider this. Knowledge is produced through a process of interpreting information, which occurs in a variety of learning activities, such as reading, discussing, writing, and listening. However, the various ways in which knowledge is produced poses challenges for the data capturing methods.

Once an individual has acquired knowledge, that knowledge will not necessarily be retained by that individual for a long period of time. This is due to human memory being associative, meaning that memories are recalled in response to stimuli [Anderson & Bower, 1980]. A research method that was developed to address the difficulty of tacit knowledge, or knowledge-locked-in-memory, is that of the 'discourse-based interview' [Odell *et al.*, 1983], which uses stimuli to trigger memories, such as using objects or documents to elicit memories of experiences that involved that object.

Product development typically involves the use of many different types of information sources from which knowledge is produced [Kettunen, 2003]. These information sources include people (e.g., colleagues or trainers), documentation (e.g., journals,

University of Cape Town

datasheets, manuals), and Internet resources (e.g., forums and blogs). Due to the associative, and time-limited, nature of knowledge, knowledge workers need to use such information sources so that they can reproduce knowledge used previously in order to solve an immediate problem. This process of relearning knowledge may involve reviewing the same information sources as used for the original instances of learning, or studying alternative information sources (e.g., notes in a logbook, or comments in source code) that contain sufficient information to trigger the memory (or "to stimulate the mind to ... derive that knowledge" [Anderson & Bower, 1980, pg. 4]) and thus to reconstruct the original knowledge. Information sources, such as paper or digital documents, emails, and talking to people, are not the only means, or necessarily the only sufficient means, of reconstructing learning. Performing actions and observing results, such as changing the configuration of a tool and seeing what happens, are also means by which knowledge can be gained, or previously learned knowledge relearned. For these reasons, the different phases and versions of product development are useful data sources for KMS research.

Developers are generally aware of the time-limited nature of knowledge, understanding, for instance, that a complicated task that took many hours to complete, may have initially involved a significant amount of reading and experimentation, and that this may need to be repeated due to the knowledge from the original learning efforts being forgotten. For this reason, developers typically keep copies of the information sources that they used, and often keep logs of procedures that they used to solve problems. This recorded information not only saves time in locating original information sources, but it may be essential in reconstructing knowledge. For research purposes, the developers' logs form an important data source.

University of Cape Town

3.3.2 Information overload

The need to reproduce knowledge leads to developers justifiably retaining documentation, saving web pages, making logs, archiving software tools, and collecting email, amongst other tasks that involve retaining information. Although the retention of information is highly desirable from a research point of view, it can have some detrimental consequences for product development. For example, a developer may become overwhelmed by the sheer amount of information retained for a project, making the task of reproducing knowledge extremely time-consuming. This situation is obviously worse for a developer maintaining a project developed by someone else [Aman *et al.*, 2006].

3.3.3 Research challenges: Confidence, confusion, and lost property

ES engineers involved in the complex process of developing embedded products do not always have time to take perform tasks to facilitate research processes applied to their work. This problem is particularly apparent when the design process is rushed, as often happens close to a milestone delivery date. During the design process, there are inevitably lost documents, unlogged changes, unrecorded trials, and other activities that the researcher may not be made aware of. There are further complications, in terms of the people studied, which can influence the research process, such as the extent to which an engineer has read about the application domain for the product (e.g., for the development of medical systems, a developer that has studied medical systems may be at an advantage over those who have not). The researcher may also be unaware of the degree to which the engineers understand, or are confused by, their own processes. Kamsties & Rombach [1997] discuss these problems in depth in relation to researching requirements specifications for ES; their recommendations have influenced the research design described below (see Section 3.5 below).

This section has highlighted the research challenges, as well as ways in which these were addressed. The next section explains in greater depth the research focus and its associated sub-problems, research questions, and assumptions made.

University of Cape Town

3.4 Problem statement

This thesis argues that new projects that involve the development of novel ES products can be facilitated by the application of a specialised KMS, namely the ESAOA KMS, applied within the context of ESAOA activities. As specified in Section 3.2.2, this study focuses on the management of ESAOA knowledge performed during ESAOA activities.

3.4.1 Research question

As described in Section 3.2, the research objective focuses on the construction, evaluation and evolution of an experimental ESAOA KMS. Following the information-focused stream of KM (defined in Section 2.4.2), the ESAOA KMS needs to be structured appropriately for the effective capture, organisation, classification, and dissemination of ESAOA knowledge. Based on the generic design of a KMS given in Section 2.6.4, this means refining the various aspects of the KMS, such as the

people (or roles), activities and artefacts, so that the KMS as a whole can operate effectively. The following question was accordingly composed to guide this research:

> **Research question:**
> What is an effective structure for the ESAOA KMS (i.e., the roles, activities, artefacts, etc.) that will contribute to the successful completion of ES implementation tasks?

3.4.2 Sub-problems

In order to gain further insight into answering the above research question, the general problem statement, described at the start of Section 3.4, has been divided into a set of related sub-problems, which will be used to investigate the research question from different perspectives. Each sub-problem is described in the form of a sub-problem statement, and is accompanied by a short explanation for its inclusion. Each sub-problem was chosen to gain insights into directing the design of the ESAOA KMS, such as contrasting the effectiveness of different KM approaches, which in turn also help in gaining insights into how the research question can be addressed.

1. *Identify different forms of ESAOA knowledge*: The information stream of knowledge followed in this thesis (see Section 2.4.2) necessitates identifying which forms of knowledge occur in projects and how different methods are used to facilitate use of these dissimilar types of knowledge [Sveiby, 2000; Nonaka, 1994]. Consequently, this thesis will investigate the various forms of ESAOA knowledge that occur in ESAOA activities. Some ESAOA activities may make heavier use of a certain type of ESAOA knowledge, which will provide insights into specialised KM approaches for particular kinds of ESAOA activities.
2. *Determine the relative complexity of different ESAOA KM tasks:* KM tasks performed during some ESAOA activities may be more time consuming or complicated than those executed in other activities. Identifying the relative difficulty of these KM tasks will highlight areas of the KMS that can be optimised.
3. *Establish the relative difficulty of ESAOA knowledge production associated with different ESAOA knowledge forms (based on knowledge forms identified in sub-problem 1):* Some forms of ESAOA knowledge may be easier to

University of Cape Town

produce than others; knowing what forms of knowledge are more difficult to produce will make it possible to optimise the KMS design.

4. *Investigate how the time taken to complete different ESAOA activities may vary, and what factors may contribute to these variations:* A particular type of ESAOA activity may take on average a much longer time to complete successfully than other activities; identifying such activities will help to optimise scheduling of tasks in designing the KMS.
5. *Evaluate the relative frequency of different types of problem/solution occurrences in which ESAOA knowledge is used:* Some types of problem/solution occurrences, or sequences of problem-solving tasks, may tend to be more common than others. Insight into these will help in structuring KM tasks.
6. *Determine the portion of ESAOA activities that are incomplete or that have been abandoned:* Insights into these problems may indicate a lack of knowledge producers or poor knowledge producer/consumer relations in the KMS (as per the KM perspective presented by Milton [2005] and discussed in Section 2.7.1). These findings can provide indications where the KMS needs to be optimised, for example by providing team members additional training to help with communication tasks.
7. *Establish how the structure of a general KMS (see Section 2.6.4) is refined to become an ESAOA KMS:* This aspect focuses on refining the roles, groups, activities and other aspects as laid out in Section 2.6.4. A means of modelling the structure will be needed to aid explanations.
8. *Determine implementation tasks that benefit the most from an ESAOA KMS:* When implementation tasks involve a variety of ESAOA activities, they can be considered to be at a higher level. Identifying these general types of implementation tasks makes it possible to produce higher-level guides to be used in building a KMS.

3.4.3 Research assumption

It is assumed that the embedded software engineers will be able to use the ESAOA KMS to learn, recall, and express ESAOA knowledge that occurs during the implementation phase of embedded software development projects. This assumption is based on the general case of 'knowledge workers' [Drucker, 1998] being able to use a formalised KMS to learn, recall and express effectively the form of specialised knowledge for which the formalised KMS has been designed. Embedded software engineers are an example of such 'knowledge workers' [Conradi & Fuggetta, 2002].

University of Cape Town

3.5 Research design

The research design is intended to enable the study of the KM methods used by ES engineers to manage ESAOA knowledge in the process of developing a product. The scope of KMS evolution for the purposes of this thesis is limited to a single type of knowledge worker, namely, ES engineers.

Empirical research methods were applied in two experiments, which are respectively referred to as Experiment 1 and Experiment 2. The research methods are described in detail in this section. The research methods were developed following an experimental approach, which started by collecting data concerning KM methods from case studies based on data collection techniques used by Cross [2000; 2004] and Cross *et al.* [1996]. These techniques involved collecting data from a variety of sources in order to represent a broad spectrum of observations of designers at work. The data sources included minutes of meetings and records (e.g., reports) written by the designers themselves, among other sources. This strategy was used to enable observations to range "from the most direct contact with working designers to the most abstract and theoretical, such studies include the following types: interviews with designers, case studies of particular design projects, observations of designers at work, and protocol studies of design activity, laboratory experiments based on selected features..." [Cross 1990, pg. 130].

After data were collected from the first experiment, a preliminary study of the data was performed. The preliminary study involved observation of the data, with support from the literature, and influenced by approaches used by Cross [1994; 2004] and Cross *et al.* [1996]. The objective of the preliminary study was the development of data analysis methods for evaluating the KM techniques in the experiments (see Section 4.2). Section 3.9 describes the data analysis method.

University of Cape Town

3.5.1 Research design for evolving the ESAOA KMS

The research process followed the progression of capturing data, analysing it, and creating or refining the ESAOA KMS. The research design thus comprised a cycle of the following three parts:

1. Data are obtained from an experiment;
2. Data are analysed in order to gain insights into the ESAOA KMS;

3. The KMS is refined based on the results of the analysis. The cycle then repeats.

This cycle has been represented as the two separate but interacting phases of *KMS Framework Construction* (referred to as the 'C phase') and *KMS Analysis* (referred to as the 'A phase'). Data obtained from experiments were fed into the A phase of the cycle. As mentioned in Section 2.6.4, the framework of a KMS refers to the visible aspects of a KMS, such as supporting artefacts, tools, and role descriptions. Thus the C phase concerns obtaining and developing artefacts and documents that can be used to establish a KMS within a project team, whereas the A phase involves studying the resultant KMS in use by the team.

Two iterations of the A and C phases were performed for this thesis. Accordingly, two experiments, referred to as Experiment 1 and Experiment 2, were carried out to obtain data for the A phases. The preliminary study preceded the first A phase, and was used to develop the data analysis and KM measurement procedures that were later used in the A phases. Figure 3.2 illustrates the research design. As indicated in the figure, the same data from Experiment 1 was used in both the preliminary study and in the first A phase.

University of Cape Town

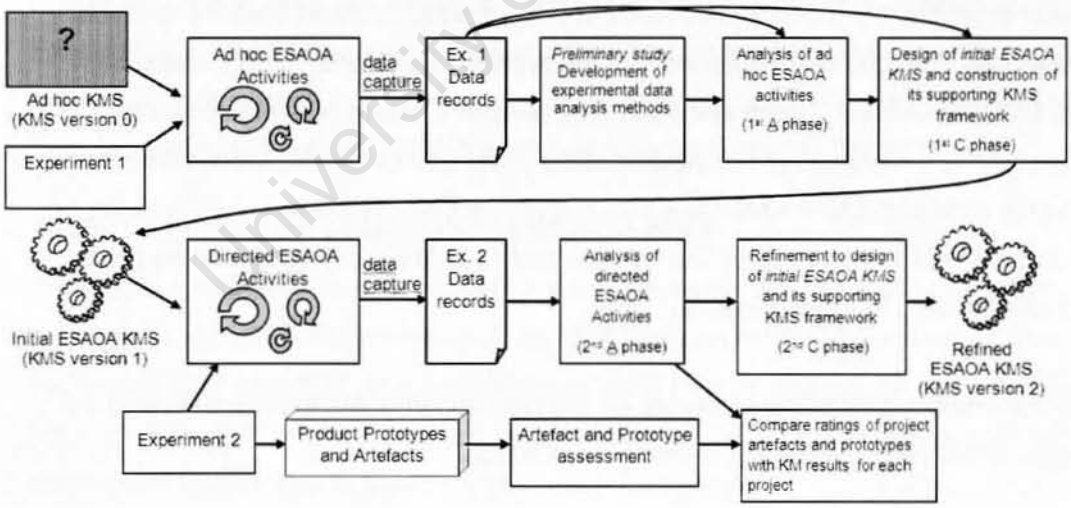


Figure 3.2: Research design for studying evolution of the ESAOA KMS.

The research began with Experiment 1 (see top left of Figure 3.2), which involved capturing data related to *ad hoc* KM strategies used by ES engineers during ESAOA activities. These *ad hoc* KMS strategies are represented in Figure 3.2 as the *ad hoc KMS* (or *KMS version 0*), which had no predefined design or framework. The ESAOA

activities are represented in Figure 3.2 as a block containing circular arrows (representing activities that the developers performed in the). Experiment 1 continued for three months.

Experiment 1 was followed by the preliminary study (see Section 4.2) in order to develop the analysis techniques applied in the first A phase. The first A phase was carried out using the same data as that used in the preliminary study. After this, the first C phase was carried out (the block on the top right of Figure 3.2), which involved designing the *initial ESAOA KMS* (or *KMS version 1*, shown as the gears on the bottom left of Figure 3.2), together with developing the tools and artefacts of its supporting *KMS framework*. This initial KMS framework comprised the artefacts and support structures that were part of the more coherent and visible ESAOA KMS, based on the *ad hoc* KM practices observed in Experiment 1.

After the initial ESAOA KMS had been constructed, Experiment 2 began, with 13 separate ES development teams using the KMS. This experiment continued for a period of eight months, in which the ESAOA activities were directed by means of the initial ESAOA KMS. At the end of the experiment, each team perform a product demonstration (see Section 3.8.6) and project artefacts of each team were assessed (see Section 3.12) – these assessments were later used to compare the quality of project artefacts, the product prototypes and the results obtained from the analysis phase. After this, the second iteration of the A phase was done, during which the data from Experiment 2 were analysed. The second *C* phase was subsequently carried out to develop a refined version of the initial ESAOA KMS, which was referred to as the *refined ESAOA KMS* (or *ESAOA KMS version 2*, illustrated by the gears on the bottom right of Figure 3.2). This refined ESAOA KMS provides a starting point that can be used in future projects.

University of Cape Town

The subsections that follow elaborate on the description of the experiments and the design of the various versions of the ESAOA KMS.

3.5.2 Overview of Experiment 1

As illustrated in Figure 3.2, Experiment 1 consisted of first capturing the data from *ad hoc* ESAOA activities, then analysing these activities, and lastly constructing the tools and other structural aspects (see Section 2.6.4) of the initial ESAOA KMS based on the results of the analysis. In this experiment, two ES product development projects were carried out by two separate teams of ES engineers, each of which

used their own *ad hoc* KM methods. The selection criteria for the ESAOA activities, sites, and participants used in the experiments are described in Section 3.6. Ethical considerations related to the experiments are description in Section 3.7. The data collection, data analysis, and data synthesis methods are respectively elaborated in Sections 3.8, 3.9, and 3.10.

3.5.3 Construction of the initial ESAOA KMS

The initial ESAOA KMS was developed in two steps. The first step involved exploring design strategies, modelling methods, and listing resource needs (i.e., identifying the parts needed for the KMS framework). This step also involved a requirements analysis [Kirikova & Grundspenkis, 2000; Broomé & Runeson, 1999] of the KM needs in projects, in addition to formulating the *ESAOA modelling language* (see Section 3.11) that would be used to describe the design of the initial ESAOA KMS.

The second step involved designing and implementing the structural aspects (i.e. the KMS framework) of the initial ESAOA KMS; this included selecting appropriate resources and software tools, as well as writing executable scripts and software programs to facilitate use of the ESAOA KMS.

University of Cape Town

3.5.4 Overview of Experiment 2

The initial ESAOA KMS was set up in each of the 13 ES teams who participated in Experiment 2, and the data were captured from this experiment over a period of eight months. The data were then studied in order to improve upon the initial ESAOA KMS, thus generating the *refined ESAOA KMS* and its accompanying framework.

3.5.5 Construction of the refined ESAOA KMS

Through a synthesis of the Experiment 2 data (detailed in Section 3.10), the effects of using the ESAOA KMS became clearer. KM methods that were either effective or ineffective were identified, and these findings were used to change the design of the initial ESAOA KMS in order to create a more refined ESAOA KMS, while also making corresponding changes to the underlying KMS framework.

3.6 Selection criteria: ESAOA activities

This section presents the selection criteria for the experiments (i.e., the ESAOA activities as shown in Figure 3.2 above), together with the summarised design briefs, the research sites, the choice of research participants, and the choice of reviews for

artefacts and prototype demonstrations. The section begins with a detailed overview of the ESAOA activities as research experiments, and motivates for the value of experimentation in obtaining sound empirical evidence related to ES KM methods [Kamsties & Rombach, 1997].

3.6.1 ESAOA project selection and project briefs

The development projects in Experiments 1 and 2 involved teams of novice engineers developing novel ES products. According to Cross [2004], novice designers typically follow more 'depth-first' practices than the 'breadth-first' approaches used by experienced individuals; accordingly, this research looks only at novice engineers (as emphasised in Section 1.6).

The selection criteria for ESAOA activities in both experiments were largely the same. For example, in both experiments the ES products had to be non-trivial, being of a type that might in fact be developed commercially. The conditions of development were thus made similar to those of commercial projects; however, due to the delimitations of this study, together with the time and budgetary constraints, a highly accurate simulation of a typical commercial context could not be attained. In both experiments, the engineers were required to install an embedded operating system, to program device drivers, and to write ES application code that used the device drivers and operating system services, which are commonly undertaken in commercial projects (see Section 2.2). These tasks are complex and encompass many activities generally found in *knowledge work* (see Section 2.7.4), including activities such as researching components (e.g., reading datasheets), looking for and understanding examples (e.g., finding and reading sample code), and testing development solutions (e.g., integrating and running sample code).

University of Cape Town

The projects followed similar stages in both Experiment 1 and Experiment 2, which are listed below. The first five stages occurred before the experiment proper was started. Data concerning ESAOA activities began to be captured at stages seven, and continue until stage twelve. The entire sequence took three months in the first experiment and eight months in the second. The stages are as follows:

1. Teams were organised.
2. The products were conceptualised through participant brainstorming and emailing of ideas.
3. Contracts were developed and specific projects assigned to teams.

4. Requirements were specified – teams worked on their specifications and met with the researcher to refine the requirements (all teams made a start at this stage to obtain reference documentation, such as looking up information related to the specific application domain for their proposed product).
5. *Review 1:* the first code and design review was conducted a two to three weeks after the teams had decided on a prototype to construct (this review focused on checking that the developers had a clear idea of what they planned to do, and to what extent they had creative ideas of how to do it). Teams received feedback from the researcher during and after this review.
6. The high-level design process was applied.
7. Implementation commenced, and the experiment proper was started (i.e. data were captured from ESAOA activities).
8. *Review 2:* the second code and design review of each team occurred shortly after the team had made the transition from working on the high-level design (such as block diagrams and UML class diagrams) toward working mainly on the implementation (e.g., building circuits and writing embedded software). Teams also received feedback during and after this review.
9. Product parts were integrated;
10. *Review 3:* the third review only occurred in Experiment 2 (as a refinement to the analysis process). The third review did not involve any face-to-face meetings; instead the researcher closely scrutinised project artefacts. Feedback of this review was *not* given to the teams.
11. Product demonstration took place, comprising a simulated acceptance test.
12. A review of the subsystems, components, and tools used was conducted (using a requirements check sheet). The experiment ended at this point.

Table 3.1 provides a list of the ESAOA activities studied, together with a brief description of each project and a motivation that explains why and how it approximates an actual commercial project. The numbering of the projects reflects whether they were done as part of the first or second experiment. Projects related to Experiment 1 (from which ESAOA KMS version 1 was constructed) were given project numbers P1-1 and P1-2. Projects related to Experiment 2 were numbered P2-1 to P2-13.

University of Cape Town

Table 3.1: Overview of the projects studied in Experiment 1 and 2.

Project No.	Project Title	Project Description	Motivation
P1-1	Software Signal Generator (SoSiG)	Programmable signal generator, able to communicate and to be programmed using a standard ASCII terminal over a serial interface.	Similar to products used by commercial developers.
P1-2	Antenna Controller (ANTCON)	Control system to control and monitor the azimuth and elevation of an antenna pedestal. Connects to parent radar control computer over TCP/IP using Ethernet.	Based on an existing RADAR control system used in military applications.
P2-1	Location-aware Tourist Information System (TIS)	The Tourist Information System is a global positioning system (GPS) device that allows a tourist to determine his/her current position and provides relevant information about the current area.	Similar to the idea of GPS-type navigation systems in wide use.
P2-2	GPS Bus Stop Navigator (GBT)	A GPS-based device that gives the closest bus station to the current position and the time of the next bus to a given destination. Provides a function to show the optimal path to the final destination.	Idea to extend GPS navigation system for use with public transportation.
P2-3	Vibynet	A 'Vibynet piece' is a compact electronic device carried or worn by people. It has an identity code and allows storage of other identity codes used to recognise other Vibynets pieces in the vicinity. Features: shows recognised identity codes, direction, distance, call.	Product to be used by a wide range of private individuals, similar to a 'beeper' product.
P2-4	MyIP Phone Station (MPS)	The voice over internet protocol (VoIP) answering machine with video is a stand-alone (non-PC) system that connects directly to your Ethernet. Answers any VoIP calls not answered manually within a certain time limit.	Product similar to a commercial telephone answering machine.
P2-5	Home Automation System (HAS)	Consists of a central unit that connects to other systems/household products around the house, such as input devices (keypads and sensors, etc.), and output devices (LCD screens, linear actuator, etc.).	Builds on the idea of 'intelligent home' products and service (see http://www.intelligenthome.com.au/)
P2-6	Automation Headlights Dimmer (AHD)	System will sense a bright on-coming light source and dim headlights until the light source has passed.	Similar to DIY kits advertised in *Popular Mechanics* magazines.
P2-7	Field Sensor for Maglev Trains (FSMT)	Based on the idea of using electromagnetic fields to levitate and drive high speed trains. Focused on a system for sensing electromagnetic fields and telling other systems.	A type of off-the-shelf component that a development company may buy.
P2-8	Cordless Stereo (CST)	A stereo that has no speakers but outputs digital audio for connection to Blue Tooth headphones.	Similar to commercially available Blue Tooth products.
P2-9	Central Alarm Clock (CAC)	Main controller based in a common room in the home. Has buttons that correspond to rooms in the house. Remote control to help parents wake kids up without moving an inch. Communicates over AC power lines.	Similar in concept to burglar alarm systems, which are commonplace products.
P2-10	Voice Activation System (VAS)	Voice recognition system to activate/deactivate/control electrical appliances (e.g. TV's, lights).	Similar to automatic dimmer switches that are readily available.

University of Cape Town

P2-11	Supermarket Query Device (SQD)	Device communicates with supermarket server, which holds database of items on special. Lets user browse through current specials, locate items of interest.	Comparable to electronic information directory found in shopping centres.
P2-12	Personal Protection Device (PPD)	Has one button that can be used to set off an alarm to notify campus/site security people that a certain individual, at a certain location, is in danger so that the closest security officers can be sent to the rescue. Device is small enough to attach to ID swipe card or to wear around neck.	A product that could replace a standard panic alarm system.
P2-13	Vehicle Usage Tracker (VUT)	Affordable vehicle usage monitoring kit (for DIY installation). Displays real-time car statistics, such as average speed and fuel consumption. Colour display and audio warning system.	Could be sold as a DIY kit, like others currently advertised in *Popular Mechanics* magazines.

3.6.2 Site selection

An ES laboratory, rather than a workplace, was chosen as the main site for the experiments. Conditions were created (including project meetings, deadlines, code and design review meetings and demonstrations [Jurison, 1999; Schach, 2005]) that would closely simulate conditions likely to be encountered in a commercial development context.

Laboratory conditions make it possible to ensure consistency and conformity across the experimental projects in terms of team size, role allocation, task difficulty, access to resources – and other factors that would impact on the reliability and validity of the research findings and the robustness of the knowledge produced by the research process [Brennan & Gupta, 1993; Törngren et al., 2007]. The logistics of managing the activities of 15 different teams are clearly facilitated by the laboratory conditions.

The choice of the laboratory as a site was also made for ethical reasons, namely to ensure the confidentiality of individuals and organisations, and to protect their intellectual property rights (see Section 3.7). For the same reason, simulations done in the laboratory were studied, rather than actual work done in private organisations.

Although all the developers worked mainly in the laboratory, they also had the option to take work home with them (however, they were not permitted to take any of the laboratory equipment or embedded platforms out of the lab). Work outside the lab was thus limited to activities such as coding and web searches (in Experiment 2, those developers who had chosen to do some project work outside the lab installed the necessary tools for the ESAOA KMS on their own computers).

University of Cape Town

3.6.3 Selection of embedded platform, cross-compilers, and IDE

The ES development process (elaborated in Section 2.2) involves using one or more microprocessor or microcontroller architectures, together with a collection of software development tools, which are used in conjunction with these processors.

In order to improve consistency in the results of this study, the same processor architecture was used in all projects in both experiments. In Experiment 1, the developers could choose their own cross-compiler tool chain (provided it was based on the open-source GNU Compiler Collection [GCC, 2008a]), embedded operating system and integrated development environment (IDE). However, in Experiment 2 all teams started with the same cross-compiler tool-chain, embedded operating system and IDE, as these aspects had been fixed by the initial ESAOA KMS. In both experiments, the developers were permitted to add other types of development tools to supplement the prescribed tools, such as computer-aided software engineering (CASE) tools to assist software development, and computer-aided design (CAD) programs for drawing schematics. Experiment 2 developers could furthermore modify or upgrade the development tools that were provided with the initial ESAOA KMS framework. This was permitted, as it was expected that the use of certain peripherals by a team might necessitate adapting the tools as well as making changes to other artefacts provided with the initial ESAOA KMS.

The cross-compilers and IDEs were required to be free or open-source programs that had versions available for Linux. All the embedded software was consequently built using cross-compilers that executed under the Linux operating system. Linux-based tools were chosen so that they could be installed on a centralised server and accessed remotely by the developers. This strategy was followed to facilitate remote access as well as to ensure that the project repositories and development tools were all kept in a centralised store (and so that it would thus be easy for the researcher to obtain a copy). In Experiment 2, an added advantage of this strategy was that it helped to reduce the lengthy process of installing and configuring the initial ESAOA KMS framework on each developer's computer workstation[1].

[1] In Experiment 2, developers were permitted to spend extra time configuring their own personal desktop or laptop computer with Linux and the same set of tools for the ESAOA KMS. These team members were responsible for ensuring that additional tools and resources they used on their own computers were not lost once the experiment ended.

University of Cape Town

3.6.3.1 The CSB337 embedded platform

The CSB337 evaluation board from Cogent Computers [2005] was used as the embedded platform in all projects. The CSB337 contained an ARM9-based microcontroller, specifically the AT91RM9200 manufactured by ATMEL [ATMEL, 2005]. The Advanced Risk Machine (ARM) architecture was chosen because none of the participants in the experiments had prior experience with this architecture. The CSB337 is illustrated in Figure 3.3.

Different application-specific hardware devices were used in the projects studied. This hardware was connected to the CSB337 board via standard hardware interfaces, such as SPI [Davis, 2008] and RS232 [Catsoulis, 2002], which were provided on the evaluation board.

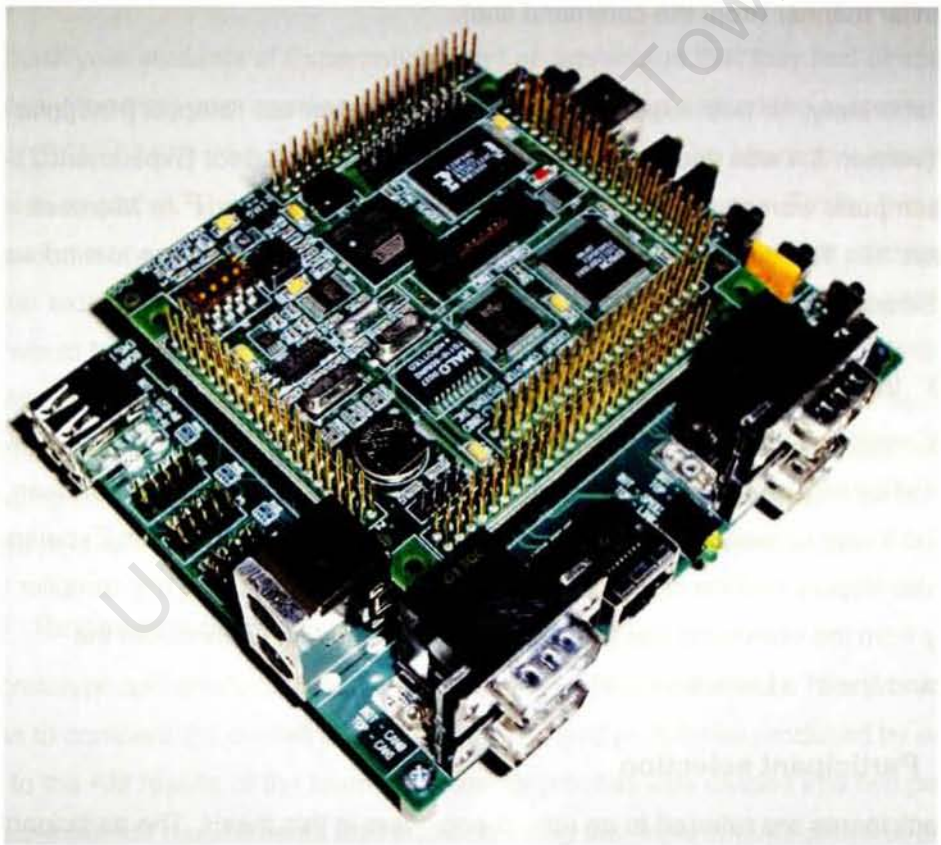


Figure 3.3: The CSB337 evaluation board produced by Cogent Computers.

3.6.3.2 GCC cross-compiler tool-chains

The GNU Compiler Collection (GCC) cross-compiler tool chain was used as the primary software development tool for all the projects. Two specialised versions of the GCC were prepared for use in Experiment 2; both versions were tailored for

compatibility with the AT91RM9200 microcontroller and the CSB337 evaluation board. The first version of the GCC (namely 'gcc-arm9tdmi') was prepared for compiling code that could run without an underlying embedded operating system, i.e. it was loaded and executed directly from the MicroMonitor boot monitor [Sutter, 2002]. The second version of the GCC (called 'arm-linux-gcc') was prepared for compiling code that executed on an embedded Linux operating system, namely uCLinux produced by SnapGear [SnapGear, 2007]. Both versions of GCC were initially prepared by developers in Experiment 1 as part of their projects; both teams independently chose the 'GCC crosstool', a framework developed by Kegel [2007] that assists in customising and building GCC tools for a wide range of processors.

Both versions of the GCC were rebuilt by the researcher and incorporated into the initial ESAOA KMS; they were integrated in a way that allowed them both to be used in a similar manner (from the command line).

In the laboratory, for both experiments, the central server ran Knoppix [Knoppix, 2009] (version 3.4 was used for Experiment 1 and version 3.6 for Experiment 2). Each computer workstation in the lab could dual-boot either Linux[2] or Microsoft Windows XP. The developers could access the central server and run *X-windows* tools [Scheifler & Gettys, 1986] from either operating system[3].

3.6.3.3 Integrated development environment

The 'KDevelop' integrated development environment (IDE) [Gehrmann *et al.*, 2004] was used by all projects (in Experiment 1 both teams had independently chosen this IDE, and it was consequently integrated into the initial ESAOA KMS). In Experiment 2, the developers had the choice of using either KDevelop or using the compiler tools directly from the command line (i.e., running make and text editors from the command line in a Linux terminal).

3.6.4 Participant selection

The participants are referred to as *novice engineers* in this thesis. The participants were all registered students in the electrical and computer engineering (ECE) programme at the University of Cape Town (UCT). Experiment 1 teams comprised undergraduate university students in their fourth year, each working on an embedded

[2] *Debian* 3.0 'woody' was used for Experiment 1 and *Ubuntu* 4.10 for Experiment 2.
[3] Regardless of which operating system the developers booted, they used their lab workstation as a front-end to the same Linux-based tools running on the central server.

University of Cape Town

system as part of their final year project (Appendix D.1 further details these teams). Experiment 1 comprised two teams of two members each. Experiment 2 teams comprised undergraduate university students in the third year, all enrolled in the embedded systems (EEE3074W) third year course at UCT (Appendix D.2 gives further details of the participants). Experiment 2 involved 39 students, divided into thirteen teams of three members each.

Experiment 1 used fourth year students because the third year embedded systems course had not yet been created at the time of the first experiment. Participants in both experiments were enrolled in the ECE programme. All participants completed the same set of first and second year core courses, including the prerequisites courses specified for EEE3074W, which ensured that all the participants had the necessary competencies to work on the embedded systems projects.

The fourth year students of Experiment 1 had an advantage that they had already completed the third year courses, with the exception of the embedded systems course (EEE3074W) that did not exist at the time (they took an alternative elective course in its place). This plan was intentional for two main reasons. Firstly, this study focuses on management of ESAOA knowledge – a skill that none of the participants in either experiment received formal training in. Secondly, the results of the ESAOA KMS would be strengthened (rather than diminished) if it is found that Experiment 2 participants, who had generally received one year less university education than the other participants, were found to achieve better results (i.e., if third years did better at managing knowledge with the KMS, despite the fourth years' additional year of education, it would then be clear that the KMS had beneficial influences).

University of Cape Town

3.6.5 Reviewer selection

The prototype and artefacts produced by each team were assessed. This provided a means to compare the overall quality of artefacts and prototypes produced by each team to the KM results of the team. This review process was divided into two parts: 1) assessment of requirements and artefacts using the *requirements check sheet* (e.g., rating the final version of code modules and design documents); and 2) evaluation of the final prototypes using a demonstration *check sheet*. The assessment criteria were based on the literature; see Section 3.8.6 for details. The *review panel* performed both assessments; this panel comprised of two individuals: the research and an electrical engineering Masters student with embedded systems experience. The second reviewer was included to reduce subjectivity, and to broaden

inputs, during reviews (this process was inspired by Briggs [1992] and Zingheim & Schuster [1995]).

3.7 Ethical considerations in the ESAOA activities

All the developers involved in the study were emailed a letter (see Figure 3.4), requesting that they agree to be part of the research project, and informing them that their anonymity would be maintained. The participants were assured that confidentiality will be maintained, and that their contact details would not be made public, and that any institutional or organisational affiliation they might have (besides an affiliation to the University of Cape Town) would not be specified.

The projects were structured as simulations, performed separately from other projects that the participants may have been involved with. In addition, the teams all worked within time limits (not more than an average of one day per week over a period of eight months) in order to ensure that the research activities did not interfere overly much with their other work.

University of Cape Town

Dear Sir/Madam:

Thank you for agreeing to participate in my PhD study. I will undertake to ensure your confidentiality by not revealing your name, contact details, or institutional/organisation affiliation. You are entitled to withdraw from this research project at any time.

By return of email please indicate your consent to participate in the research. Please feel free to raise any issues or concerns with me.

With thanks

Simon L Winberg

Figure 3.4: Consent letter emailed to participants in Experiments 1 and 2.

3.8 Data collection

Similar data collection processes were used in the first and second experiments. As mentioned in Section 3.5, this process of capturing of data from ESAOA activities was influenced by Cross [2000] and Cross *et al.* [1996]; the specific methods that were used are described in the subsections that follow. The data capture methods have been specialised towards the ESAOA activities described in Section 1.1.7.

There were seven main sources of data, namely: 1) code and design reviews; 2) email archives; 3) forums; 4) project meetings; 5) developer logs; 6) product demonstrations; and 7) End-of-project surveys. These data sources are detailed in Sections 3.8.1 to 3.8.7 respectively, and limitations of these data capture methods are described in Section 3.8.8.

3.8.1 Code and design reviews

There were two code and design reviews per project in Experiment 1 and three in Experiment 2 (i.e., 43 reviews all together); the reviews are referred to as Review 1 to Review 3. These took the form of both on-line collaboration and face-to-face meetings between the researcher and the members of the project teams (many of which correspond to meeting listed in Section 3.8.4) – except for the third review, in which teams either emailed a selection of artefacts to the researcher, or allowed the researcher to access their team artefacts remotely. Most of design reviews 1 and 2 were conducted in person between the researcher and team members, otherwise an on-line collaborations tool was used, namely 'Yahoo! Instant Messenger' [Yahoo! Messenger, 2004]. Digital design documents were accessed when needed via network file sharing.

The data produced in these reviews included notes of researcher-team member correspondence, documentation regarding code and design changes, activity logs, rating forms (for Experiment 2 only) and memos recording allocation of further tasks and responsibilities to team members. Figure 3.5 is an example of an activity log, showing the tasks carried out by team members.

#	*Person*	*Hrs*	*Title*	*Description*
1	James* + Mary*	1	Meeting 1	Meet to discuss project requirements and how to split the workload.
2	James	1.5	Task breakdown	Perform task breakdown as per project requirements
3	Mary	0.5	CVS repository	Set up the CVS repository for the group
...				

*All names are pseudonyms, to preserve the anonymity of participants

Figure 3.5: Example of activities log.

University of Cape Town

In Experiment 2, the researcher performed a quantitative rating of each team's deliverables for each design review.

The first review involved evaluating the overall creativity of the concept prototype proposed by the team; the proposed functionality and elegance of the planned prototype was also judged at this review (Appendix B.4.1 shows the evaluation form used). The criteria used for evaluating creativity were based on properties of creative projects described by Runco & Pritzker [1999], Sefton-Green & Sinker [2000], and Sophia [2006] – these properties were for projects in general; for the purpose of this study they were made specific to the ES field and expressed as rating criteria.

The second review concerned evaluating the design (a sample of the evaluation form is provided in Appendix B.4.2). The evaluation criteria were largely based on work by Douglass [1999], Liu [2000] and Schach [2005]. During the second design review, participants were asked to orally comment on knowledge production methods and information sources that they used in their project. Each team was asked to respond to the same set of three questions, which were designed to respectively gain insights into production of data knowledge, process knowledge, and innovation knowledge. A time limit of ten minutes was given to these questions. The researcher used handwritten point-form notes to record responses to the questions (Appendix B.5 presents a scanned copy of the question form). After the last design review, the researcher investigated all the notes and compiled a list of commonly occurring responses to each question (these are reported on in Section 5.4.1.2). These questions were handled orally to avoid team members spending valuable project time making sense of survey questions, possibly misunderstanding the question, and writing unclear responses (likely to necessitate follow-up meetings).

The third review involved evaluating artefacts, such as code files and prototype enclosures (see Appendix B.4.3 for the evaluation form used). The third review meeting was not held in person; instead the researcher was emailed artefacts or given remote access to project artefacts. The evaluation criteria chosen for this evaluation were based on a combination of techniques; these comprised: metrics that could be implemented quickly to rate code [Kan, 2002]; textbook- and teaching-based methods to rate schematics, circuits and other design artefacts (specifically, based on guidelines explained by Nelson et al. [1995] and Mano & Ciletti [2006]); other artefacts (e.g., reports and logs used to record procedures) were also evaluated using more general principals (specifically, methods to assess quality of

University of Cape Town

specification documents [Davis *et al.*, 1993], following the general guideline provided by Schach [2005] that states the quality of software relates to the extent to which its specifications are met). Change or additions to the default ESAOA support tools (i.e., ESAOA scripts and programs) provided in baseline workspaces were noted. Determining changes to baseline scripts/programs were done using the GNU find utility program [GNU, 2008b], and additional scripts were found by comparing file listings in the Tools directory of the baseline team workspace and the team's modified team workspace. The third code and design review did not evaluate the quality of artefacts in the *final* product; this was done after the demonstrations using the requirements check sheet (see Section 3.8.6).

3.8.2 Email archive

Email correspondence between the development teams and the researcher were archived. In Experiment 1, 42 emails were archived, and in Experiment 2 460. Figure 3.6 provides a sample.

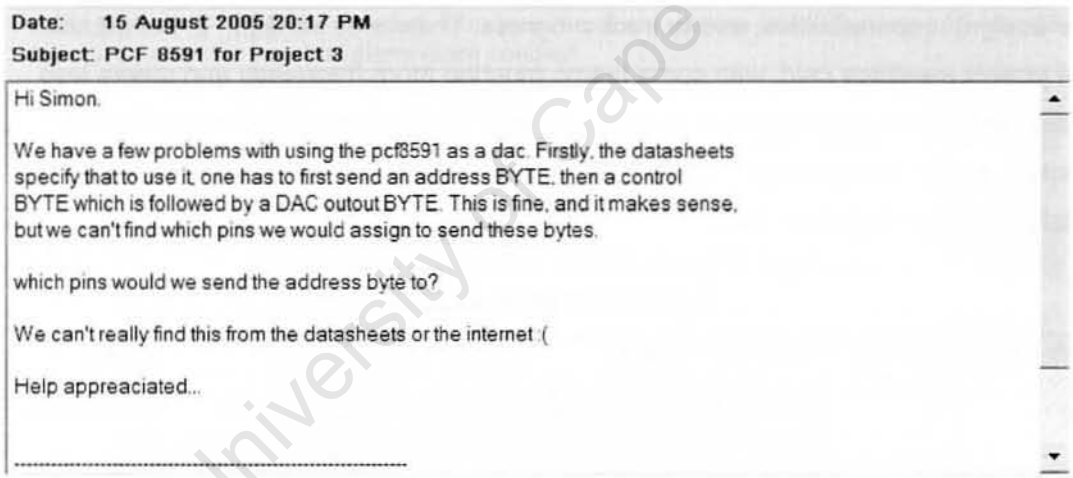


Figure 3.6: Example of email message (email #283) stored in the email archive.

3.8.3 Group forums

Three on-line forums were used in Experiment 2, namely a 'Project Discussion Forum', a 'Development Forum', and a 'General Forum'. These forums allowed engineers to post questions relating to the specific project, or to issues with product development, or to raise general concerns. These forums could be accessed by *all* the participants in the study; any one of the participants could read and respond to postings (Figure 3.7 shows a sample posting). Data obtained from these three group forums comprised 527 postings. A forum was not used in Experiment 1.

University of Cape Town

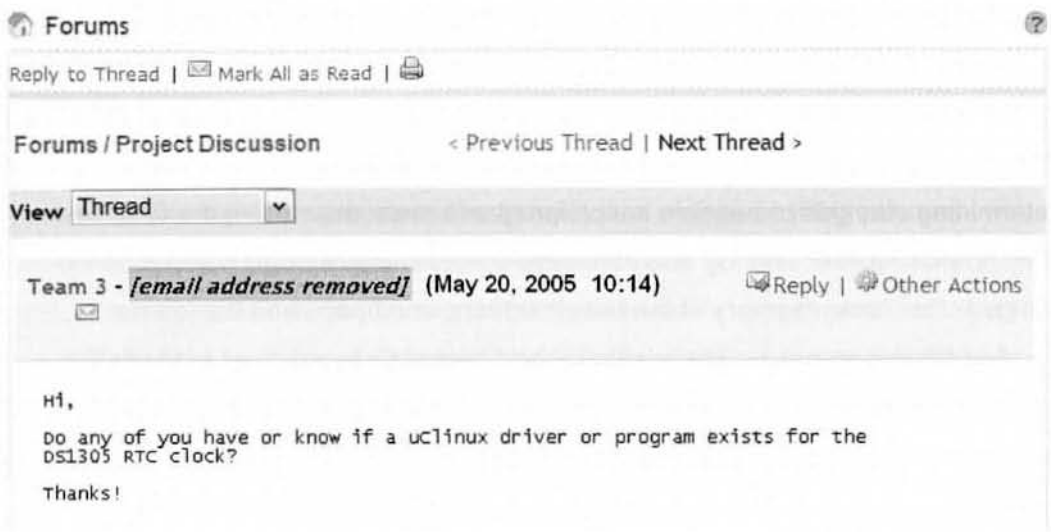


Figure 3.7: Example of a posting from the 'Project Discussion Forum'.

3.8.4 Project meetings

Each of the project teams held meetings to clarify development tasks, to assign (or re-assign) responsibilities, and to track progress. There was variation in the number of project meetings held, with some teams meeting more frequently and others less frequently. Data obtained from project meetings took the form of informal minutes, notes, and/or memoranda. Table 3.2 lists the meetings, indicating their focus and the date on which they were held.

University of Cape Town

Table 3.2: Project meetings.

Project No.	Meeting No.	Focus of meeting	Date
P1-1	1	Meeting to clarify project brief of the Software Signal Generator (SoSiG) product, and to meet team members.	10-Jun-04
	2	Initial planning meeting.	14-Jun-04
	3	Progress meeting.	11-Aug-04
P1-2	1	Meet to clarify project brief for the Antenna Controller (ANTCON) product, to meet team members and to discuss their roles.	7-Jun-04
	2	Planning meeting and division of development tasks.	14-Jun-04
	3	Progress report meeting. Refine concept design, and focus for development tasks. Initial code design review.	11-Aug-04
	4	Meeting with researcher to discuss progress, focus.	13-Sep-04
	5	Final code and design review. Focused largely on Linux driver for DAC device.	30-Sep-04
P2-1	1	Introduce team and assign roles. Discuss plan for the Location-aware Tourist Information System.	14-Mar-05
	2	Meeting to clarify concept of location-aware tourist information system and division of work.	23-Mar-05
	3	First progress report. Review breakdown and task allocation. Discuss problem of GPS module interfacing.	19-Jul-05
	4	Second progress report.	29-Sep-05
P2-2	1	Meet members; discuss GPS bus stop navigator idea.	17-Mar-05

	2	First status review meeting, plans for prototype.	20-Jul-05
	3	Second status review meeting.	27-Sep-05
P2-3	1	Clarify project brief for VibyNet concept.	14-Mar-05
	2	First review meeting.	19-Jul-05
	3	Second review meeting.	30-Sep-05
P2-4	1	Assign roles to team members and introduce the VoIP answering machine concept.	15-Mar-05
	2	First review meeting.	21-Jul-05
	3	Second review meeting.	30-Sep-05
P2-5	1	Introduce team, assign roles discuss overview of Home Automation System concept.	17-Mar-05
P2-6	1	Introduce members (3rd member absent), discuss plan for prototyping the Automation Headlights Dimmer concept. Decide on team leader and responsibilities.	10-Mar-05
	2	Meet with 3rd member to discuss project and his involvement.	15-Mar-05
	3	A second review meeting was not performed as a team member was ill.	N/A
P2-7	1	Introduce team and the Magnetic Field Sensor System.	14-Mar-05
	2	First review meeting, refine plan for prototype.	20-Jul-05
	3	Second review meeting.	29-Sep-05
P2-8	1	Introduction of team members, allocate roles, overview of Cordless Stereo concept.	17-Mar-05
	2	First review meeting.	19-Jul-05
	3	Second review meeting.	27-Sep-05
P2-9	1	Initial meeting, introduce members, assign roles, discuss central alarm clock concept.	15-Mar-05
	2	First review meeting.	18-Jul-05
	3	Second review meeting.	27-Sep-05
P2-10	1	Meet group, clarify the Voice Activation System (VAS) concept.	17-Mar-05
	2	First review meeting.	21-Jul-05
	3	Second review meeting.	27-Sep-05
P2-11	1	Initial meeting. Introduce team members and discuss the Supermarket Query Device concept.	18-Mar-05
	2	First review meeting.	18-Jul-05
	3	Second review meeting.	03-Oct-05
P2-12	1	Initial meeting to meet members and discuss brief of the Campus Protection Device (CPD) concept.	15-Mar-05
	2	First review meeting.	22-Jul-05
	3	Second review meeting.	30-Sep-05
P2-13	1	Initial meeting to introduce members and discuss overview of the Vehicle Usage Tracker (VUT) concept.	18-Mar-05
	2	First review meeting.	21-Jul-05
	3	Second review meeting.	26-Sep-05

3.8.5 Developer logs

All engineers kept logs of the major problems that they encountered, as well as *trial solutions* (solutions completed but later found to be ineffective or inappropriate and then replaced) and final solutions (solutions that were not later replaced) – or, in some case, *dead-end solutions* (i.e. possible solution that were started on but not completed and later abandoned).

University of Cape Town

There were 15 primary logs (one log per project), with some project teams keeping multiple logs (logs kept by individual developers that were later either submitted in addition to the primary log or added to the primary log before it was submitted). Some developers kept digital logs in the form of spreadsheets or text documents (as illustrated by Figure 3.8), but most teams used log books or A4 paper to record hand-written records.

PID	Problem	Solution / Comments	Time (h)
1	What components should be used for interface board?	Searched internet, requested prices from supplier	2.00
2	How should the files be managed?	Keep work files on local workstation, use samba to copy back/forth from server	0.10
3	Problem mounting samba	Recompiled kernel on developer PC	2.00
4	How to connect up CSB337?	Connected RS232 to port 0, needed 9 pin Female D connector. Eth crossover cable.	0.20
5	How to communicate with CSB337 over RS232?	Need a terminal program for Linux. Had short look, but no luck. Try again later.	0.50
	...		
15	Power supply problems	Power supply seems faulty. Testing it.	1.40
16	Replace power supply	Replaced power supply, some time to find replacement.	0.50
17	Compiling Snapgear	Found snapgear linux. Seems better. Decided to toss emdebian. Attempting to compile snapgear. Gives errors.	5.00
	...		
31	Gave up on Snapgear	Searched Snapgear fixes; decided to give up on it after not finding anything useful.	1.00
32	DC motor interface	Researching DC motor drive	1.00
33	AC-DC circuit	Constructed AC-DC circuit prototype on breadboard.	2.10
34	AC-DC circuit	Problem with AC-DC circuit; rebuilt it	3.00

University of Tasmania (UTAS)

Figure 3.8: Example of developer log.

3.8.6 Product demonstrations and project evaluations

Each team demonstrated their product on completion. Table 3.3 provides the schedule of demonstrations for Experiment 1, and Table 3.4 below is the schedule for Experiment 2. Each demonstration took between 30 to 60 minutes. All team members participated in the demonstration. The demonstrations were given to the *review panel*, which comprised the researcher and the second reviewer (an electrical engineering Masters student with embedded systems experience). Teams did not watch one another's demonstrations.

In addition to a basic introduction, each development team was expected to explain their methods, tools and components that they used. For Experiment 2, each team was permitted to use their ESAOA KMS during the demonstration and when answering questions. Figure 3.9 is a photograph of a typical demonstration set-up.

Table 3.3: Schedule of Experiment 1 demonstrations.

12 November '04	Project	14 October '04	Project
15h00	P1-1	15h00	P1-2

Table 3.4: Schedule of Experiment 2 demonstrations.

13 October '05	Project	14 October '05	Project
09h00	P2-4	09h00	
10h00	P2-13	10h00	P2-6
11h00	P2-8	11h00	P2-5
12h00	P2-3	12h00	P2-7
14h00	P2-9	14h00	P2-10
15h00	P2-12	15h00	P2-11
16h00		16h00	P2-1

The demonstration data comprised a *demonstration check sheet* (Figure 3.10), a *requirements check sheet* (Figures 3.11 and 3.12). Both types of check sheets included categories that were awarded numerical ratings between zero and a maximum value (indicated on the sheet). Space for additional comments was provided. All criteria of the demonstration check sheet were given rating values between 0 and 10. The ratings were chosen to develop a quantitative assessment of the final prototype and how well the team was able to demonstrate their prototype, and, in the case of the requirements check sheet in particular, a numerical value indicating the quality, organisation and access of artefacts. These values were also used as a means to compare the resultant products and artefacts between projects.

University of Cape Town

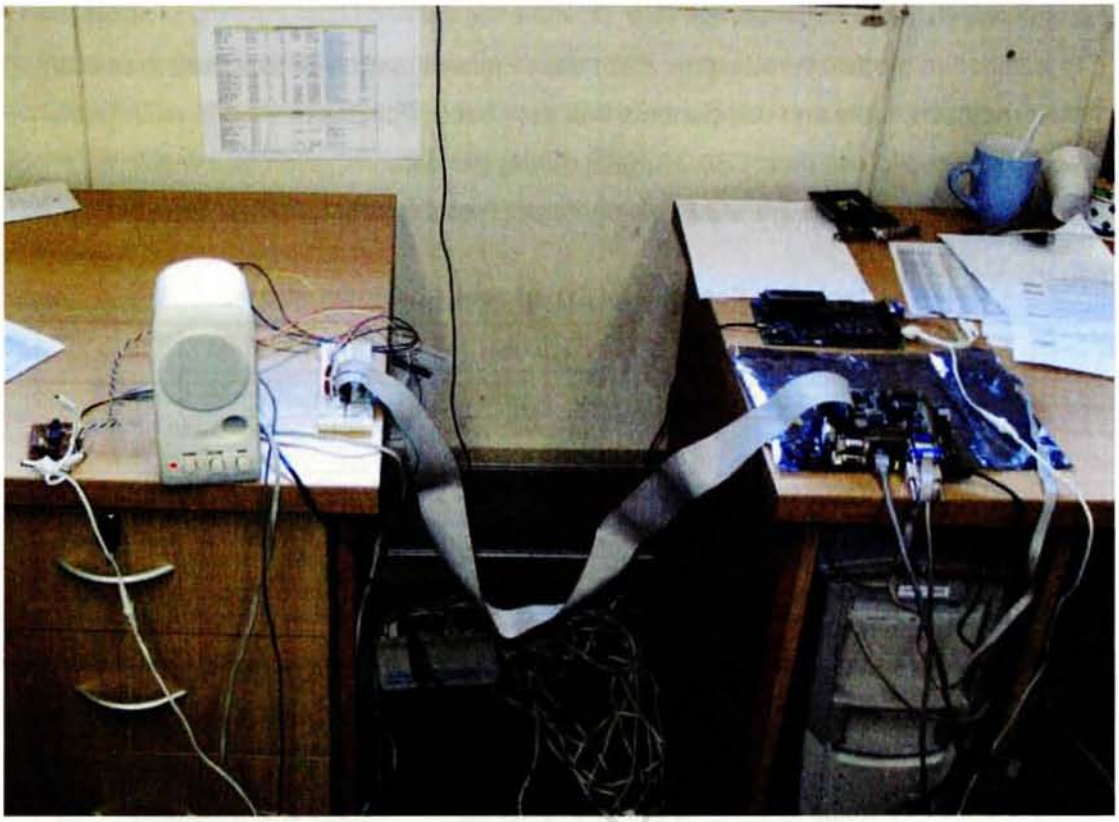


Figure 3.9: Example of ES prototype set up for a demonstration.

The demonstration check sheet was completed in rough by researcher during the product demonstrations, and then the researcher and second reviewer discussed and agreed upon the final ratings in private after each demonstration. The agreed upon ratings for each team was then entered into a final digital version of the demonstration check sheet. Figure 3.10 shows the final demonstration check sheet for Project P2-6.

The demonstration check sheet comprised sixteen criteria, each criterion rated from 0 (for entirely lacking), 1 to 4 (to indicate a level of poor performance), 5 in indicate mediocre performance, 6 to 9 (for good performance) and 10 (to indicate excellence). As shown in Figure 3.10, descriptions are included or each criteria and ratings.

The requirements check sheet was completed by the review panel after the demonstration. A similar procedure as to the one used for the demonstration check sheet was used in deciding ratings of the requirements check sheet; to be specific, the researcher completed a draft version of the check sheet and the ratings were then discussed with the second reviewer, and if necessary modified, to reach agreed upon ratings for each criterion.

The requirements check sheet provides a broader set of rating criteria. A different rating strategy was also implemented for this check sheet; criteria that were more important were given higher maximum rating values. The maximum rating value was approximately based on the amount of effort involved in providing a solution for that requirement. For example, in Figure 3.11, the schematic was given a maximum rating of 30, whereas the responsiveness of the system was given a maximum rating of 3. The requirements check sheet was used to rate the construction of the product, the quality of the product, its operation (Figure 3.11), and the quality and organisation of its design artefacts (Figure 3.12).

The structure of the requirements check sheet, and the evaluation of the requirements, were based on theories of embedded and real-time systems by Liu [2000] and Douglass [2000], and on evaluation methods described by Briggs [1992]. The check sheet had to maintain a level of generality, as the same criteria were to be used with a variety of different products, not all of which had the same set of requirements. Consequently, the specific items in the requirement check sheet include issues that are generally considered important concerns for embedded system products (as identified by Liu [2000] and Douglass [2000]), namely, robustness, predictability and timeliness (which have accordingly been included in the requirements check sheet). The weighting for each set of criteria (e.g., 44% for functional requirements) are approximately based on the relative effort involved in satisfying the corresponding aspects of the product (these estimations are broadly based on work by Blackburn *et al.* [1996] and Ko *et al.* [2007a]).

University of Cape Town

DEMONSTRATIONS CHECK SHEET

Team Number: 6

For each criterion below, please tick one of the boxes to the right that most closely corresponds to your impression of the team's performance. Note that boxes to the left indicate poorer performance, boxes to the right indicate better performance.

	CRITERIA	POOR				REASONABLE					EXCELLENT
	Rating value	1	2	3	4	5	6	7	8	9	10
Preparedness	**Readiness** *Tick a box:*										✓
	(Did you think the team was suitably prepared for this demonstration?)	Team is poorly prepared. Had to wait for them to configure things at the start of the demo.				better > < worse	Team very well prepared. Prototype all set-up at start. They have to wait for them to fixing things at the beggining.				
	Order *Tick a box:*										✓
	(Did the demo follow a good sequence?)	Poorly sequenced. Difficult to follow and understand logic of the steps of the demo.				better > < worse	Well sequenced. Follows a logical sequence of steps (i.e., builds from objective towards experiment and				
	Flow *Tick a box:*									✓	
	(How was the transition between parts of the demo?)	Poor flow. Stops and starts. Presenters did not hand over well from one to the other. Wasted time fixing things.				better > < worse	Good flow. Presentation progressed smoothly. Presenters hand over from one to another elegantly. Didn't have to spend time fixing things.				
	Time keeping *Tick a box:*										✓
	(How good was the team at time keeping?)	Poor time keeping. Spent way too long on unimportant issues and far too little time on the main parts.				better > < worse	Good time keeping. The team balanced their time well, focusing most of the presentaion on showing and explaining results.				
Project Summary	**Clarity** *Tick a box:*								✓		
	(How clear were the speakers?)	Incomprehensible / illogical. Speakers spoke too softly.				better > < worse	Well delivered and understandable. Speakers spoke clearly.				
	Explanation *Tick a box.*								✓		
	(How good were explanations of the project concept?)	Explanations were hard to follow / illogical				better > < worse	All explanations were very clear and understandable.				
	Description *Tick a box:*										✓
	(How good were descriptions of what was to be tested in the demonstration?)	Descriptions were incomprehensible. Eg. Did not describe the componets used. Did not point things out in the setup.				better > < worse	All descriptions and explanations were well delivered and understandable.				
Setup	**Explanation** *Tick a box:*										✓
	(How good were explanation of how the prototype was setup/configured in terms of things to test)	Explanations were hard to follow / illogical. Did not understand how things had been configured for the experiment.				better > < worse	All explanations were very clear and understandable. Easy to follow how the setup has been configured for the particular experiment to be done.				
	Description *Tick a box:*								✓		
	(How good were descriptions of the various components)	Descriptions were incomprehensible. Eg. Did not describe or point out componets to be used in the setup				better > < worse	Descriptions were well delivered, items were poited out.				
Experimentation	**Method** *Tick a box:*								✓		
	(How well was experimental procedure described?)	Poorly described procedure.				better > < worse	Well presented procedure. Use of diagrams (e.g. flow diagram) to help explanation.				
	Experiment results *Tick a box:*										✓
	(How well were experiment results shown?)	Results were largely invisible or difficult to					Results very clearly shown and easy to interpret.				
	Analysis/interpretation of results *Tick a box:*										✓
	(Additional procedures or discussion concerning the results)	Results poorly analysed, or no analysis or conclusions given.				better > < worse	Procedure for analysis the results clear (note: this may have already been given 'setup explanation'). Clear description of what the results mean.				
Code	**Explaination of code** *Tick a box:*									✓	
	(Team is asked to discuss an important signal processing and/or driver routine)	Team doesn't seem to understand their own code. Code is very untidy.				better > < worse	Team understand their own code well and can explain what it does. Code appears tidy and commented.				
General	**Recommendations** *Tick a box.*								✓		
	(Team is asked to discuss improvements or future work ideas)	Team has no ideas for improvement or future work.				better > < worse	Team has good ideas for possible refinements and upgrades, as well as future work suggestions.				
	Questions *Tick a box:*										✓
	(How to the members handle questions)	Unable to answer questions or only incomprehensible answers given.				better > < worse	Teams answered questions well. All team members participated in answering questions.				
	Language *Tick a box:*										✓
	(Did the team members use language appropriately?)	Unprofessional, too informal.				better > < worse	Team members used professional language at a suitable level of formality.				
	TALLY:	0	0	0	0	0	0	0	5	2	9

COMMENTS (please write any comments you may have to add in the space below):

TOTAL SCORE: 148 /160

Percentage: 93%

University of Cape Town

Figure 3.10: Demonstration check sheet for project P2-6.

PROJECT REQUIREMENTS CHECKSHEET

Issue	Max	Section Total	Category Weight	P1	P2	...	P13
1. FUNCTIONAL REQUIREMENTS *Note: Rating of functional requirements for a particular product are based on the specific use-cases provided in the product's requirements*			**44%**		...		
Hardware		**50**		**48**	**33**		**50**
Prototype assembly - robustness & elegance - for endosure, see below	20			18	18		20
Schematics and/or circuit diagrams	30			30	15		30
Software / Program Execution		**70**		**60**	**59**		**29**
Penalties for execution errors: - Program runs smoothly: no penalties - Program hangs after a while or handles invalid/valid input poorly or terminates unexpectedly: up to 20 points removed - Program does not run: *see penalty notice below*				0	0		0
Start-up message displayed	2			2	2		2
Argument parsing works	6			6	6		6
Menu works	14			6	6		0
Menu ease of use (6=user friendly; 0=very user unfriendly)	6			6	6		0
Communications	21			19	21		21
Event & service processing	21			21	18		0
2. TEMPORAL REQUIREMENTS Quality of Service (QoS) (real-time performance issues)			**11%**				
Total quality of service points		**30**		**29**	**19**		**26**
Predictability future response of system is predictable from past events	6			6	6		3
Speed product is responsive	2			2	0		2
Timeliness the time between occurrence of an event and handling of the event is sufficiently small	4			4	3		3
Throughput max no. of jobs/time unit	3			3	2		3
Responsiveness worst-case latency not exceeded	3			3	2		3
Capacity ability of system to meet all hard deadlines; no. synchronous jobs / queue size sufficient	4			3	0		4
Reliability / sustainability system keeps important deadlines even in overload conditions	4			4	2		4
Safety where applicable, account for safety issues	2			2	2		2
Security extent to which product is protected against misconduct	2			2	2		2

Figure 3.11: First page of the requirements check sheet.

University of Cape Town

Issue	Max	Section Total	Category Weight	P1	P2	...	P13
3. QUALITY OF ARTEFACTS			**40%**				
ESAOA Repository		**20**		**20**	**19**		**16**
All files within project root.	3			3	3		3
Artifact Organization Diagram (AOD)	14			14	14		10
Location of files	3			3	2		3
Code		**80**		**76**	**74**		**53**
Conformity to ANSI C	5			5	5		5
Use of comments	10			6	9		3
Module interfaces	20			20	18		18
Required modules							
- Main module	3			3	3		3
- Encoding of welcome / version information	4			4	4		4
- Configuration settings and command parsing	6			6	6		6
- Menu component	8			8	8		2
- Service module(s)	12			12	10		0
- Drivers	12			12	11		12
Compiling and Linking		**10**		**10**	**10**		**10**
- No warnings or errors: all 10 points - Warnings: 5 points (if many warnings) - Minor error: 0 points; Major error: penalties	10			10	10		10
4. QUALITY OF ENCLOSURE			**5%**				
Enclosure		**15**		**5**	**12**		**15**
A rough enclosure is provided with labels indicating important controls and displays	15			5	12		15
TOTAL POINTS FOR PROJECT		**275**	**100%**	248	226		199
PERCENTAGES				90%	82%	...	72%

Guidelines for requirements: refer to these books
- Douglass, B. *Real-Time UML*, Reading: Addison Wesley Longman, 2000, pp 55-57
- Liu, J. *Real-Time Systems*, Upper Saddle River: Prentice Hall, 2000.

Figure 3.12: Second page of the requirements check sheet.

3.8.7 End-of-project survey

At the end of Experiment 2, each participant completed a two-page questionnaire regarding the ESAOA KMS they had used (Experiment 1 did not have a survey because its participants used their own ad hoc methods). The questionnaire included specifics about how individual members had used aspects of the ESAOA KMS framework, their views on its general usefulness, the difficulties they encountered in using it, and suggestions for further improvements. Figures 3.13 and 3.14 respectively show the first and second pages of a completed questionnaire.

University of Cape Town

13 October 2005 Embedded System Artifact Organization and Adaptation Framework **ESAOA**

SURVEY

Circle a score between 1 (for bad) and 5 (for good) to indicate your rating for each criteria.
Use 0 to indicate you cannot answer the question because you did not encounter that feature.

#	*Description*	*Rating*
0	**Comments on project**	
0.1	How did you find the project? ... 1=bad, 5=great	0 1 2 3 4 [5]
0.2	Comment on the concept, or make recommendations here: The work was interesting.	
1	**General**	
1.1	Do you think ESAOA is suitable for educational purposes? (1=no, 5=yes)	0 1 2 3 [4] 5
1.2	Would you consider using ESAOA for professional work? (1=no, 5=yes)	0 1 2 3 [4] 5
1.3	Were the Pracs at a suitable level? 1=too hard, 3=just right, 5=easy	0 1 [2] 3 4 5
1.4	Were the Projects at a suitable level? 1=too hard, 3=just right, 5=easy	0 1 2 [3] 4 5
1.5	Comment on the main difficulty you experience with using ESAOA: When it was working, it worked fine. But there were cryptic error messages, and I didn't know what to do when it breaks, especially for the mm tool. More information or a manual about how the makefile is generated would probably help with the errors.	
1.6	Comment on issues you think ESAOA helped with: ESAOA was very useful in the compiling, and intelligently combining the code modules and other parts of the programs together. This saved time for the project work.	

Please turn over (continue on next page if submitting electronically) ...

University of Cape Town

Figure 3.13: First page of questionnaire completed by a team member.

#	*Description*	*Rating*
2	**ESAOA Directory Structure**	
2.1	Did the **go** utility help with navigation? (0=didn't use it, 1 = useless, 5 = useful)	0 1 2 3 4 [5]
2.2	Ease of navigating directory structure (1=difficult, 5=easy) e.g. go SPDM # goes to PDM directory	0 1 2 3 [4] 5
2.3	How logical did you find the directory structure (0 = illogical, 5 = logical)	0 1 2 3 [4] 5
2.4	Directory lists operated differently in different directories (i.e. contexts). It is *unlikely* you found this format user-friendly initially, but with experience is this format beneficial? I used the standard ls (1 = remained awkward, 5 = became quite useful) Example: 1 MAP File: BitBanger.cm.map 1 object file --- | 1 C Files | --- | BitBanger.c | ---	0 1 [2] 3 4 5
3	**ESAOA Entry and Exit**	
3.1	The "source enter-esaoa" method was used to setup the environment for ESAOA, without having to make changes to .profile. Did you find this strategy to be effective?	0 1 2 3 [4] 5
3.2	The command prompt was configured to show the subdirectory from the project root, i.e. '*Project*:SH/*subdirs*/>'. (1 = was confusing, 5 = clear) Initially it was confusing, but I got used to it.	0 1 [2] 3 4 5
3.3	Use of esaoa.rc, i.e. actions to perform at start of working on a project. Such as: displaying project and developers' names. (1 = waste of time, 5 = useful) I did not know how to use this. We did make some of our own scripts.	0 1 [2] 3 4 5
3.4	Exiting project (PROJ_EXIT macro). Although you probably didn't notice, ESAOA implements "exit" operations when the framework environment is exited, e.g. to backup all C files. Would you consider using exit macros? (1 = definitely not, 5 = definitely)	0 [1] 2 3 4 5
4	**MakeMake (mm) utility**	
4.1	Did you find the mm utility to be useful? (1=don't like; 5=like)	0 1 2 3 4 [5]
4.2	Ease of changing platform? (0=didn't try; 1=didn't work; 5=works easily) i.e. compiling code for PCBox/Linux and porting to CSB337/MicroMonitor	0 1 2 [3] 4 5
5	**Environment Variables**	
5.1	Most of the environment variables are intentionally invisible to the user, except for a few. If you used, or had known of, these variables, would they be useful? (1 = probably wouldn't use, 5 = probably would used) *Some environment variables:* SH (project root), SAPPS (applications directory), SPDM (Platform Deployment Modules), and SDEFAULT_PDEFS (active PDM make module, e.g. SPDM/CSB337/MicroMonitor/defs.make).	0 1 2 [3] 4 5

THANK YOU

University of Cape Town

Figure 3.14: Second page of questionnaire completed by a team member.

3.8.8 Limitations of the data capture methods

All members of the development team worked in the laboratory for some of the time. The teams worked in a number of different ways: some teams worked solely in the laboratory, while others followed more decentralised [Crowston & Howison, 2005] work practices (i.e., with some developers choosing to work outside the laboratory while others worked in the laboratory). Some team members followed a pair programming (or 'extreme programming' [Beck, 1999]) approach in which one developer used a computer while his/her colleague watched and made suggestions. All teams used the laboratory as an environment to integrate and test their work. Teams generally held group meetings outside the laboratory in order to avoid disturbing other teams. Design and review meetings with the researcher were also held outside the laboratory; but demonstrations were done in the laboratory. Teams captured their own minutes of group meetings, which they presented to the researcher; the researcher himself was not present at these meetings, as it would have biased the operation of the meetings.

Oral conversations between team members were not recorded; doing so was outside the scope of this research project. Furthermore, recording conversations might have reduced the number of volunteers willing to participate in the study, and the development methods of those volunteers willing to participate might have been effected by the presence of recording devices [Speer & Hutchby, 2003]. For example, some developers may have spoken less to avoid having their voices recorded.

3.9 Data analysis

The data analysis is influenced by methods used by Cross [1994; 2004] and Cross *et al.* [1996] for researching engineering methods. The general approach of these analysis methods is as follows: first, data are captured from a variety of sources to represent a broad spectrum of observations from case studies. The data are then investigated, looking for commonly occurring practices, difficulties, shifts in design goals, and problem-solving strategies performed by engineers in the case studies. These investigations include studying features of development such as "problem spaces" and "solution spaces" [Cross, 2004, pg. 5], which involves determining how developers 'frame' design situations, and how these situations relate to future moves or adjustments to project objectives and other decisions made during project activities. Adapted versions of these analysis methods are applied to the study of ESAOA activities for this thesis; the specific methods are described in this section.

University of Cape Town

3.9.1 Overview of data analysis

The data analysis method depends on the definition of a *knowledge event*. A knowledge event is defined as an action that specifically involves knowledge – such as finding information, applying a process, testing an idea, or solving a problem.

The data analysis method involved the following steps: 1) systematising the data; 2) categorising knowledge events according to knowledge type; 3) mapping problems/solutions and identifying knowledge event chains; 4) determining whether knowledge events produced productive knowledge or non-productive knowledge; and 5) finalizing *knowledge registers* (defined in Section 3.9.2.2) for each project. Trend analysis, which utilised a custom-developed program and graphing, was done once the knowledge register was finalized. Figure 3.15 illustrates this analysis method.

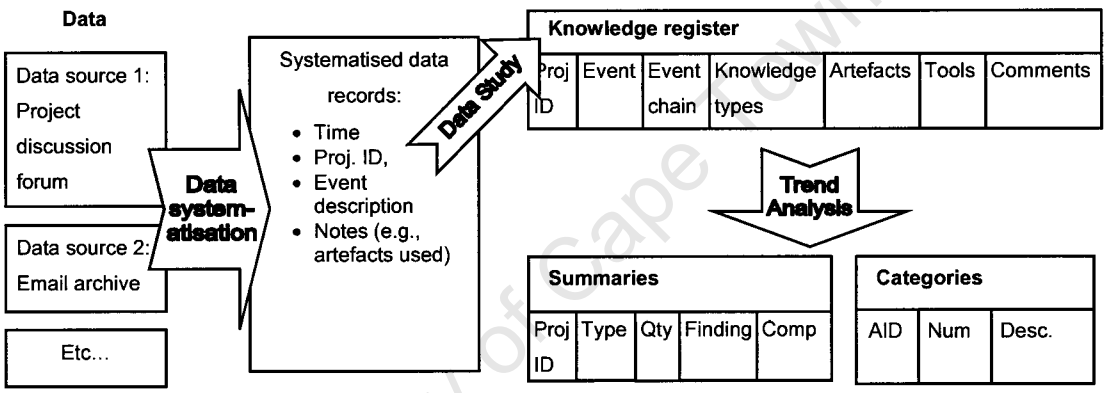


Figure 3.15: The data analysis process.

All the data sources were printed before starting the first step, and then placed in separate piles corresponding to each data source. The piles of printouts were sorted chronologically from the earliest to the most recent entry. Three passes through the printouts were performed in Experiment 1, whereas only two passes were needed in Experiment 2. Each pass through the printouts followed a breadth-first progression, moving across the piles, rather than one pile at a time, so that the data entries were studied in a chronological order. The order of the analysis steps differed between experiments, due to optimisations noted during the analysis of Experiment 1, and then applied to Experiment 2 data.

For both experiments, steps 1, 2 and 3 were done together in the first pass of the data, during which initial *knowledge registers* and initial listings of *knowledge event chains* were constructed (see definition in Section 4.2.10). For both experiments, step 3 (viz. mapping problems/solutions and listing the event chains) and step 4 (i.e.,

University of Cape Town

categorising knowledge as either productive or non-productive) were finalized by performing a second pass through the data. For both experiments, the completion of the knowledge registers, step 5, was done during a final pass through the data (this constituted a third pass for Experiment 1 and the second pass for Experiment 2). Step 6 involved creating graphs from the knowledge registers. The subsections below elaborate on these steps of the analysis process.

3.9.2 Systematising the data (step 1)

The process of systematising the data involved identifying a unit of analysis. The initial unit of analysis was determined based on the results of a preliminary study (Section 4.2) performed on the data obtained from Experiment 1. The *knowledge event* was used as the unit of analysis for Experiment 1. As mentioned earlier, a knowledge event is defined as an action that specifically involves knowledge (e.g., finding or requesting information, applying a process, or solving a problem).

3.9.2.1 Annotating printouts

During this first analysis step, all the data printouts were investigated by the researcher in order to identify the knowledge events that occurred as part of, or in response to, ESAOA activities. This was done through a process by which the researcher annotated the printouts. An *item number* was assigned to each data entry; these numbers started from 1 for each data source (i.e., email 1 referred to the first email message in email archive; log 1 referred to the first developers' log). The item number was followed by the project number. Text referring to artefacts, tools, and issues indicating a problem or solution were identified by underlining or circling the text on the printout. Lists of important artefacts or tools (e.g., artefacts referred to in multiple data records) were written down on the printouts. Figure 3.16 shows a scanned annotated printout, illustrating how printouts were annotated; this particular printout is a forum posting.

Knowledge events were identified in all data sources. These events were entered chronologically into the knowledge register, as per the chronological progressing of investigating the data across printout piles (as apposed to piles at a time).

Developer logs, code reviews, email archives, and group forums were found to be the most useful sources of data, as they were created close to the time the ESAOA activities happened, and were found to be 'knowledge-rich' [Kitamura *et al.*, 2006].

University of Cape Town

During Experiment 1, the analysis moved towards a process of identifying knowledge-producing *event chains*, which was part of problem/solution mapping process (see Section 3.9.4), in which each event chain was a sequence of linked knowledge events leading from a problem to a solution or dead-end.

For Experiment 2, the unit of analysis was changed to *knowledge event chains* (a sequence of related knowledge events) rather than discrete knowledge events. Each knowledge event chain in Experiment 2 was classified depending on the predominant form of knowledge produced during its composite knowledge events. A set of counters were associated to each event chain to keep track of the number of knowledge events for each type of knowledge produced (i.e., a counter for the number of data knowledge events, a counter for the number of process knowledge events and a third counter for the innovation knowledge events). The counter values were used to determine the predominant type of knowledge produced in the event chain, and also for producing knowledge occurrence graphs (see Section 3.9.7.2).

The decision to use event chains as the unit of analysis for Experiment 2 was made chiefly because it was found that the large accumulation of discrete knowledge events was both difficult and time-consuming to work with during the analysis phase; for instance, it gave rise to problems such as needing to refer back to preceding events, and to look forward to succeeding events, in order to determine the issues that were relevant for an individual event, such as what problem it was related to, and whether that problem resulted in a successful solution, or was eventually abandoned. Changing to a unit of knowledge event chains expedited the analysis of Experiment 2, by dealing with the larger 'chunks' of knowledge event chains, instead of smaller 'bits' of knowledge events. The analysis of Experiment 1 was repeated using event chains to establish a basis of comparison between experiments (see Section 4.5.3).

University of Cape Town

3.9.2.2 Building knowledge registers

In order to facilitate the study of the experimental data, a *knowledge register* was developed for each project to capture and systematise all the data related to knowledge events. The knowledge register was implemented as a spreadsheet comprising the following columns to maintain information for each knowledge event:

- The number of the event (starting from 1);
- A set of keywords characterising the event;
- Predecessor events (a list of zero or more event numbers);

- A more detailed description of the event, describing the issue/problem encountered or a solution tried (including mention of artefacts used);
- The data source in which the event was found; and
- Time spent (in hours) performing the event.

The event numbers were assigned chronologically, starting at 1. An artificial starting event was added as the first entry in the knowledge register and assigned event number 0. In subsequent steps of the analysis process, additional columns were added to the knowledge registers to record other results, such as knowledge classifications. Table 3.5 shows an excerpt from the initial knowledge register from Project P1-2. The columns are as follows: event number; keywords characterising the event; predecessors; comments and description of the event; the data source; and the time it took to complete the event. The entries in the 'Source' column are abbreviated as 'L' for developer log, and 'M' for meetings.

Table 3.5: Except of initial knowledge register from Project P1-2.

Event No.	Event description	Pred.	Event comments / solution description	Source	Time (h)
0	<Root>		This is the starting point of development		
1	What components should be used for interface board?	0	Searched internet, requested prices from supplier	L	2
2	How should the files be managed?	0	Supervisor says to keep work files on local workstation, use samba to copy to server	M	0.1
3	Problem mounting samba	2	Had to recompile kernel on developer PC	L	2
4	How to connect up CSB337	0	Had to figure out how to connect up CSB337	L	0.2

University of Cape Town

3.9.3 Categorising knowledge events by knowledge type (step 2)

The preliminary study (see Section 4.2.7) revealed that knowledge acquired and applied by the ES development teams could be divided into categories of: 1) *data knowledge* (knowledge of information sources), 2) *process knowledge* (how to carry out development methods), and 3) *innovation knowledge* (knowing which design ideas work). These knowledge categories became a focus of the data analysis.

During the first pass of the data, each knowledge event was categorised according to which of these knowledge categories were the most predominant type of knowledge produced during the knowledge event. The classifications were recorded by

annotating the data printouts corresponding to knowledge events; events that produced mainly *data knowledge* were marked 'DK', events that produced *process knowledge* were marked 'PK', and those that procedure *innovation knowledge* were marked 'IK'. Figure 3.16 demonstrates how printouts were annotated with information about knowledge categories. An additional column (titled 'K' for 'knowledge type') was added to knowledge register to record these classifications.

3.9.4 Mapping problems and solutions (step 3)

During the first pass through the data, development problems and attempted solutions to these problems were identified. This approach involved determining the knowledge events related to the same 'problem-solution cycles' or their sub-problems. The concept of problem-solution cycles was identified as part of the preliminary study (see Section 4.2.3). Each problem-solution cycle was given a number and a list of the various problems and their solution attempts was maintained. This step involved reading through the data to determine problems that the developers attempted to solve, and then searching the various data sources to determine how each problem was solved (if at all). These sequences of knowledge events are referred to as *event chains.* Each event chain related to a *non-trivial solution cycle* (see Section 4.2.4) and aggregated a collection of knowledge events.

An optimisation was applied in this step during the analysis of Experiment 2 data, in which each event chain was associated with a set of *knowledge occurrences.* Each set of *knowledge occurrences* were essentially counters that kept track of the number of data, process and innovation knowledge-producing events that occurred in the event chain. This optimisation avoided the need to traverse the knowledge register in order to calculate the number of knowledge events in an event chain, an undertaking that would otherwise be necessary during step 6 (the trend analysis step).

3.9.5 Categorising productive vs. non-productive knowledge (step 4)

Following the preliminary study (Section 4.2.9), the concepts of *productive time* and *non-productive time* were identified. Productive time was time spent on learning tasks in which knowledge was acquired and used in developing the final product; in contrast, non-productive time was time spent on acquiring knowledge that was not used in developing the final product[4].

[4] Note this does not mean that learning knowledge that is not used in for that particular project is a waste of time; in the bigger scheme of things, knowledge and skills obtained while learning non-productive knowledge may benefit the developer in other ways.

University of Cape Town

In the case of both experiments, each knowledge entry in the data was investigated to determine whether or not it was used in the design of the final product. Event chains leading towards a dead-end were also determined. The 'P' and 'NP' columns were added to the knowledge register to maintain this information; the 'P' column demarking acquisition of productive knowledge, and the 'NP' column indicating acquisition of non-productive knowledge. The corresponding data entries in the printouts were annotated either as productive (labelled 'pk' in a box) or as non-productive (labelled 'npk' in a box). Figure 3.16 provides an illustration of how these annotations were done.

For Experiment 1, the amount of time spent in knowledge events was recorded, or estimates calculated based on interviews with developers. Based on the results of the previous procedure of classifying knowledge events according to productive and non-productive, and from investigation of the data, the time record of each knowledge event in the knowledge register was separated into two parts: an amount of productive time, and an amount of non-productive time. Printouts corresponding to each knowledge event were accordingly annotated by the amount of productive and non-productive time. For example, a note such as '2h pk' was added to a printout if the corresponding knowledge event involved two hours of productive time, whereas a note such as '2h npk' would indicate two hours of non-productive time. Thus, the total time spent obtaining productive knowledge versus the total time spend obtaining non-productive knowledge could be determined for Experiment 1.

A concern identified during Experiment 1 was the difficulty of capturing and maintaining an accurate record of the time spent on individual knowledge events. Recording the amount of time spent on these activities depended on the developers accurately logging their activities, as well as deducing the amount of time spent on productive or non-productive knowledge generation (e.g., during meetings and interviews). The data capture method for Experiment 2 was therefore simplified, which restricted the amount of detail required in respect of time spent on knowledge events. Step 4 of the data analysis method was changed so that a set of *knowledge occurrences* were maintained for each event chain (as described in Section 3.9.4). After this, each *knowledge occurrence* counter was separated into two counters, one to keep track of the number of *productive events*, and the other to track the number of *non-productive events*. A *productive event* (corresponding to productive time) is a knowledge event within an event chain that led to a final solution used in the final

University of Cape Town

version of the product (i.e., did not result in a dead-end). Whereas, a *non-productive event* in a knowledge event *not* used in developing the final product (i.e., or resulted in a dead-end). Thus, the revised analysis was based on the number of knowledge events per knowledge category, whereas the previous version was based on time used in producing knowledge per category. This analysis method was also applied to Experiment 1 to establish a basis for comparison between experiments (Section 4.5.3 presents the results for Experiment 1 using the revised method).

Figure 3.16 provides a scanned image of an annotated forum message; dashed lines and typed labels have been added on top of the scanned image to explain the hand-written annotations (the project number was determined based on the 'From' field of the posting; names of the participants have been removed).

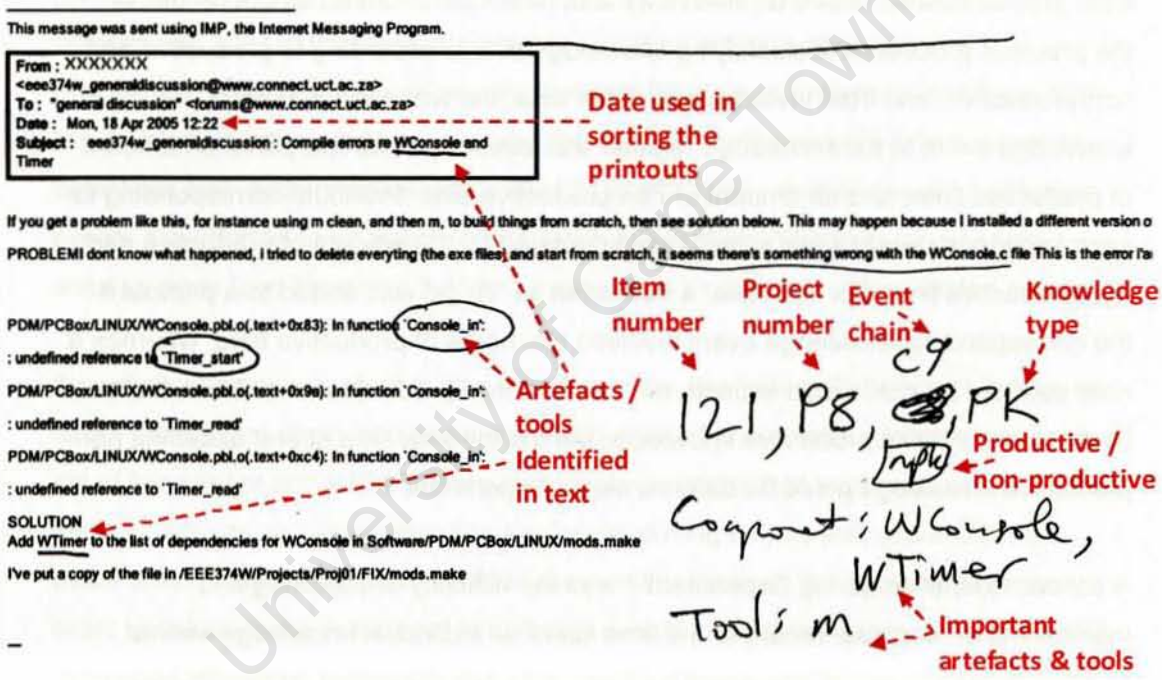


Figure 3.16: Sample annotated forum posting taken from Experiment 2.

3.9.6 Finalizing the knowledge register (step 5)

The knowledge register for each project of both experiments was finalized by checking that each knowledge event identified in the data still had a corresponding entry in the knowledge register for the project concerned. The knowledge register was also checked to ensure that no predecessor links erroneously pointed to the wrong event number (this was a necessary to ensure that the PTHC program, used in trend analysis, would terminate).

University of Cape Town

Table 3.6: Except of the knowledge register for Project P2-1.

No.	Data Source *	Project number	Event chain	Phase	Data knowledge	Productive knowledge	Role	Logistic	Innovation knowledge	Productive knowledge	Non-productive knowledge	Artefacts: Tools	Artefacts: components	Researcher's Comments
1	E	1	1	1		1	1			1		Role responsibilities		
2	E	1	1	1		1	1			1				
3	E	1	2	1		1	1			1			EM303	
4	E	1	2	1		1	1			1				
5	E	1	2	5	1					1				
6	E	1	2	3		1	1				1			upset about group member changing the roles
7	E	1	3	1	1					1				Re: What to access
8	E	1	4	1		1		1		1				
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

University of Cape Town

Legend to Data Sources: E: Email, D: Developers' log, M: Meeting, F: Forum

3.9.7 Analysing trends

Once the knowledge register had been completed, it was used for further trend analysis, which included producing graphs, tabular summaries, comparisons and correlations using the knowledge register. Data synthesis followed data analysis by modelling and generalising the results, which were then applied in the framework construction phase. Data synthesis is explained in Section 3.10.

Graphs and summary tables were used to identify trends within particular projects, and across the projects. For Experiment 1, *productivity graphs* (see Section 4.4) were produced, which plotted productive time and non-productive time (on the vertical axis) against knowledge events (ordered chronologically on the horizontal axis); separate graphs were used to illustrate trends for the data, process on innovation knowledge categories. The summary tables for Experiment 1 give totals for the hours of productive and non-productive time spent on knowledge events, separated between the data, process and innovation knowledge categories.

The data analysis methods were changed after analysis of Experiment 1 data, which lead to modification of graphing approaches used for Experiment 2. For Experiment 2, knowledge occurrence graphs were produced, which plotted productive and non-productive knowledge occurrences against knowledge events ordered chronologically. The summary tables for Experiment 2 provide the total number of non-productive and productive knowledge occurrences, separated into the data, process and innovation knowledge categories.

The sections below elaborate on how the graphs and summary tables were created.

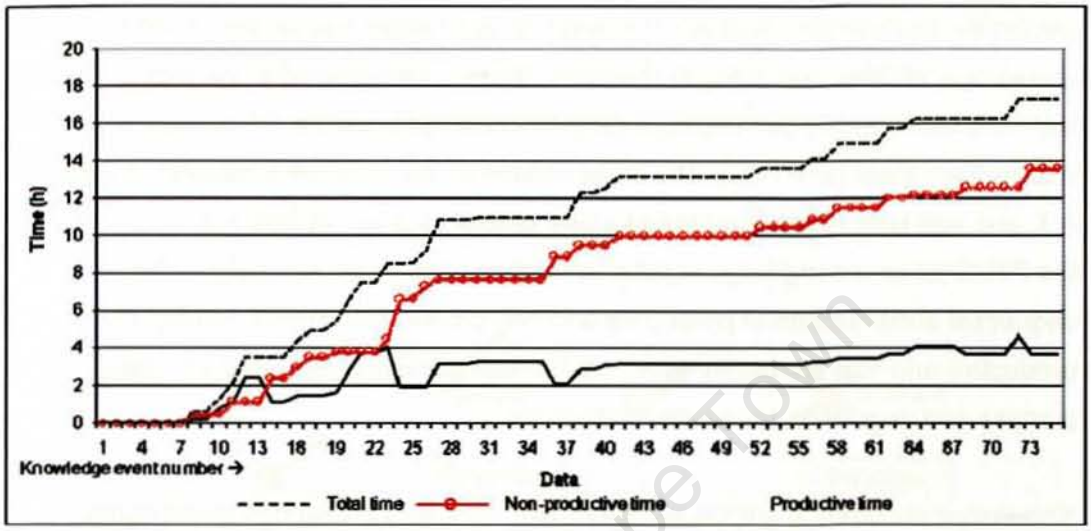
The method for comparing results across Experiments 1 and 2 involved repeating the data analysis phase of Experiment 1 using the methods applied in Experiment 2 (see Section 3.9.7.4 for the details).

3.9.7.1 Productivity graphs

Productivity graphs were prepared for the Experiment 1 projects. This task involved using each project's knowledge register to plot the accumulation of productive and non-productive time (in hours) spent acquiring knowledge, against the individual knowledge events ordered chronologically. The accumulation of time is shown on the vertical axis; the knowledge events are shown on the horizontal axis. Separate

University of Cape Town

graphs were used to show results for the different categories of ESAOA knowledge (namely the categories of data, process and innovation knowledge identified in Chapter 4). In addition, a fourth graph was provided for each project that plotted the total accumulation of productive and non-productive time against knowledge events. Graph 3.1 shows a productivity graph for data knowledge.



Graph 3.1 Productivity graph for data knowledge in Project P1-1.

This task was facilitated by developing a program, referred to as the *Partitioned Time History Calculator* (or PTHC). The initial version of the program is described in Section 4.4, and was later redesigned to use event chains as a unit of analysis. In summary, the PTHC takes a knowledge register spreadsheet as input, and outputs the sequential accumulation of productive and non-productive time (or number of productive and non-productive events in the later version) separated into the data, process and innovation knowledge categories.

Knowledge occurrence graphs in Experiment 1 showed knowledge event number versus time, whereas those of Experiment 2 displayed knowledge chain number versus number of event occurrences (which was an approximate indication of time, based on aggregated knowledge events – see definition in Section 4.2.10). These graphs were used to compare results between projects in the same experiment.

The method for comparing results across the experiments is explained in Section 3.9.7.4. Graphs 3.1 – 3.4 are examples of knowledge occurrence graphs, taken from the results for Project P2-1.

University of Cape Town

3.9.7.2 Knowledge occurrence graphs

The knowledge registers were used to plot the knowledge events related to the different categories of ESAOA knowledge produced (namely the categories of data, process and innovation knowledge identified in Chapter 4) for each of the projects in the experiment. This involved generating three separate *knowledge occurrence graphs* for each project. In addition, a fourth graph was provided for each project that plotted the occurrences of productive and non-productive knowledge for each type of knowledge, ordered according to the event chains that occurred in the project. This task was facilitated by developing a program, referred to as the Partitioned Time History Calculator (or PTHC). The initial version of the program is detailed in Section 4.4, and was later redesigned to use event chains as a unit of analysis. In summary, the PTHC takes a knowledge register spreadsheet as input, and outputs the sequential accumulation of productive and non-productive time (or number of productive and non-productive events in the later version) separated into the data, process and innovation knowledge categories.

Knowledge occurrence graphs in Experiment 1 showed knowledge event number versus time, whereas those of Experiment 2 displayed event chain number versus number of event occurrences (which was an approximate indication of time, see Section 5.2). Graph 3.2 is an example of a knowledge occurrence graph (it is the data knowledge occurrence graph for Project P2-1).

Understanding of the knowledge occurrence depends on the understanding of knowledge event chains and knowledge occurrences counters. Each knowledge event chain in the knowledge register is associated to six knowledge occurrence counters (counters indicating the number of knowledge events that occurred for each type of knowledge produced; i.e., the number of productive data knowledge events, the number of non-productive data knowledge events and so on). Knowledge occurrence graphs essentially plot the accumulation of knowledge occurrences for a particular knowledge type vs. event chain number. The event chains are numbered chronologically (based on when their first knowledge event occurred – in the case that the same knowledge event is shared between two event chains, the ordering is based on the time of the second event). Figure 3.17 illustrates how the knowledge occurrence graphs increase in steps as the knowledge occurrences accumulate. The top half of Figure 3.17 visualizes knowledge events within event chains; the first three event chains (numbers 1, 2 and 3) had only productive events (signified in the

University of Cape Town

diagram as green tick marks); the last event chain had non-productive events (shown as the event chain cumulating in a dead-end). The graphs is the lower section of Figure 3.17 shown knowledge occurrence graphs; clearly, for a *productive* event chain, the *productive* graphs for the corresponding knowledge type steps up by one; similarly, for a *non-productive* event chain, the relevant non-productive graph steps up by one. Consequently, when viewing a knowledge occurrence graph, such as the one shown in Graph 3.2, the reader should be aware that each step in the productive trend line (the black line) corresponds to an event chain that was productive, and each step in the non-productive line (the dotted line) corresponds to an event chain that was non-productive.

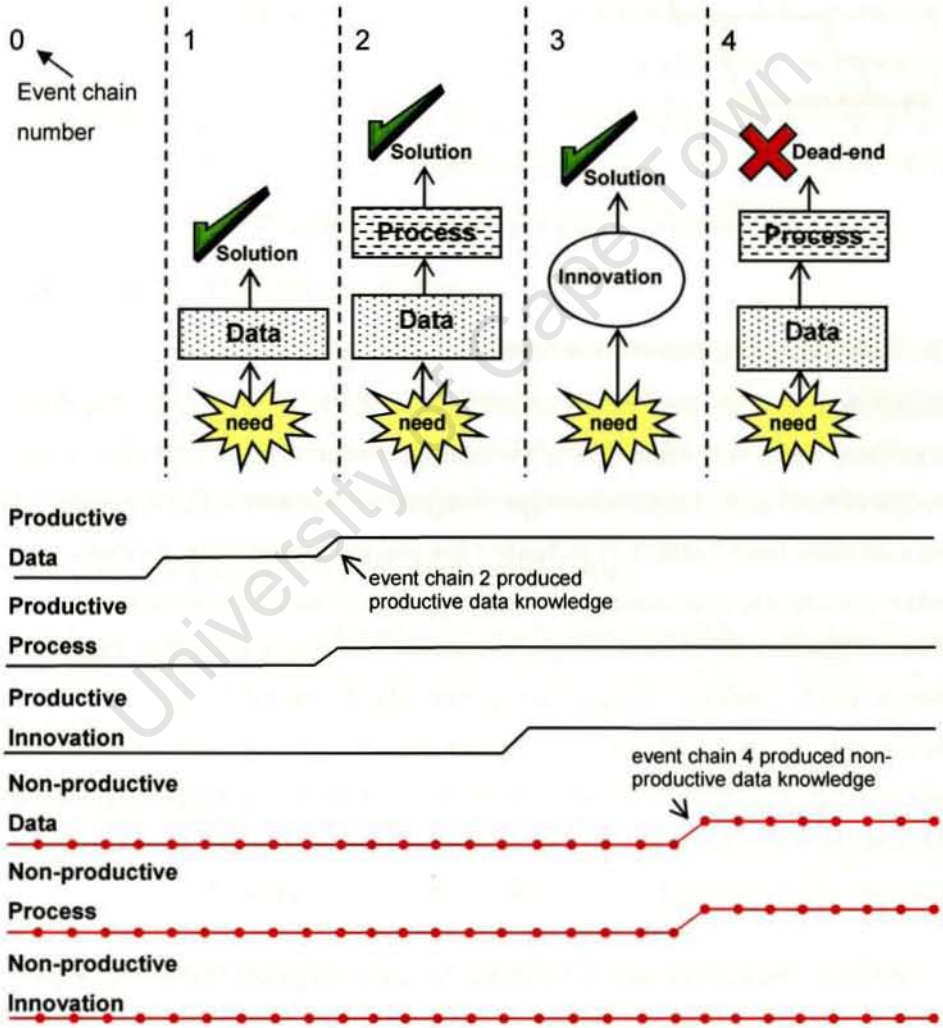
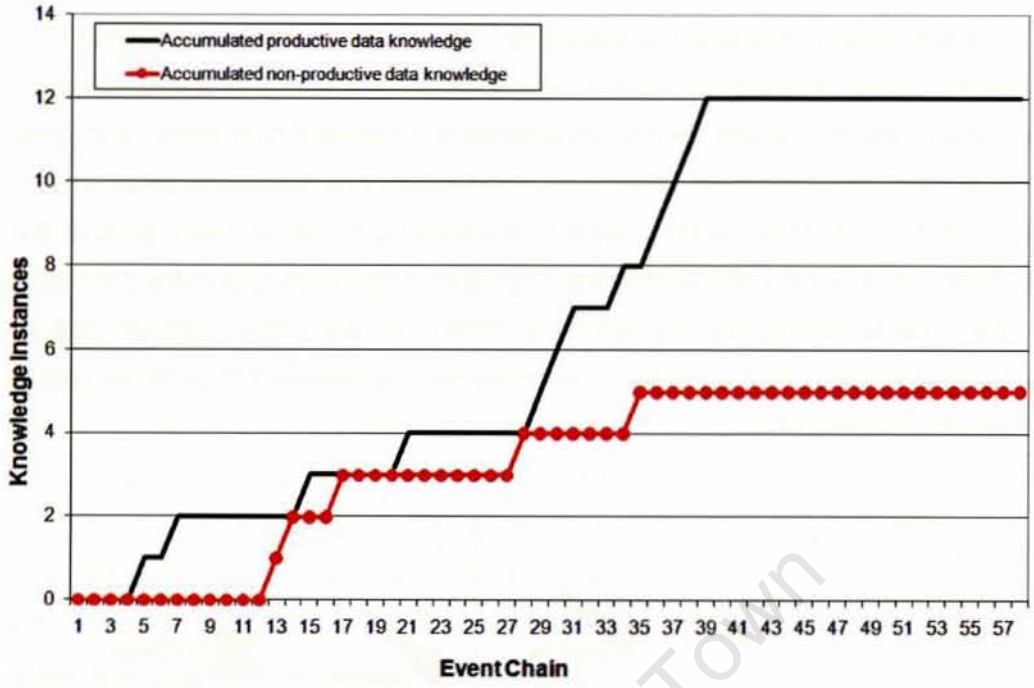


Figure 3.17: How knowledge occurrence graphs relate to event chains.

University of Cape Town



Graph 3.2 Data knowledge in Project P2-1.

3.9.7.3 Knowledge occurrence tables

Knowledge occurrences were summarised and tabulated in two ways. The first summary (see Table 3.6) showed the percentages of productive and non-productive knowledge within each of the knowledge categories identified in Experiment 1. The second summary (see Table 3.7) indicated the proportions of data, process and innovation knowledge produced by the project. The process knowledge, as discussed in Section 5.7.2, was further separated into role knowledge, logistics knowledge and knowledge of engineering methods (these sub-categories of process knowledge are indented in Table 3.7). Tables 3.6 and 3.7 use data from Project P2-1. The tables were used to analyse and compare knowledge occurrences within and across projects. Final summary tables were also created for the first and second experiments, which enabled their overall results to be compared.

Table 3.7: Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-1	PK	NPK	Tot
Data Knowledge	71%	29%	100%
All Process Knowledge	63%	38%	100%
Role	25%	4%	29%
Logistics	8%	8%	17%
Engineering methods	29%	25%	54%
Innovation Knowledge	82%	18%	100%

University of Cape Town

Table 3.8: Proportions of data, process and innovation knowledge produced.

Knowledge Type: P2-1	PK	NPK	Total
Data Knowledge	21%	9%	29%
Process Knowledge	26%	16%	41%
Innovation	24%	5%	29%
TOTALS	71%	29%	100%

3.9.7.4 Comparisons across experiments

In order to compare results between the two experiments, a common basis for comparison was needed. This was done by repeating the data analysis phase of Experiment 1 using the same methods as applied in Experiment 2. Correspondingly, the knowledge events were categorised according to productive and non-productive knowledge production, instead of according to productive time and non-productive time spend on knowledge events (as detailed in Section 3.9.5). The overall results of Experiments 1 and 2 could then be compared in a consistent manner.

3.9.8 Analysing other forms of data

The email archive, project discussion (or idea) forum, general group forum and developer logs were the main sources of data in this study. There were additional data sources which solicited participants' feedback, provided opportunities for participants to clarify issues, and to evaluate the work done during the projects.

3.9.8.1 Code and design reviews: ESAOA activities

The code and design reviews captured documentation regarding code and design changes, modifications to the ESAOA KMS, and allocation of further tasks and responsibilities.

3.9.8.2 Project meetings

Data obtained from meetings and design reviews resulted in identifying few additional learning tasks. In these cases, the developers seldom remembered past learning tasks accurately and frequently used their logs and code solutions to jog their memory. For these reasons, minutes from meetings and product demonstrations were not used as primary data sources (although they were referred to in evaluating the quality of the knowledge produced in Experiment 2).

University of Cape Town

3.9.8.3 End-of-product questionnaires

Basic statistical analyses were done of the ratings awarded by research participants. A content analysis was done of the qualitative comments provided.

3.10 Data synthesis

The data were collected from each project and then analysed to determine what problems were encountered, who produced ESAOA knowledge, which people used ESAOA knowledge, how artefacts were organised, used and exchanged, and so on.

The data synthesis phase took place after the data had been analysed; it involved generalising and combining the detailed results from the data analysis to form a more abstracted view of the KM processes, and to represent the 'ecology' (explained below) of the ESAOA KMS in use. This was a necessary step in order to view the results from a higher level and thus to determine the generalised findings for the study. Figure 3.18 illustrates the data synthesis process.

The analysis phase broke down the use of ESAOA knowledge into smaller related parts (i.e., the 'knowledge events'), as a means of analysing discrete phenomena of KM taking place during ESAOA activities. The synthesis phase went somewhat in the reverse direction, resulting in an abstract, visual representation of the system as a whole. This phase was needed to describe and explain the linkages between the different types of the knowledge, and the associated data, roles, and KM processes used in the projects studied. For this purpose, an 'ecology' of the development process was produced, which draws on the genre ecology framework [Spinuzzi, 2002], and the ecology mapping approach [Spinuzzi, 2003] that involves constructing a formal model of a genre ecology.

University of Cape Town

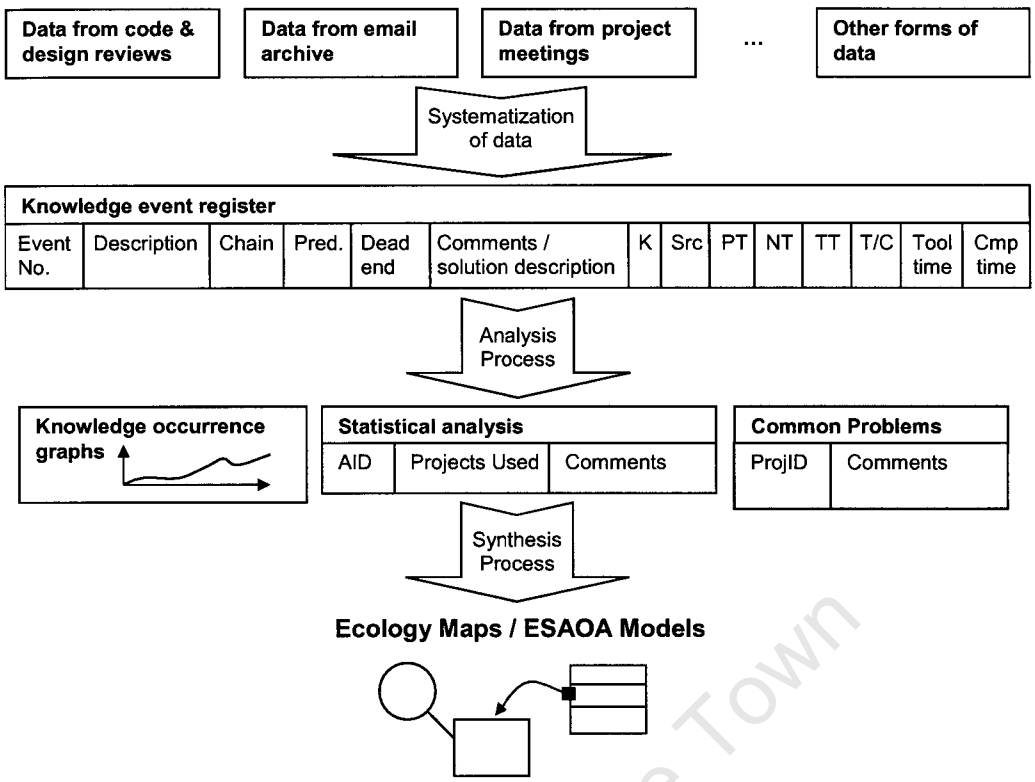


Figure 3.18: Data synthesis process, in which ecology maps were produced.

University of Cape Town

The genre ecology framework is an "analytical framework for studying how people use multiple artefacts ... to mediate their work activities" [Spinuzzi, 2002, pg.1]. Examples of these 'artefacts' include documentation, notes and similar items that are used during technical work. The genre ecology framework was developed specifically for technical communication research, its emphasis being on the interpretation of artefacts amongst developers [Spinuzzi, 2002]. The genre ecology framework concept has been brought into this research project as a means to represent knowledge work, with the focus on the interpretation of artefacts used in development (i.e., the tools and documentation, and the individuals who perform this interpretation). This relates closely to how knowledge is produced and communicated and to a lesser extent to the functionality of the artefacts [Spinuzzi, 2003].

The way in which the genre ecology framework has been applied in this project adopts methods used by Spinuzzi [2003] for constructing a formal model of genre ecologies. Accordingly, a specialised modelling language, namely the *ESAOA modelling language*, has been developed to provide formal models for visualizing and studying the 'ecology' of an ESAOA KMS. Section 3.11 describes the ESAOA modelling language. The genre ecology mapping technique and the ESAOA

modelling language are further explained, and used, in Chapters 4 and 6 in relation to the design of the ESAOA KMS.

3.11 The ESAOA Conceptual Modelling Language

The ESAOA modelling language is a graphical language that builds on version 2 of the OMG Unified Modelling Language (UML) standard [OMG, 2005; Rumbaugh *et al.*, 2005] and provides an implementation of a genre ecology framework, as discussed in Section 3.10, which is used in the data synthesis phase of the analysis process for this thesis.

The structure of the ESAOA modelling language described in this section represents the final version of the language, which was developed incrementally through the application of two framework construction phases (as per Section 3.5.1). Consequently, this modelling language is based on results of the research project itself (i.e., using results from Experiments 1 and 2). The initial version of the ESAOA modelling language was established during the preliminary study phase of this project (see Section 4.2), and this initial version was built up and improved during the subsequent framework construction phases.

The structure of the ESAOA modelling language is based on the requirements for modelling the roles, artefacts, knowledge and processes involved in ESAOA activities. The ESAOA modelling language is intended to be simple to use, allowing rough graphical models to be drawn quickly using pencil on paper. Simple geometric shapes are used for this reason.

The modelling elements of the ESAOA conceptual modelling language are separated into *atoms*, *connectors*, and *spaces*. *Atoms* are textboxes or icons that represent roles, artefacts, knowledge or processes. A *connector* represents relations between atoms (namely, *flows* and *associations*), and relations between connectors and atoms (called *connector junctions*). The *space* modelling element indicates boundaries and it is used to group modelling elements; spaces model workspaces and workstations. Each type of modelling element has a *principal shape*, upon which labels and various kinds of symbols are placed to refine the specific item to which the shape refers; for example, the principal shape of a role atom is a circle and, in order to refer to a team leader, the label 'TL' is written inside the circle. Figure 3.19 summarises the principal shapes for each type of modelling element.

University of Cape Town

The subsections that follow describe each of the modelling elements: Section 3.11.1 describes atoms; Section 3.11.2 details connectors, and Section 3.11.3 explains the spaces. Each subsection begins with a table that summarises the principal shapes for a modelling element, and is followed by a description of the specialised forms in which the modelling element appears.

3.11.1 ESAOA modelling atoms

There are four principal types of modelling atoms, which are shown in Figure 3.19. These principal types are: 1) roles, 2) artefacts, 3) processes, and 4) knowledge. Roles are drawn as circles, artefacts are rectangles, processes are pentagons, and knowledge atoms are represented as three equal width rectangles one on top of another. The role, artefact, and process atoms correspond to roles, artefacts, and processes that form part of the ESAOA KMS. Knowledge atoms are used to represent a certain knowledge form or body of knowledge managed by the KMS. Details concerning specialisations of these modelling elements are given below.

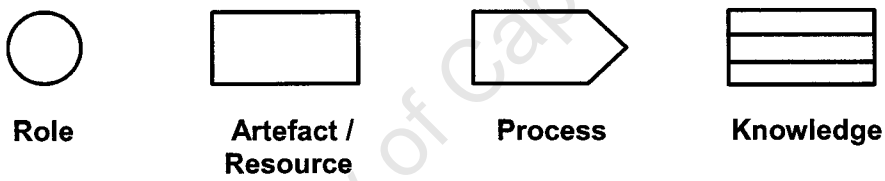


Figure 3.19: The principal shapes of ESAOA modelling atoms.

3.11.1.1 Role atoms

A role atom (drawn as a circle) represents a person who is responsible for performing certain KM activities in the ESAOA KMS. Specific roles are indicated by adding text in the middle of the circle to indicate the acronym for the role concerned. These specialisations are illustrated in Figure 3.20. The unspecified role can be used to refer to any role (i.e., it acts like a 'wildcard').

University of Cape Town

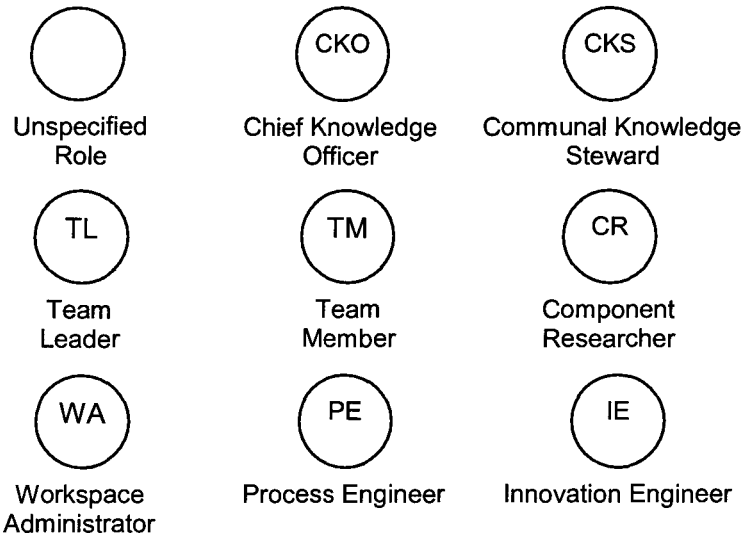


Figure 3.20: Role modelling atoms of ESAOA KMS version 2.

University of Cape Town

3.11.1.2 Artefact atoms

Artefacts are represented in the ESAOA modelling language using the atoms shown in Figure 3.21. Triangular UML inheritance connections are used in the figure to indicate a selection of the artefact atom refinements available in the ESAOA modelling language. As the figure shows, artefacts in the ESAOAS KMS are classified as hard or soft. A square is placed on the top left of an artefact atom to represent a hard artefact, whereas a triangle is placed on the top left of soft artefacts. Folders that contain soft artefacts are represented using the UML package icon (note that, for ease of drawing, a triangle is not placed on the top left of folder atoms).

As per the ESAOA knowledge ontology, an artefact can be further classified according to the workspace from which it originates and the role responsible for maintaining the artefact. Further functionality classifications can also be applied; such classifications are indicated by placing the relevant acronyms on the top left of the artefact atom. Figure 3.22 provides an example of a soft artefact, named a 'component list', which has been given three classifications, namely: data artefact, boundary artefact and team artefact. Table 3.9 lists acronym classifications that appear commonly in the models used in this chapter.

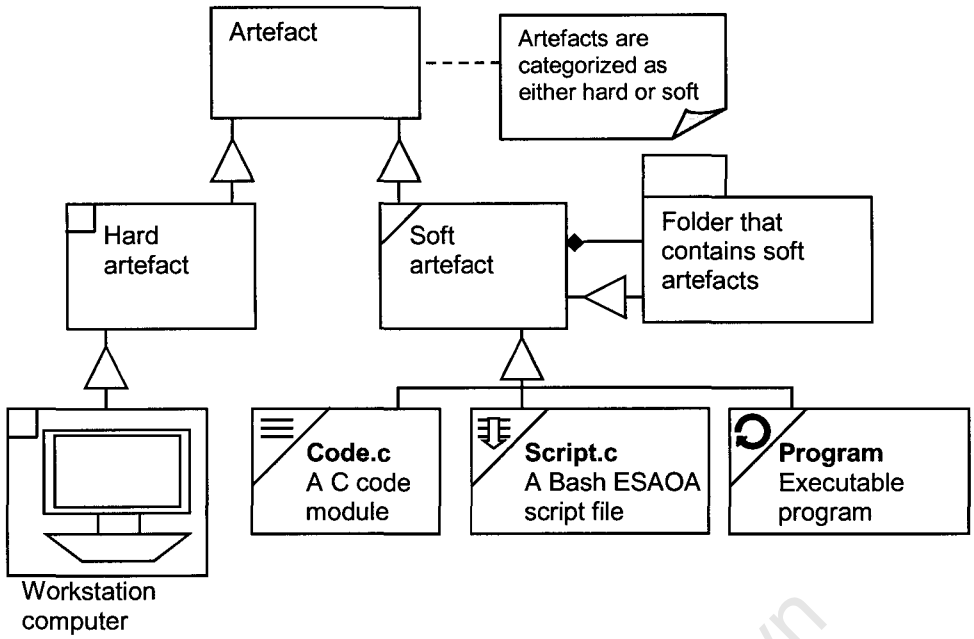


Figure 3.21: Artefacts atoms showing general artefact, hard artefact, soft artefact, and more specialised artefact atoms.

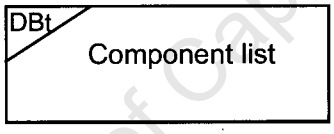


Figure 3.22: A team boundary data artefact.

Table 3.9: Commonly used artefact classification acronyms.

Acronym	Description
B	Boundary artefact (facilitates knowledge transfer)
c	Communal artefact (stored in communal workspace)
D	Data artefact (e.g., datasheet)
I	Innovation artefact (e.g., concept sketch)
K	Knowledge artefact (used to create knowledge or to make knowledge explicit)
P	Process artefact (e.g., script or a how-to guide)
t	Team artefact (stored in team workspace)

3.11.1.3 Process atoms

Process atoms relate to KM processes of the ESAOS KMS; the title of these atoms contains keywords describing the process concerned. Additional classifications can be assigned to process atoms following a similar approach as used for artefacts; in other words, a process classification acronym can be added to the top left of process atoms. Figure 3.23 demonstrates the use of process classification by assigning the

University of Cape Town

classifications of 'applying knowledge' and 'group work' to a process atom titled 'Training'. Table 3.10 lists process classifications and their corresponding acronyms.



Figure 3.23: A group-work process that involves application of knowledge.

Table 3.10: Process classifications.

Acronym	Process classification
A	Applying knowledge
D	Documenting tasks
F	Finding information
g	Collaboration / group activity
i	Independent activity
P	Producing knowledge
R	Representing knowledge
S	Problem solving / searching for solutions

3.11.1.4 Knowledge atoms

A particular knowledge form produced or applied in ESAOA activities is represented in the ESAOA modelling language by knowledge atoms. A knowledge atom looks like an UML class modelling element, comprising three rectangles on top of each other (see Figure 3.24). The top rectangle indicates the title (or keywords) of the knowledge form and its classification. The middle has keywords that indicate an individual's understanding of this type of knowledge and the competencies related to producing it. The bottom rectangle lists the capabilities required to apply the knowledge during development procedures.

Knowledge atoms can be classified by adding knowledge classification acronyms to the top left of these modelling elements; the classification acronyms are listed in Table 3.11. These knowledge categories are based on the findings from Experiment 2 (see Chapter 5). Figure 3.24 shows a knowledge atom that has been assigned the classification of process knowledge by using the acronym 'P' on the top left.

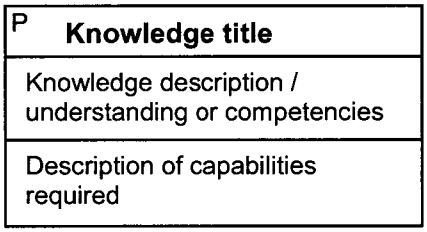


Figure 3.24: Knowledge atom classified as process knowledge.

University of Cape Town

Table 3.11: Classification acronyms for knowledge atoms.

Acronym	Knowledge classification
D	Knowledge of data
I	Knowledge of innovation
P	Knowledge of process
Pe	Knowledge of engineering methods
Pl	Knowledge of processes relating to logistics
Pr	Knowledge of processes relating to roles
Pk	Productive knowledge
npk	Non-productive knowledge

3.11.2 Connectors

Connectors of the ESAOA conceptual modelling language are used to represent relations between modelling elements. There are three principal connector types: flows, associations and connector junctions. Connector labels can be used to add keywords that provide additional information about connectors. The subsections that follow explain the types of connectors and how to use them.

3.11.2.1 Flows and associations

Connectors are used in the ESAOA conceptual modelling language to show *associations* and *flows* between modelling atoms. An *association* indicates an important relationship between atoms. A *flow* represents a transformation processes that occurred in which one atom is produced, or modified, by another. Twelve types of connectors are used in the modelling language. Flows are shown as double links, while associations are shown as single links. The names of the connectors are listed in Figure 3.25 together with illustrations for each type. Appendix C.2 provides further descriptions and example models that provide more detail on connectors.

University of Cape Town

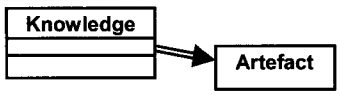
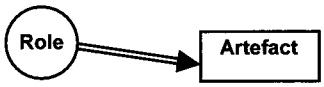
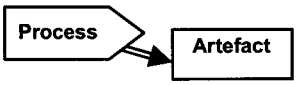
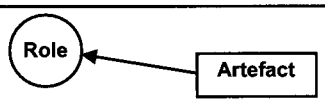
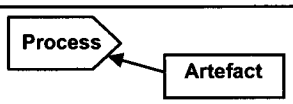
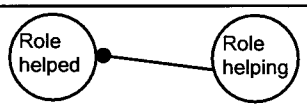
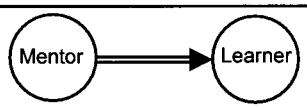
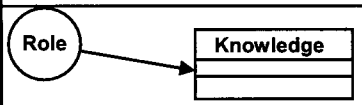
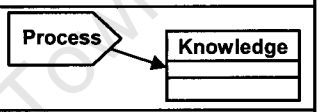
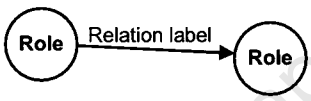
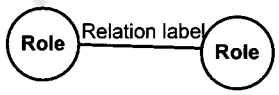
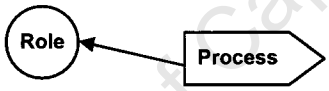
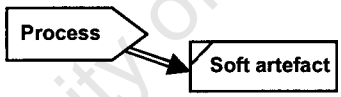
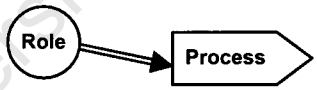
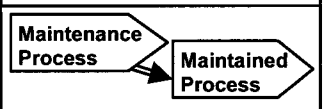
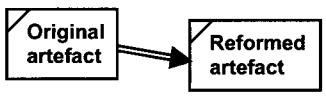
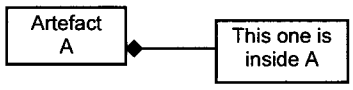
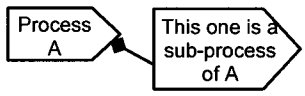
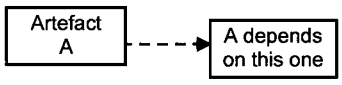
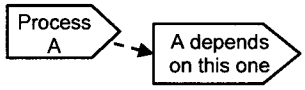
Connector name	Arrangement	Alternate
Knowledge capture flow		N/A
Artefact adaptation / maintenance flow		
Artefact use association		
Role support association		N/A
Mentoring flow		N/A
Knowledge use/produce association		
Role interaction association		
Role perform association		N/A
Process capture flow		N/A
Process maintenance flow		
Artefact conversion flow		N/A
Containment association		
Dependency association		

Figure 3.25: List of ESAOA modelling language connector types.

3.11.2.2 Connector junctions

A connector on its own provides limited information about the relation it represents. Connector junctions indicate which additional modelling atoms are involved in a relationship, such as artefacts (e.g., email, development tools and laboratory

University of Cape Town

equipment), which are involved in the relationship. A connector junction is modelled as a small circle in the middle of a connector, and has dotted lines that join to additional atoms that are involved in a relationship. There are two main types of connector junctions: a) tacit junctions, and b) explicit junctions. Tacit junctions are represented as an unfilled circle (i.e., a ring) on a connection – these junctions link atoms that relate predominantly to tacit knowledge and KM activities that are difficult to document. Explicit junctions are drawn as solid circles on a connection; such junctions link atoms that relate more to explicit knowledge or artefacts (e.g., knowledge that can be easily documented, information resources and software). Figure 3.26 illustrates the difference between explicit and tacit junctions; Figure 3.26 (a) uses an explicit junction, whereas Figure 3.26 (b) uses a tacit junction.

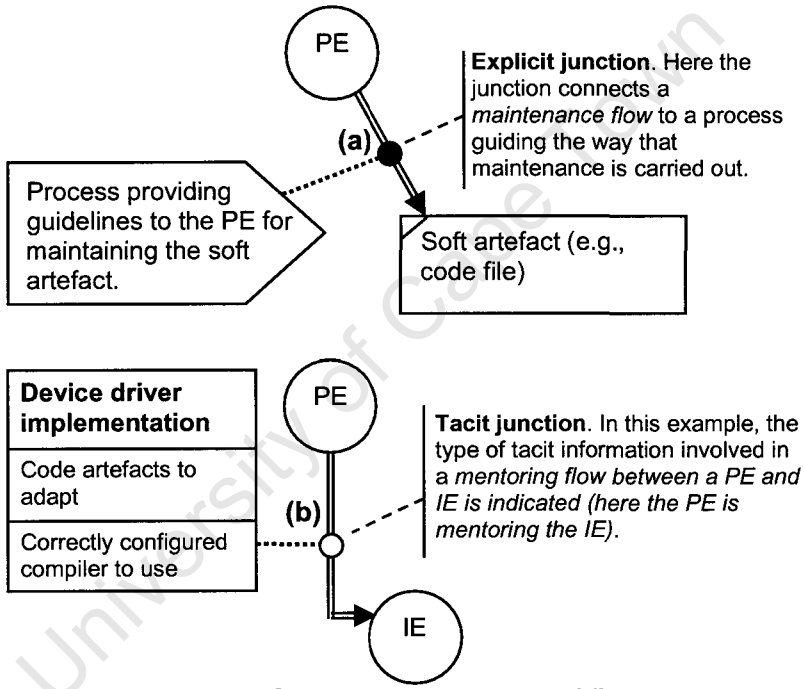


Figure 3.26: Examples of (a) explicit junction, and (b) tacit junction.

3.11.2.3 Connector labels and multiplicity

Two types of annotations can be added to a connector, namely: a connector label and multiplicity. A connector label is a set of keywords placed mid-way on the connector that provides additional information describing the relation concerned. Multiplicity can also be indicated on one or both ends of a connector, and is shown using the same approach as used in the UML, which is thoroughly described by authors such as Arlow [2005] and Douglass [2000]. Figure 3.27 provides an example of connector labels and multiplicity, showing a role interaction that involves correspondence between one person performing role R and one to three people

University of Cape Town

each performing role S. One or more multiplicity (i.e., '1..*') is implied for connectors that do not explicitly indicate multiplicity.

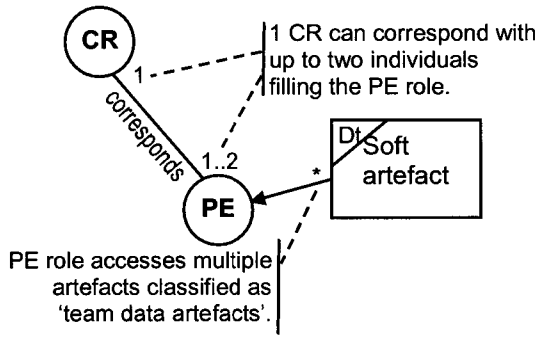


Figure 3.27: Model demonstrating connector labels and multiplicity.

3.11.3 Spaces

The *space* modelling element is used to demarcate boundaries in order to partition modelling elements in a model into groups. *Spaces* are used in the ESAOA modelling language to indicate elements that belong to certain workspaces, or to particular workstations. Spaces are drawn as rectangles with dotted lines for sides, and they have a title shown in bold capital text that is placed next to one of the sides of the boundary (usually at the top, if space permits). Figure 3.28 shows a model in which two *space* workspace modelling elements are used; the top one models part of a communal workspace, and the bottom models part of a team workspace; using these spaces, it is easy to see to which spaces certain items belong. Connectors can intersect boundaries of spaces if relations exist between modelling elements in different workspaces (as is the case between the CKS and WA in Figure 3.28).

3.11.4 Comments and constrains

ESAOA models use the standard UML method for adding notes and constraints. Comments related to a modelling element are added using a UML note anchored to the modelling element by a dotted line (as shown in Figures 3.21), or by using a dotted line and note bar (as illustrated in Figure 3.28). Connector constrains are specified using comments anchored to the relevant connectors.

University of Cape Town

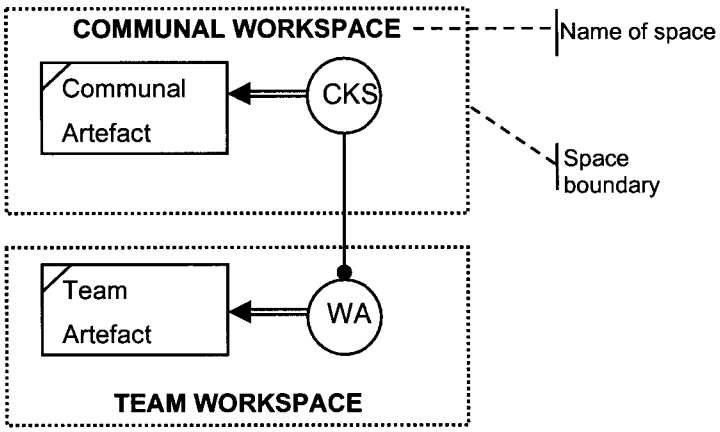


Figure 3.28: Model showing space modeling elements.

3.11.5 External processes and artefacts

An artefact or process atom drawn using dashed lines (instead of solid lines) means that the atom is an external item, and is neither related to KM activities of the ESAOA KMS, nor is it stored or maintained within an ESAOA workspace. Figure 3.29 shows a model in which an artefact named 'development tool' is an external artefact, and a process named 'Process P' is an external process. Both the external artefact and the external process shown in the figure are used by the PE, but neither of them is maintained as part of the KMS (i.e., their adaptation and maintenance are entirely ad-hoc and undefined within the KMS).

University of Cape Town

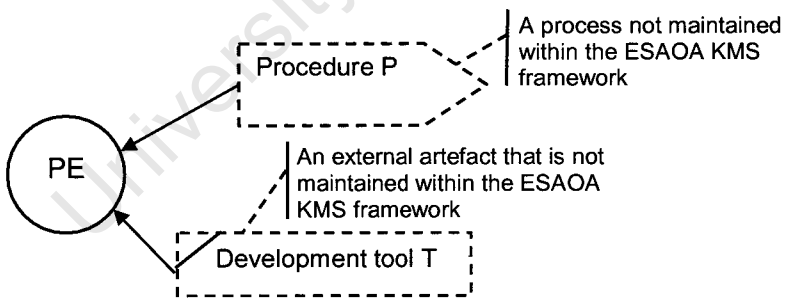


Figure 3.29: Model showing an external process and external artefact.

3.12 Comparing artefact and prototype quality with KMS analysis results

As inculcated in Section 3.5.1, assessment of product artefacts and the quality of product prototypes were performed at the end of Experiment 2. This was done in order to obtain data related to final prototype (rather than the KM methods) in order to compare KMS analysis results of a project to the overall quality of prototype produced from the project. This was done in order to gain insights into potential ways to predict the quality of the final product produced by a project based on KM trends observed during the project.

The product demonstrations were performed at the end of the second experiment and were evaluated by the review panel (described in Section 3.6.5). The review panel completed a demonstration check sheet (described in Section 3.8.6) for each team. After the demonstrations, the final version of the project artefacts (e.g., schematics, design documents and code modules) produced by each team were evaluated by the review panel using the requirements check sheet. Section 3.8.6 describes the requirements check sheet, which has a section devoted to rating the quality of artefacts stored in ESAOA team workspaces.

The methodology to compare results obtained form the experiment, such as comparing the quality of project artefacts to proportions of productive knowledge for each project, were done using correlations. The correlations using the standard linear Pearson's correlation formula [Graham, 2008; Wikipedia, 2008] shown below:

$$r = \frac{N\sum xy - (\sum x)(\sum y)}{\sqrt{[N\sum x^2 - (\sum x)^2][N\sum y^2 - (\sum y)^2]}}$$

The value *r*, which is the correlation result, will be between -1.0 and +1.0. The closer *r* is to +1.0 or -1.0, the more closely the variables are related; if *r* is closer to 0, then the variables are not related. If *r* is positive, it means that as the one variable increases, so does the other variable. If r is *negative*, it means as one variable increases, the other variable decreases, which is referred to as an 'inverse correlation' [Underhill & Bradfield, 1998]. Both positive and negative correlations will be looked at to determine relations that may be positive, or negative, influences. Chapter 5 details correlations performed for Experiment 2.

University of Cape Town

3.13 Conclusion

The subsequent chapters present the results of applying the research methodology described in this chapter. Chapter 4 presents the findings from Experiment 1, which comprises the first application of the KMS analysis phase, together with the design of the initial ESAOA KMS (ESAOA KMS version 1), which resulted from the first application of the framework creation phase. Chapter 5 discusses the findings from Experiment 2, which constitutes the results of the second application of the KMS analysis phase. Chapter 6 describes the design of ESAOA KMS version 2, which resulted from the second application of the framework creation phase. Lastly, Chapter 7 summarises the results and presents the conclusions, reflecting back on the research question, and making recommendations for future work.

University of Cape Town

University of Cape Town

Chapter 4:

First experiment findings and ESAOA KMS version 1

This chapter presents the results of the first stage of the research design described in Chapter 3 (see Section 3.5). During this first stage, Experiment 1 was performed; in this experiment, data were captured from a set of development teams, who used their own *ad hoc* KM methods to develop ES products. A preliminary study of the Experiment 1 data was done during the establishment of initial data analysis methods for this thesis. This process of establishing data analysis methods built on the KM literature; the preliminary study was used to determine how theories from the literature could be applied in the specific context of ES development. Experiment 1 was followed by the first application of the *KMS Analysis* (the 'A phase'), which involved analysing data from Experiment 1. Next, *Framework Construction* (the 'C phase') was performed to design and construct the initial ESAOA KMS. The objectives of this first stage of the research process were twofold: 1) experimentally develop a strategy to investigate ESAOA knowledge produced during ESAOA activities and to evaluate the effectiveness of the KM methods used, and 2) construct an initial version of the ESAOA KMS, which would be used in the second stage of the research design.

Section 4.1 starts with a brief overview of the first experiment. Section 4.2 goes on to describe the preliminary study of the data, which was used to construct a context for studying KM of ESAOA activities, and culminated in the initial version of the data analysis method for this thesis (described in Section 3.9). The data analysis method was then applied to the original data of Experiment 1 (i.e., the preliminary study was used as a means of developing the analysis method); the results are summarised in Section 4.3. The results of trend analysis, the last step in the analysis process, are presented and discussed in Section 4.4. The high-level design of the ESAOS KMS version 1, which was based directly on the results of Experiment 1, and the strategies used to represent the KMS, are the focus of Section 4.5. The implementation of ESAOA KMS version 1 is described in Section 4.6. Lastly, Section 4.7 concludes this chapter and leads into the next chapter.

University of Cape Town

4.1 The First Experiment

The first experiment comprised two ES product development projects: P1-1, the development of a Software Signal Generator (or SoSiG), and P1-2, an Antenna Controller (or ANTCON). Each project team comprised two members (Section 3.6.4 detailed how the participants were selected), and the projects are briefly described in Table 4.1.

Table 4.1: Description of projects for the first experiment.

Project Number	Project Title	Project Description	Motivation
P1-1	Software Signal Generator (SoSiG)	Programmable signal generator, able to communicate and program using a standard ASCII terminal over a serial interface.	Similar to products used by commercial developers.
P1-2	Antenna Controller (ANTCON)	Control system to control and monitor the azimuth and elevation of an antenna pedestal. Connects to parent RADAR control computer over TCP/IP using Ethernet.	Based on an existing RADAR control system used in military applications.

The objective of project P1-1 (SoSiG) was to develop a portable code library for a digital signal generator that controlled an actuator circuit connected to the CSB337 via an I2C hardware interface [NXP Semiconductors, 2007]. The code library had to include a state machine to drive the actuator and a simple command-based user interface that could be integrated into application code through a generic input/output byte-stream wrapper.

The aim of project P1-2 (ANTCON) was to use the CSB337 evaluation board to control the azimuth and elevation of a radar antenna pedestal. In this project, a proprietary interface was used to connect the pedestal electronics to the CSB337, which was linked to the programmable input/output pins of the AT91RM9200 microcontroller. A simple command-based user interface had to be provided via an RS232 port.

Chapter 3 (Section 3.8) described the data collection methods used in this experiment. Data was captured from code and design reviews, email archives, demonstrations and various other sources.

University of Cape Town

4.2 Preliminary study to establish the data analysis method

This section discusses the establishment of the initial data analysis method used in this thesis, which was based on a preliminary study of Experiment 1 data. In the experiment, all the ES engineers used their own *ad hoc* KMS. This preliminary study investigated the KM methods used by the developers, and the results of the study were used thereafter to develop the experiment data analysis processes for this thesis (see Section 4.2.12). The results of this preliminary study, in conjunction with the later application of the data analysis method, were then used to guide development of the initial ESAOA KMS (see Section 4.6).

4.2.1 Denoting artefacts and ESAOA activities in the data

At the beginning of the preliminary analysis all the data, arranged chronologically, was printed out. The data printouts were then worked through from the first entry, underlining artefacts that were mentioned, identifying activities, and annotating records related to adapting or managing the artefacts (annotations are shown as text in square brackets in the excerpts from the data). The third column in Table 4.2 shows excerpts from the data, illustrating how this was done.

At this stage, simple pencil on paper drawings (which later became the more formalised ESAOA conceptual modelling language) were utilised as a visual aid to keep track of artefacts and processes, and the relations between them. These rough diagrams were used during discussions with participants of the study to gain further insights into ESAOA activities and artefacts, and to confirm that the researcher had modelled processes accurately. Figure 4.1 provides an example of such a diagram. The model in the example was used to visualize the relations between the artefacts and processes used by team members, which existed during development of a device driver for a peripheral (namely, the PCF8591 digital-to-analogue converter).

4.2.2 Verification of KM models

The first step taken in the preliminary study was to verify that the data obtained from the first experiment did actually represent KM methods (as based on the literature in Chapter 2). To this end, the data were investigated in relation to the 'knowledge process' [Radding, 1998] and 'knowledge flows' [Milton, 2005], which set out general models of KM (see Chapter 2).

University of Cape Town

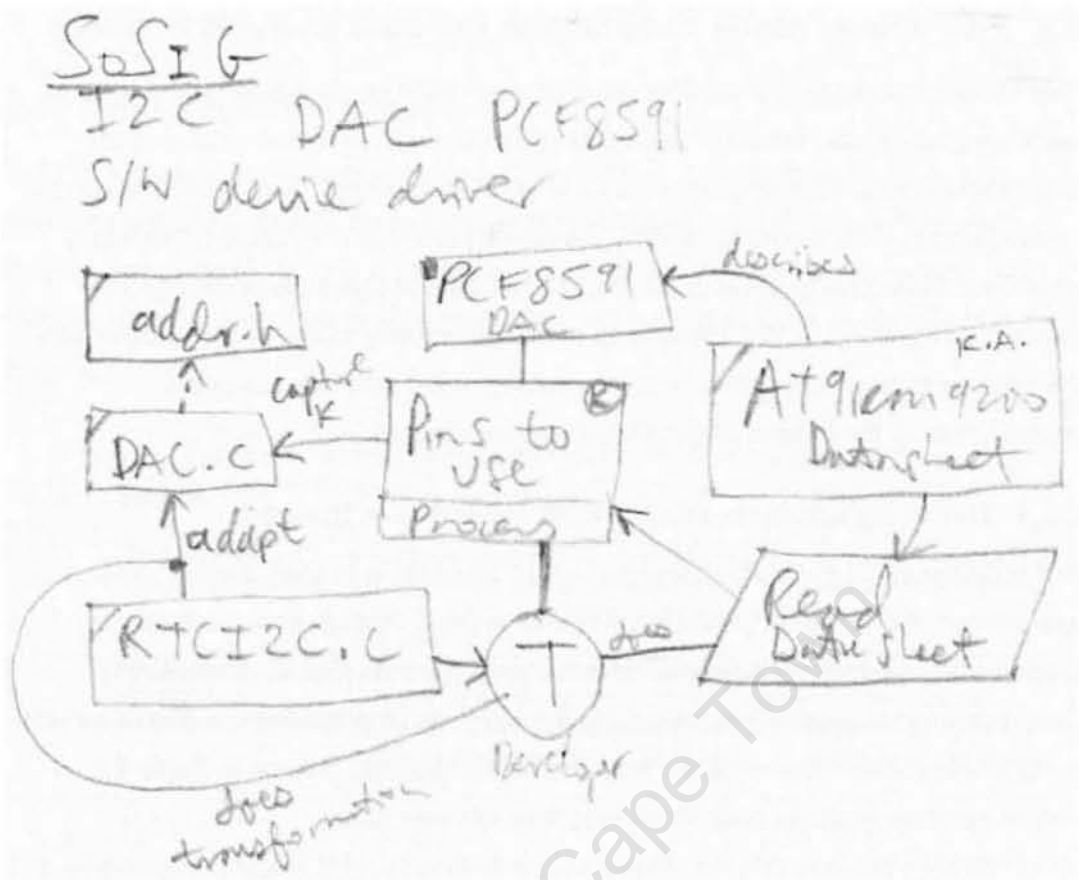


Figure 4.1: An initial version of the ESAOA modelling language used to describe relations between ESAOA artefacts and activities in Project P1-1

The data from Experiment 1 verified existence of the four phases of the 'knowledge process' [Radding, 1998]. Table 4.2 gives an excerpt from the data, using the following columns: column 1 is the record number that refers to records shown in the specific table (for each project, these numbers start from 1 and run sequentially); column 2 indicates the data source; column 3 is an excerpt from the data; and column 4 describes how the record confirms the KM models.

Table 4.2 contains evidence pertaining to all four phases of the 'knowledge process' [Radding, 1998], specifically that knowledge is: 1) captured (see entry 3 in Table 4.2); 2) stored (see entry 3); 3) processed (see entry 4), and 4) communicated (see entry 1). This is clearly evidence that 'knowledge flows' [Milton, 2005] are occurring. For example, entries 1 and 2 in Table 4.2 show that the supervisor at times acted as a 'knowledge supplier' and the developers as 'knowledge consumers'.

University of Cape Town

Table 4.2: Excerpt from Project P1-1 used to verify existence of 'knowledge processes' and 'knowledge flows'.

Record No.	Data source	Excerpt from data	Narration
The data records pick up from minutes of project meeting 1, held 10 June 2004			
1	Project Meeting 1	What computers are we to use?	The developers asked the supervisor if lab computers could be reserved. This except demonstrates the communications phase of Radding's [1998] 'knowledge process'.
2	Project Meeting 1	Windows lab computers	The supervisor tells the developers that 'Windows lab computers' can be reserved. Here, the supervisor is acting as a knowledge producer, and the developers as knowledge consumers in terms of Milton's [2005] 'knowledge flow' model.
The data records continue with the developers' log			
3	Dev. Log	Various free and shareware options are available [comms software]. Must run on Windows. HyperTerm may provide all the needed features.	The log shows that the developers performed a variety of internet searches for communications software. In this case, the developers were involved in capturing 'raw' knowledge through downloading documents and jotting down relevant points observed in a log; this demonstrates instances of the 'capturing' and 'storage' phases of Radding's [1998] 'knowledge process'.
... A variety of other issues was recorded in the log concerning other issues...			
4	Dev. Log	Look through manual for protocol settings [found settings]. 38400bps, 8 data bits, no parity, one stop bit, no flow control...	The developers looked for information in a manual regarding protocol settings, tested various settings using 'HyperTerm' and then saved working settings. Here the process and storage phases of Radding's [1998] 'knowledge process' is shown: the developers *processed* 'raw' knowledge of protocol settings and configuration techniques by adapting and testing possible choices using specific tools; the developers also *stored* their knowledge by saving working settings and by entering information into their log.

University of Cape Town

4.2.3 Problem-solution cycles

The data were scrutinized with the aim of identifying commonly occurring techniques and activities relating to KM performed during ESAOA activities. The first and most noticeable observation involved finding the same high-level strategy used by both Project P1-1 and P1-2 teams to solve problems in order to accomplish ESAOA activities. It emerged that both teams had followed a form of *problem-solution cycle,* which tended to comprise the following three parts: 1) encountering a new problem; 2) working on (and testing) a solution; and 3) searching for information or resources. A new problem-solution cycle was started, or a previous one resumed, when the developer changed focus from one problem to another. Almost all the activities

observed in the data from the email archives, the minutes of meetings and the developer logs, could be allocated to one of the stages of these problem-solution cycles. However, the data from the product demonstrations did not exhibit this cycle (probably because activities performed in demonstrations were pre-rehearsed).

Instances of the problem-solution cycle are demonstrated in Table 4.3. The table shows data samples taken (in chronological sequence) from Project P1-1; these are accompanied by a narration explaining how each data sample fits into a problem-solution cycle. The table further shows how the developers moved from encountering new problems, via seeking information, to working on solutions. The first column of Table 4.3 gives the problem number (these are numbered from the first problem observed in the minutes of project meeting 1)[1]. The second column indicates the data source in which the problem was first observed. The narration in the third column describes how the observation from the data fits into the cycle (ESAOA artefacts are underlined in this column). The last column indicates whether that particular observation resulted in a solution (i.e., a solution to the problem whose number is in the first column). Annotations by the researcher concerning recorded data are in square brackets.

University of Cape Town

Table 4.3: Demonstration of problem-solution cycles observed in Experiment 1.

Prob. No.	Data source	Excerpt from data	Narration	Solved ?
The data records picks up from the minutes of the project meeting 1, held 10 June 2004				
1	Project Meeting 1	What computers are we to use?	The first new problem identified in the project meeting was that the developers wanted to know if they could reserve lab computers.	N
1	Project Meeting 1	Windows lab computers	The supervisor indicated that the 'Windows lab computers' could be reserved. The problem 1 was thus solved quickly, and did not involve searching for information.	Y
... A variety of other issues were recorded in the minutes ...				
8	Project Meeting 1	Connect CSB337 to PC	Further on in the minutes, a more complex problem was found: the developers wanted to know how to connect to (and communicate with) the embedded platform from a PC.	N
8	Project Meeting 1	Use either a RS232C ... connection. Alternatively use an Ethernet cross-over cable ...	The supervisor suggested a variety of possible solutions to problem 8, essentially providing information.	N

[1] The numbering used here runs sequentially from 1 up to the last entry for a particular project; e.g., the first meeting minutes may be number 1, the first email assigned number 5.

9	Project Meeting 1	Where can we get cables?	The suggestions (above) for problem 8 led to a new problem (no. 9): knowing where to get resources needed to implement the suggested solutions.	N
… the minutes continue …				
The data records picks up from the developer log, from 10 June 2004				
14	Dev. Log	PC software to talk to CSB337 over RS232C?	Later, a developer records a new problem: communications software is needed. This problem is related to the earlier problem (no. 8).	N
14	Dev. Log	Various free and shareware options [software options] available. Must run on Windows. HyperTerm may provide all the needed features.	The log shows that a variety of internet searches for communications software was done (finding various options) – this was an instance of information-seeking related to problem 14. The log also suggests a tentative solution (i.e., HyperTerm) but does not confirm it is a solution yet.	N
… A variety of other issues was recorded in the log concerning other issues…				
17	Dev. Log	RS232C protocol settings	After deciding which communications software to use (i.e., having solved problem 14), the developers returned to problem 8 and soon encountered a new problem: what protocol settings to use.	N
17, 8	Dev. Log	Look through manual for protocol settings [found settings]. 38400bps, 8 data bits, no parity, one stop bit, no flow control…	To solve problem 17, the developers looked for information in a manual. They found the settings needed for problem 17; after correcting and saving the HyperTerm settings, the developers solved problem 8.	Y

It is further evident from Table 4.3 above that some solutions were arrived at more quickly than others were. There could moreover be long delays between identifying a problem and attempting to solve that problem (e.g., the data showed there was a delay between the identification of problem 8 in the meeting, and the entry in the developer's log that noted when a solution to the problem was worked on).

It was found that the problem-solution cycles observed in the data occurred at a different level to those of the 'knowledge process' [Radding, 1998] and 'knowledge flows' [Milton, 2005] discussed in Section 4.2.2. The knowledge process provides a different perspective to abstracting the progression of activities that took place in the projects, in comparison to the perspective used in this document, which will be described next.

University of Southampton

4.2.4 Trivial and non-trivial solution cycles

The complexity of solutions performed to complete ESAOA varied (i.e., some of the problems were easier to solve than others were). Two categories of solutions were devised to identify solutions that could not be solved quickly. These solution categories are *trivial solutions* and *non-trivial solutions*. Problem 1 in Table 4.3 is an example of a trivial solution: a developer essentially asked a question and the problem was quickly solved a few minutes later by the supervisor[2]. Problem 8, in contrast, is an example of a non-trivial solution: this problem required more work in arriving at a solution than Problem 1, in terms of both the number of activities recorded and the amount of effort involved in each of these activities. While the solution to Problem 1 concerned only two data entries (one recording a question and the other a response), the solution to Problem 8 consisted of seven entries. Problem 8 led to a set of sub-problems (problems number 9, 14 and 17), and each of these sub-problems in turn took a certain amount of time to solve. Consequently, a solution that has sub-problems, or involves looking up information, and takes in total more than a few minutes to solve, is classified as a non-trivial solution.

Further investigation of the data showed that the three-part problem-solution cycle described in Section 4.2.3 was an oversimplification for non-trivial solutions. Non-trivial solutions tended to be more complex, in that the parts 'working on a solution' and 'searching for information' involved many commonly recurring activities. While trivial solutions often had 'working on a solution' or 'searching for information' as part of the cycle, these steps were generally accomplished easily (e.g., when a simple working solution was provided and when it worked on the first try). The problem-solution cycle proposed earlier was thus restructured as two different models: the *trivial solution cycle* (see Figure 4.2) and the *non-trivial solution cycle* (see Figure 4.3).

The trivial solution cycle shown in Figure 4.2 was made up of three steps, which were usually in the sequence indicated, namely: when a new problem is encountered, a potential solution is discovered with little effort (e.g., as happened for problem 1 in Table 4.2), leading to a working form of the solution which allows the necessary ESAOA activities to be accomplished quickly.

---

[2] In other cases, delays may happen between the initial identification of a problem, and the moment when the problem is observed or recorded in the data. The developers tended to log actions they tried, rather than journaling about problems they might examine.

University of Cape Town

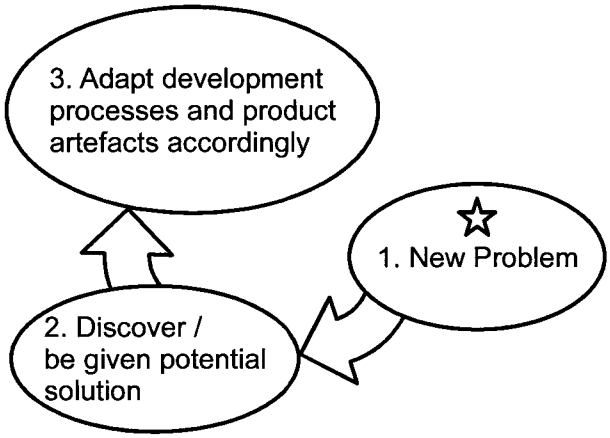


Figure 4.2: The 'trivial solution cycle'

In the case of non-trivial solutions, the problem-solution cycle was expanded into eight related actions, as shown in Figure 4.3. The cycle starts with the decision to solve a new problem (step 1 in Figure 4.3). Typically, the second step involves searching for the information and resources needed to learn more about the problem and to find information that will assist in formulating a solution. In the third step, the developers made sense of the found information and resources; this step often took time, sometimes many hours, to achieve (this step has many similarities to the processing phase of Radding's [1998] KM model). Much of the information obtained in the third step was frequently discarded (as can be observed in Table 4.3 in terms of communications software being found, and investigated, but not used further in the project). The difficulty of completing the third step was most likely exacerbated because the teams comprised only novice engineers (i.e., had there been more than two team members, who were better experienced, the third step would probably have been completed more efficiently). In the fourth step, a working solution started to take shape once the developers understood the relevant information. The attempts in the fourth step to find a working solution culminate in an extensive and decisive experiment in the fifth step to determine the overall effectiveness of the chosen solution. The sixth step involved considering the results of the test (e.g., in respect of the communications problem, which was problem 17 in Table 4.3, the success of the solution was tested by verifying that data sent from the embedded platform was reliably received by the PC). Based on the results obtained in the sixth step, the solution was regarded as either successful (step 7.1) or unsuccessful (step 7.2). Typically, if the test was successful, then the solution was incorporated into the team's overall development process; otherwise, it was discarded and the team learned from their mistakes. If the solution was successful, the team could continue carrying out the ESAOA activities according to the implementation tasks they wanted

University of Cape Town

to accomplish (step 8 in Figure 4.3). Furthermore, even if the solution was unsuccessful, team members might still be able to perform the necessary ESAOA activities due to knowledge gleaned from the process of developing a solution. For example, the Project P1-1 developers spent hours trying to solve a problem concerning interrupt code, but their solution was unsuccessful in the end. Nonetheless, because of the knowledge they acquired in the unsuccessful attempt to find a solution, the developers were inspired to improve their poll code. This example is based on events 66 and 67 in the data given in Appendix A1.

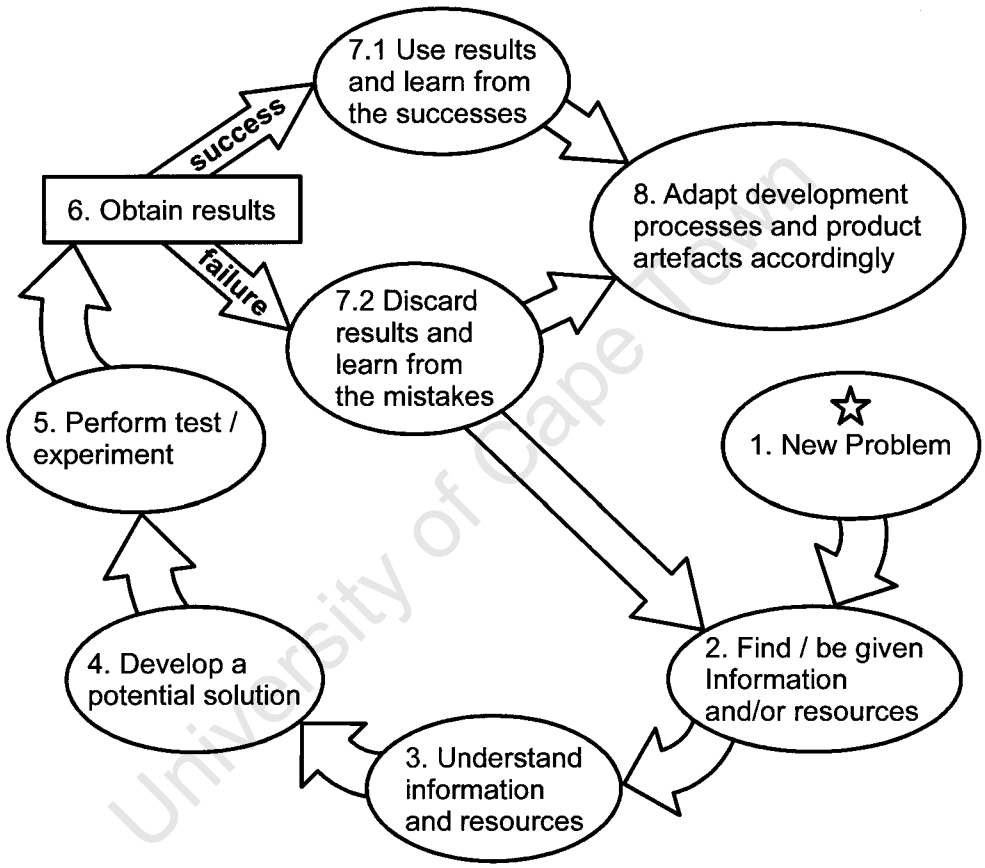


Figure 4.3: Illustration of the 'non-trivial solution' cycle

The steps for non-trivial solutions did not always follow the precise sequence shown in Figure 4.3. For example, a developer might decide to start immediately with a potential solution without having to search for information. Furthermore, the steps might be repeated multiple times: for instance, while developing a potential solution, the developer might search for additional information to help with a solution.

University of Cape Town

4.2.5 Knowledge events

The data recorded ESAOA activities in addition to other types of activities. Many of these other activities were closely related to managing the ESAOA knowledge needed to carry out particular ESAOA activities. These activities were often part of trivial or non-trivial solutions cycles. The term *knowledge event* refers to an activity, performed as part of a solution cycle, which is closely related to the *management* (i.e., finding, creating, storing or sharing) of ESAOA knowledge. A knowledge event can be considered an instance of applying a KM technique to manage the particular knowledge needed to accomplish ESAOA activities (Figure 4.4 illustrates this relationship). As defined in Section 1.1.7, ESAOA activities involve classification and adaptation of ES artefacts. Based on these definitions, some ESAOA activities may also be knowledge events (e.g., instances of artefact adaptation can be knowledge events that store knowledge).

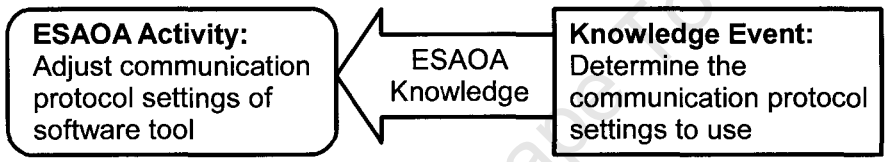


Figure 4.4: Relationship between ESAOA activities and knowledge events

In order to clarify the distinction between ESAOA activities and knowledge events, consider problems number 8 and 17 in Table 4.3. The data show that developers first obtained communications software (a collection of artefacts), and thereafter obtained knowledge about protocol settings in order to use this software. In this case, the knowledge of protocol settings is a form of ESAOA knowledge that was acquired by means of various knowledge events (i.e., reading manuals) in order to complete ESAOA activities (i.e., adapting the settings for the communications software).

4.2.6 Knowledge event types (KETs)

Six main types of knowledge events were identified from the investigation of non-trivial solution cycles discussed in Section 4.2.4. These six knowledge event types (or KETs) are referred to as KET-1 to KET-6. Each KET is elaborated in Table 4.4.

University of Cape Town

Table 4.4: Types of knowledge events.

ID	Knowledge event type	Description
KET-1	Finding information and resources	Obtaining knowledge about where to find information and resources (corresponds to step 2 in Figure 4.3).
KET-2	Understanding information and resources	Obtaining knowledge from information and resources, such as reading and discussing the information found (step 3 in Figure 4.3).
KET-3	Constructing or theorising a potential solution	Obtaining knowledge to plan and construct a solution (step 4 in Figure 4.3).
KET-4	Performing tests	Obtaining knowledge about the planned solution, and determining how to apply the solution (step 5 in Figure 4.3).
KET-5	Interpreting results	Obtaining knowledge concerning the success and limitations of the solution, through a process of acquiring and interpreting the results of testing the solution (steps 6 and 7 in Figure 4.3).
KET-6	Refining development processes and adapting development artefacts	Obtaining knowledge of how to change existing development processes and artefacts to enable use of the solution (step 8 in Figure 4.3).

4.2.7 Data, process and innovation knowledge categories

A common approach used in analysing KM is the identification of different forms of knowledge that exist in the context of the knowledge work studied, as is discussed in Chapter 2 (Section 2.4.5). This approach was followed in the preliminary study in order to identify commonly occurring 'top-level' categories of ESAOA knowledge in Experiment 1. The term 'top-level categories of knowledge' is defined in this study as knowledge forms that are largely distinct from one another in terms of information used and activities performed in order to produce knowledge.

Three top-level knowledge categories were observed in Experiment 1, namely: 1) data knowledge; 2) process knowledge; and 3) innovation knowledge. These categories are inspired in part by Allee's [1997] 'learning and performance framework', and in part by Bloom's taxonomy [Bloom *et al.*, 1964]. The knowledge categories are defined below:

- **Data knowledge** relates to information that predominantly answers "What?" questions, such as: "What are the product requirements and what development tools are available for the selected microprocessor?"
- **Process knowledge** relates to the "How?" questions, for example: "How is the work to be done, and how are the tools used?"
- **Innovation knowledge** is more product-specific, and involves knowledge about which design ideas work. This form of knowledge is used to create parts of the product.

University of Cape Town

These knowledge categories can be viewed as a hierarchy in which innovation knowledge is produced through the application of process knowledge, and process knowledge is in turn acquired by finding and analysing data. These knowledge categories, and the ways in which they relate to one another as a hierarchy are portrayed in the scenario given in Table 4.5 (the scenario is from Project P1-1).

Table 4.5: Scenario for hierarchy of data, process and innovation knowledge.

Hierarchy level	Description of knowledge event	How this knowledge event is categorised
1: Level of data knowledge	Developer searches through manuals and web pages in order to find toolchains that support a given microprocessor.	An example of acquiring *data knowledge*. This knowledge is obtained by interpreting documentation, through actions such as reading web pages, manuals and datasheets.
2: Level of process knowledge	The developer evaluates the toolchains to determine how they are used, and which is best for the application concerned.	An example of acquiring *process knowledge*. Process knowledge is knowledge about how to do development tasks, and is obtained through activities of manipulating design artefacts, testing development techniques, and looking at results.
3: Level of innovation knowledge	Once a development tool is selected and its operation understood, the developer starts to implement embedded software for the microprocessor to test aspects of the software design.	An example of acquiring *innovation knowledge*. Innovation knowledge is acquired through the application of development processes to change design artefacts in attempts to achieve product requirements.

Table 4.6 elaborates on each knowledge category, showing typical learning tasks involved, how the knowledge acquired in these tasks is represented, and the KM tools or systems that are commonly used to manage each knowledge form. The three categories of knowledge are separated for the sake of clarity. In practice, though, development does not follow a simple linear process that begins with the assimilation of data, continues with the perfection of a perfect development process, and ends with innovation. Instead, the knowledge being produced tends to cycle through these categories in an incremental and iterative fashion [Schach, 2005].

Other types of knowledge also relate to development projects, such as contextual knowledge, interpersonal knowledge, strategic knowledge, and others [Allee, 1997]. For the purposes of this study, the focus is on the three knowledge forms listed in Table 4.6, because these were found to be the main types of knowledge used by the developers during knowledge events and ESAOA activities [Winberg, 2005a].

University of Cape Town

Table 4.6: Taxonomy of knowledge for embedded system KM.

Knowledge Type	Type of Learning	KET	Knowledge Produced	KM Tools/Systems
Data: What needs to be done? (Reading datasheets, manuals, and 'how-tos')	Descriptive, comparative, summarising	KET-1, KET-2	Text	Search techniques, files, data bases
Process: How to do the work? (Configuring and using tools)	Analytical, defining, categorising, sorting	KET-3, KET-4, KET-5	Procedures	Compilers, assemblers
Innovation: Which design concepts work?	Synthesising, evaluating	KET-5, KET-6	Creative problem solving	Designs, implementations, decision making

Based on the knowledge types above, knowledge events can be classified according to the principal type of knowledge produced or managed during the event [Winberg & Schach, 2007]. Accordingly, a knowledge event can be referred to as a data knowledge event, a process knowledge event, or an innovation knowledge event. Examples of these types of knowledge event are given below:

1. **Data knowledge event:** e.g., an engineer looks for and then reads a datasheet;
2. **Process knowledge event:** e.g., an engineer constructs and tests a new development method;
3. **Innovation knowledge event:** e.g., an engineer uses process and data knowledge to implement and experiment with a design concept.

4.2.8 Productive and non-productive knowledge categories

Not all the time spent acquiring knowledge necessarily results in knowledge that will be useful to the development of a product [Richter & Abowd, 2004]. This was frequently observed in the context of the activities studied in Experiment 1. For example, although the developers spent a significant proportion of their time performing knowledge events to obtain ESAOA knowledge, there were many situations in which only a small proportion of the knowledge produced in these events was actually used to construct the final product prototype. Similarly, knowledge produced at one point in the project could not always be guaranteed to remain useful for the entire duration of the project. The following scenario, drawn from the Project P1-2 data, substantiates these observations regarding the temporary usefulness of ESAOA knowledge produced:

University of Cape Town

Scenario from Project P1-2

In Project P1-2 (see event 27 in Appendix A.2), the developers spent several days learning how to use a particular set of development tools (or 'toolchain'), namely the *emDebian* [emDebian, 2007] operating system and its toolchain. However, later in the project (event 38 in Appendix A.2), it emerged that much of the time and effort expended in learning *emDebian* had been wasted because the developers decided to change to the *Snapgear* [SnapGear, 2007] toolchain.

In the above scenario, the knowledge acquired while learning to use the toolchain that was later rejected could be considered as 'non-productive knowledge', whereas the knowledge acquired while learning how to use the toolchain that was ultimately used to develop the final project (i.e. *Snapgear*) could be considered as 'productive knowledge'. Accordingly, the concepts of 'non-productive knowledge' and 'productive knowledge' categories were applied in the KMS analysis method as a means of distinguishing between activities that resulted in knowledge that was, or was not, used to build the product. These terms are defined further below.

4.2.8.1 Definition of non-productive and productive knowledge

The term 'non-productive knowledge' refers to the acquisition of knowledge that is not useful to development, whereas 'productive knowledge' refers to the acquisition of knowledge that is useful, in that it is built on and ultimately leads to the construction of the final product.

Definition: productive knowledge

Productive knowledge refers to knowledge, which is obtained from knowledge events in the project of study, and which is used in the construction of the final product for the same project studied.

Definition: non-productive knowledge

Non-productive knowledge refers to knowledge, which is obtained from knowledge events in the project of study, and which is *not* used in the construction of the final product for the same project studied.

Nonetheless, it could be argued that knowing *what not to do* could be as valuable as knowing *what to do*, and that establishing what not to do is sometimes only determined through a failed experiment. Consequently, distinguishing between

University of Cape Town

productive and non-productive knowledge should be considered as a tool for abstracting and analysing KM techniques within the context of individual projects in relation to the final product constructed. It should not be considered a means of evaluating the general usefulness of resources or abilities of developers.

4.2.8.2 Using dead-ends to determine non-productive knowledge

The term 'dead-end' refers to a solution cycle that ended unsuccessfully, in that the developer either could not find useful knowledge to solve a problem, or decided not to use the knowledge or solution developed. For example, in the scenario from P1-2 presented earlier, a dead-end occurred when the developers decided to give up on the first toolchain.

Each of the three knowledge categories described previously were divided into subcategories of productive and non-productive knowledge. Consequently, terms such as 'productive innovation knowledge' (which refers to innovation knowledge that was used to build the final product) are used in this thesis.

4.2.8.3 Backwards tracing to classify knowledge events as productive or non-productive

As described above, a dead-end that terminates an event chain can lead to its predecessor events resulting in non-productive knowledge (i.e., knowledge that is not used in constructing the final product). However, at the time when a developer is working on one of these predecessor events, he or she did not know that the event would lead to a dead-end. For the purposes of this study, knowledge events in an event chain are considered productive unless they result in a dead-end (i.e., the events are always classified retroactively as productive and non-productive).

In order to gain more information on the progression of knowledge events and knowledge production, knowledge events are categorised as either productive or non-productive in relation to the time when a particular knowledge event occurred. Following this approach, the classification of productive and non-productive knowledge classifications changes as more events are performed (i.e., the classification of previous events may change from productive to non-productive). Figure 4.5 illustrates how this history is produced. The figure is divided into five sections, numbered from 1 to 5 starting from the top left. Each section has a label at the top indicating which knowledge event is added. Two running totals are shown at

University of Cape Town

the bottom of each section, viz. one for productive events and another for non-productive events.

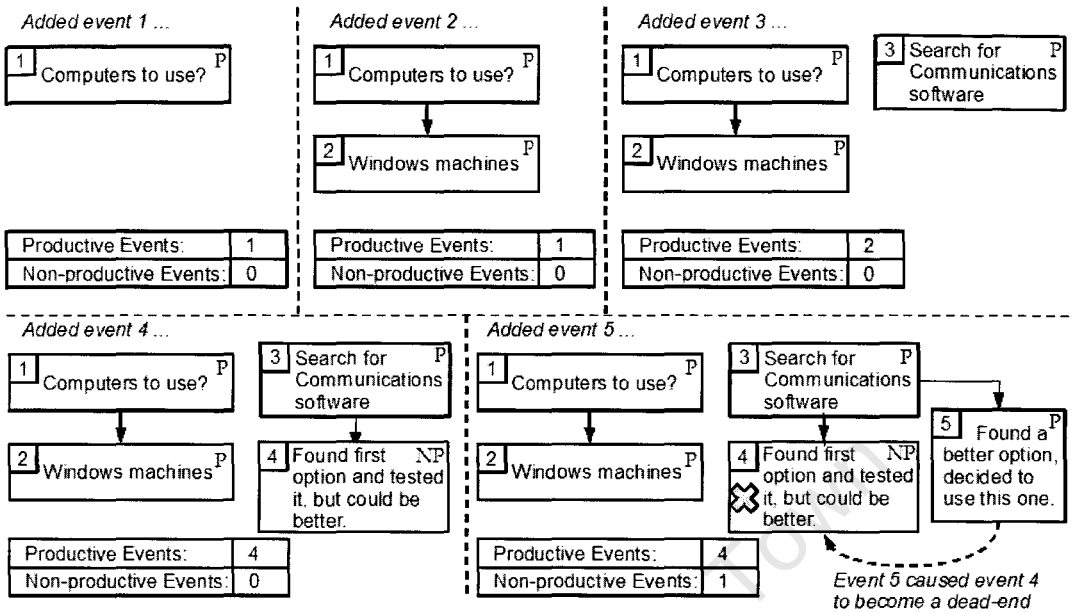


Figure 4.5: Determining non-productive and productive knowledge acquisition.

Each section shows what takes place when a new event is added (this is done in chronological order from the knowledge register); essentially, one of the two operations below is performed:

- If the added event caused a dead-end for the event chain, then all the predecessors of the dead-end event (though *not* necessarily the event that was added) are checked to see which of its predecessors led directly to the dead-end; for each such predecessor that has not been already changed to a non-productive event, the following is done:
    - The predecessor is marked as a non-productive event (in Figure 4.5 this is shown by setting its label to 'NP');
    - The tally of productive events is decremented, and
    - The tally of non-productive events is incremented;
- Otherwise, if the added event did not cause a dead-end, it is marked as a productive event (labelled 'P' in Figure 4.5) and the productive event tally is incremented.

University of Cape Town

4.2.9 Productive time and non-productive time

The terms *productive time* and *non-productive time* refer respectively to time spent acquiring productive knowledge and time spent acquiring non-productive knowledge. These categorisations are defined only within the context of *knowledge events* (as defined in Section 4.2.5), and not for general development tasks.

4.2.10 Knowledge event chains

As defined in Section 4.2.5, a knowledge event is an activity that is closely related to obtaining or managing the ESAOA knowledge needed to complete ESAOA activities. A knowledge event is a step in a *solution cycle;* that is to say, a series of knowledge events that are performed by one or more members of the team in order to obtain a solution, which is then used to carry out ESAOA activities. The *solution cycle* (see Sections 4.2.3 and 4.2.4) describes a high-level generalisation that represents, firstly, the way in which knowledge events are typically performed and, secondly, the common categories of knowledge events observed (e.g., 'finding information', 'understanding resources', and so on). The term *event chain* is used to refer to a sequence of related knowledge events that were physically carried out in a project by one or more of the project team members. An event chain can be viewed as an implementation of a solution cycle that took place during a project.

An event chain is defined as a sequence of knowledge events, which has a particular *start event*, and one or more *end events*. The *start event* concerns the identification of a problem that needs to be solved in order to carry out certain ESAOA activities. Each knowledge event in the event chain has one or more predecessors. Predecessor of a knowledge event may be in the same event chain or in a different event chain. Knowledge events within the same event chain are associated relative to one another; for example, if event A is the predecessor of event B, then it implies that B was performed some time after event A. Each knowledge event can result in zero or more successor events. An *end event has* no successors in the same event chain, although it may have a successor in a different event chain. For an event A, each successor event (referred to as A.1, A.2, etc.) relates to activities carried out to solve the same high-level problem that led to event A. Each successor event in an event chain has one of these two characteristics: (1) the identification of another issue, difficulty or sub-problem related to the same higher-level problem that the event chain concerns; or (2) a solution to a previous issues/difficulty, or a solution to the high-level problem that initiated the event chain.

University of Cape Town

In summary, an event chain can be considered a sequence of ESAOA KM activities, where each knowledge event in the event chain results in one of the following:

- Multiple connected knowledge events (i.e., continuation of the event chain);
- Another issue or sub-problem that needs to be overcome to obtain a solution;
- A solution to either a previous issue or sub-problem observed earlier in the event chain, or a *final solution* to the higher-level problem that began the event chain;
- A *dead-end* (i.e., a knowledge event in which it is determined that the developer decided to abandon a potential solution).

Event chains can be traced in communications between engineers and captured in log book entries, email archives, and in other separate communication events related to the common problem [Winberg, *et. al*, 2008]. Knowledge events are evident in the extract from the data provided in Table 4.3. For example, the problem 1 in Table 4.3 had two knowledge events associated with it; the first involved asking a question, while the second involved determining the solution. This straightforward sequence of events can be viewed as an event chain, which resulted in a *final solution*. A *final solution* is a knowledge event that resolved the higher-level problem that began the event chain.

However, problems can have sub-problems (e.g., problem 17 in Table 4.3 is a sub-problem of problem 8). Accordingly, a new event chain can be started for a sub-problem, if there are many knowledge events related to solving specifically that sub-problem; otherwise, knowledge events related to a sub-problem can be added to the same event chain. The latter approach was used to avoid having many short event chains (e.g., chains of two events in which the problem was found and soon fixed). The following section describes how event chains are visualized using event chain graphs.

4.2.11 Visualizing event chains using event chain graphs

Event chains graphs were used in the preliminary study (and in the subsequent data analysis methods) as a means of visualizing event chains (for a particular sequence of activities that may have been performed by one or more members of the team investigated). This visualization strategy was used to model, construct and understand event chains. These graphs were also use during meetings with participants to clarify sequences of knowledge events, and relations between them.

University of Cape Town

A sample event chains graph, based on Project P1-1 data (see Appendix A), is illustrated in Figure 4.6; a key describing the elements of the graph is provided at the bottom of the figure. As the key shows, the circle on the top left points to the first knowledge event in the project. The arrows indicate successor events. Each knowledge event (the boxes) has two numbers: the event number (on the left), followed by the event chain number (that starts with the letter 'c').

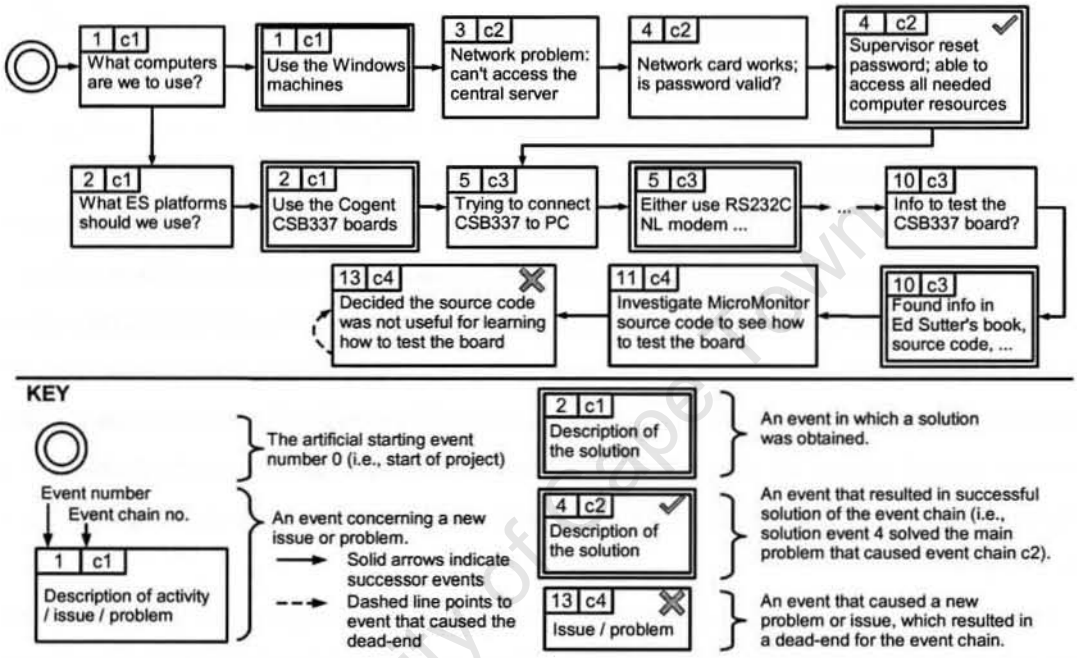


Figure 4.6: Event chains graph.

The arrows show the genealogy of the knowledge events; for example, event 1 is the predecessor of events 2 and 3 (i.e., event 1 occurred before events 2 and 3, and furthermore led to the occurrence of those events).

A knowledge event drawn with a double border indicates a solution, whereas a knowledge event with a single border indicates that another sub-problem or other issue was encountered at that point in the event chain. Knowledge events with the same number may have an associated problem (drawn with a single border) and an associated solution, referred to as a *solution event* (and drawn with a double border); event 1 in Figure 4.6 gives an example of this situation.

A solution marked by a tick symbol (e.g., solution event 4) indicates that a final solution was found for an event chain. A knowledge event marked by a cross (e.g., event 13) indicates a dead-end (i.e., the solution attempt described by the event

University of Pretoria

chain was abandoned); the dashed line indicates which event caused the particular dead-end. An event chain does not have to end with a final solution event or a dead-end; it may simply be a composition of several events that together constitute a workable solution (as is the case for event chain c1 in Figure 4.6).

4.2.12 Development of the KMS analysis strategy

The objective of the preliminary study was to develop a data analysis method that could be applied to data captured in experiments. The produced data analysis method integrates the techniques presented in the preceding sections (Sections 4.2.1 to 4.2.11). The data analysis method is described in Section 3.9, together with an explanation of how the method was optimised for application to Experiment 2 data. The optimised version of the data analysis method was also applied to Experiment 1 data in order to establish a consistent basis for comparison between the experiments.

4.3 Results

This section presents the results of applying the first version of the data analysis method to the data gathered in Experiment 1. As mentioned in Section 4.1, Experiment 1 involved two projects, namely Project P1-1, the Software Signal Generator (or SoSiG), and Project P1-2, the Antenna Controller (or ANTCON).

The following subsections are structured according to the steps of the data analysis method described in Section 3.9.

4.3.1 Results of data synthesis (step 1): Initial knowledge registers

The captured data were synthesised to produce a knowledge register for each project, as described in Section 3.9.2. Appendices A.1 and A.2 provide the final knowledge registers for projects P1-1 and P1-2 respectively. Note that, as explained in Section 3.9.2.2, additional columns were added to the knowledge register in subsequent steps of the analysis process (to avoid redundancy, however, the appendices include only the final knowledge registers).

For Project P1-1, 74 knowledge events in total were identified, separated into 28 event chains. For Project P1-2, a total of 78 knowledge events were found, separated into 45 event chains. The construction of the initial knowledge register for Experiment 1 took approximately 32 hours, and this includes the time involved in annotating the

University of Cape Town

printouts of the data sources, as per the analysis process described in Section 3.9.2. Annotating the printouts accounted for the bulk of this time (around 27 hours).

4.3.2 Results of categorising knowledge events (step 2)

The knowledge events listed in the knowledge register were first categorised, according to the predominant form of knowledge produced, into one of the three categories of data knowledge, process knowledge or innovation knowledge, as explained in Section 3.9.3. An additional column was added to the knowledge register created in step 1 to record the predominant form of knowledge for each event. Table 4.7 summarises these results, indicating the total number of knowledge events for each category (see Appendix A for the complete knowledge registers). This process of classifying the knowledge events of Experiment 1 was accomplished in approximately ten hours.

Table 4.7: Number of knowledge events in each of the data, process and innovation knowledge categories.

Knowledge Category	Project P1-1 no. of events per category	Project P1-2 no. of events per category
Data knowledge	23	35
Process knowledge	31	32
Innovation knowledge	20	10
TOTAL	74	77

As shown in Table 4.7, Project P1-2 involved twelve more knowledge events than Project P1-1 with regard to the acquisition of data knowledge (e.g., reading manuals and websites). The two projects had almost the same number of events with regard to process knowledge (e.g., how to use tools). Project P1-1 had twice the number of knowledge events related to innovation knowledge than Project P1-2.

4.3.3 Results of problem/solution mapping (step 3): Event chains and event chain tables

For each project, the knowledge events in its knowledge register were traversed to establish problem/solution mappings; these were done based on the description of the knowledge events and the recorded predecessors. From these findings, event chains for the project were determined. The event chains for each project were maintained by assigning to each event chain a number (chronologically starting at 1), and a brief description stored in an event chain table (note the event chain table incorporates event chains from all members of the team). The event chain number was placed in the corresponding knowledge event entries in the knowledge register. The column titled 'Evnt Chn' in the knowledge registers, in Appendices A.1 and A.2,

University of Cape Town

shows event chain numbers for the knowledge events for Experiment 1. The event chain tables for Projects P1-1 and P1-2 are shown in Tables 4.8 and 4.9 respectively. Project P1-1 had 28 event chains, whereas Project P1-2 had 45 event chains.

Table 4.8: The event chains for Project P1-1 (SoSiG).

Event Chain	Starting Event No.	Event chain summary
1	1	Identifying embedded platform and lab computers to use
2	3	Connecting to network
3	5	Connecting CSB337 to PC
4	11	Investigating umon source
5	14	Deciding which software tools to use for CSB337
6	18	Using Toolchain: arm-linux
7	21	Identifying method to develop software
8	22	Writing example C program for CSB337
9	24	Learning how to use a Makefile
10	28	Installing program on CSB337
11	32	Assessing why program crashes
12	36	Running program
13	37	Reading user interface spec
14	38	Designing user interface
15	39	Learning how to send bytes
16	40	Learning how to read bytes
17	41	Implementing command processor
18	43	Developing start-up module
19	47	Optimising jump table
20	48	Looking at LED control
21	51	Converting strings to integers
22	54	Designing I2C Interface for actuator board containing DAC
23	58	Connecting up actuator board
24	59	Powering actuator board
25	61	Designing I2C software interface
26	68	Improving timing
27	69	Improving communications
28	73	Preparing final prototype for demo

Table 4.9: The event chains for Project P1-2 (ANTCON).

Event Chain	Starting Event No.	Event chain summary
1	1	Identifying components to use for interface board
2	2	Deciding how to manage files
3	3	Mounting samba
4	4	Connecting up CSB337
5	6	Solving power supply problems
6	8	Identifying suitable software for CSB337
7	9	Learning about MicroMonitor
8	10	Assessing software development options

University of Cape Town

9	11	Looking at Linux options
10	12	Assessing whether to use RTEMS as the O/S
11	15	Studying MicroMonitor training guide
12	16	Reading documentation supplied with MicroMonitor source
13	17	Solving problem of communicating with CSB337 over Ethernet
14	21	Identifying program to use for diagrams
15	22	Constructing interface board
16	23	Learning how to use Arm-Linux
17	27	Selecting operating system and its toolchain
18	30	Acquiring parts parts for interface board
19	32	Fixing network
20	35	Configuring ATFTP
21	36	Executing example application on MicroMonitor
22	37	Fixing linker options
23	38	Compiling Snapgear
24	39	Installing uCliux rpms
25	46	Finding Arm-linux new option
26	47	Compiling arm-linux
27	50	Installing arm-linux
28	51	Booting arm-linux
29	52	Deciding how to use the Ramdisk
30	53	Installing ramdisk
31	54	Testing and optimising ramdisk
32	55	Tackling problems with DC motor interface
33	56	Designing AC-DC circuit
34	58	Examining Busybox
35	60	Using CSB 337 Linux networking
36	62	Using new snapgear toolchain
37	63	Implementing envelope detector method
38	64	Researching ADCs
39	67	Configuring Busybox
40	68	Implementing device drivers
41	72	Programming LED driver
42	73	Determining AT91RM9200 interrupt techniques
43	74	Implementing I2C driver
44	76	Coding antenna control module
45	77	Creating user interface

University of Cape Town

It is clear from Tables 4.8 and 4.9 that there were commonalities between the two projects with regard to their event chains. For example, event chain 4 in Project P1-1 involved learning how to use MicroMonitor, which was similar to event chain 7 in Project P1-2. However, there were also differences between the two sets of event chains. For instance, Project P1-2 developers had more knowledge events relating to the use of embedded Linux than the developers working on Project P1-1 did. The construction of the event chains for Experiment 1 took approximately 16 hours.

4.3.4 Categorising knowledge events according to productive and non-productive knowledge (step 4)

The many event chains that linked knowledge events to dead-ends or to solutions had to be followed in order to determine which events were ultimately non-productive (ending in a dead-end) or productive (contributing towards a solution used in building the final product). This allowed for the isolation of 'success paths', that is, paths that connected the root event to events that did not terminate in a dead-end but in one that was used in developing the final product. In order to determine these 'success paths', and the corresponding productive or non-productive knowledge events in the knowledge register, all dead-ends in the knowledge register had to be found. The predecessors of each of these dead-end events then had to be traced backwards to establish which of its predecessors had resulted in productive or non-productive knowledge, as detailed in Section 4.2.8.3. During this process of 'backwards tracing', the entries in the knowledge register were marked as producing productive or non-productive knowledge. Thereafter, the time spent on each task was separated into productive and non-productive time. The subsections below further describe these processes and the results obtained.

4.3.4.1 Classifying knowledge events as productive or non-productive

During the 'backwards tracing' process, entries in the knowledge register were marked as having resulted in productive or non-productive knowledge. The 'P' and 'NP' columns were added to the knowledge register for this purpose; the 'P' column denoted the acquisition of productive knowledge, whereas the 'NP' column indicated the acquisition of non-productive knowledge.

The categorisation of data, process, and innovation knowledge (as per Section 4.3.3) was maintained. Each knowledge event was thus effectively classified into one of six categories (i.e., productive data knowledge, non-productive data knowledge, productive process knowledge, non-productive process knowledge, and so on). Tables 4.10 and 4.11 show the total number of productive and non-productive knowledge events for the projects, separated into the aforementioned categories of data, process and innovation knowledge.

University of Cape Town

Table 4.10: Number of knowledge events per knowledge category for P1-1.

Knowledge Type: P1-1	Productive knowledge events	Non-productive knowledge events	Total knowledge events
Data Knowledge	5	18	23
Process Knowledge	10	20	30
Innovation Knowledge	8	12	20
Combined	23	50	73

Table 4.11: Number of knowledge events per knowledge category for P1-2.

Knowledge Type: P1-1	Productive knowledge events	Non-productive knowledge events	Total knowledge events
Data Knowledge	12	23	35
Process Knowledge	15	17	32
Innovation Knowledge	6	4	10
Combined	33	44	77

For Project P1-1, 23 knowledge events resulted in productive knowledge, whereas 50 events resulted in non-productive knowledge. In comparison, Project P1-2 had 33 productive and 44 non-productive knowledge events. Although Project P1-2 had three more knowledge events than Project P1-1, Project P1-2 had a higher percentage of events from which productive knowledge was acquired: Project P1-1 had 31% productive events, whereas Project P1-2 had 43%.

It took approximately 12 hours to classify all the Experiment 1 knowledge events as generating either productive or non-productive knowledge.

4.3.4.2 Calculation of non-productive and productive time

The 'time' column in the knowledge registers that was used to record the amount of time spent on each knowledge event, was replaced by two columns: the 'PT' column to record *productive time*, and the 'NT' column to record *non-productive time*. Initially, the 'PT' column was assigned to the values in the previous 'time' column.

While performing the 'backwards tracing' process for classifying events as either productive or non-productive, the 'PT' and 'NT' columns were respectively used to record hours spent acquiring productive and non-productive knowledge. It was not always a simple matter of assigning the 'NT' entry to the 'PT' value (and setting 'PT' to zero) for events found to be non-productive. Instead, the data sources were revisited to establish a closer approximation of how time was divided between productive and non-productive time.

University of Cape Town

Tables 4.12 and 4.13 show the total number of productive and non-productive hours spent on knowledge events. The 'time' column sets out the total time spent on knowledge events per knowledge category, and the '% of total time' column gives this time as a percentage of the total time spent on knowledge events (e.g., in P1-1, 17 hours was spent acquiring data knowledge, which was 29% of the total time spent on all categories of knowledge events).

Table 4.12: Breakdown of knowledge acquisition times for P1-1 (SoSiG).

Knowledge Acquired	Time (h)	% of total time	Productive time (h)	Non-productive time (h)
Data knowledge	17	29%	3	14
Process knowledge	14	24%	4	10
Innovation knowledge	27	47%	14	13
TOTAL	58	100%	21	37

Table 4.13: Breakdown of knowledge acquisition times for P1-2 (ANTCON).

Knowledge Acquired	Total Time (h)	% of total time	Productive time (h)	Non-productive time (h)
Data knowledge	36	40%	13	23
Process knowledge	28	31%	11	17
Innovation knowledge	26	29%	15	11
TOTAL	90	100%	39	51

The results shown in Tables 4.12 and 4.13 indicate that 32 more hours were spent on knowledge events in Project P1-2 than in Project P1-1. Furthermore, there was no correspondence between the percentages of time spent on knowledge events between knowledge categories. However, almost the same amount of time was spent on acquiring innovation knowledge in both projects: 27 hours for Project P1-1 and 26 hours for Project P1-2.

4.3.5 Finalizing the knowledge registers (step 5)

This step involved a final comparison of the data sources and the knowledge register to ensure that there were no omissions or knowledge events lost during the preceding steps. In addition, the predecessors of each knowledge event were checked to ensure that no predecessor links were connecting to an event with a higher number (i.e. one that occurred later), as such mistakes would have represented the data incorrectly and caused the PTHC program to hand during trend analysis.

University of Cape Town

4.4 Trend analysis and graphing

The first version of the data analysis method involved determining the amount of *productive time* and *non-productive time* spent on each knowledge event, as per the procedure described in Section 3.9.5. The results of applying this procedure are given in Section 4.3.4 (the results in that section are based on all knowledge events being categorised in relation to the final product).

In order to visualize trends in knowledge acquisition during the development process, the analysis process of classifying events as either productive or non-productive was applied incrementally, starting with event 0 and adding one event entry at a time, while maintaining a history of the changes in total non-productive and productive time as described in 4.2.8.3. The results presented in Section 4.3.4 show only the final totals for these times, whereas the graphs for trend analysis show the fluctuations in these times after each knowledge event has been completed. A program, referred to as the *Partitioned Time History Calculator* (or PTHC), was developed to perform this analysis procedure automatically.

The PTHC program loads the spreadsheet that represents a knowledge register, and converts it into an internal graph representation, where each node of the graph refers to a knowledge event, and each edge to a predecessor link. No pre-processing is performed on the input prior to constructing the graph.

Each node *N[i]* of the graph contains the following fields: *k_type*, *p_time*, *np_time* and *marker*. These fields respectively maintain the following: 1) the predominant type of knowledge produced in the knowledge event (i.e., data, process or innovation); 2) the productive time in hours, 3) the non-productive time in hours, and 4) a marker value. Initially, all *marker* fields of the nodes are marked 'excluded' and an output file is created. Each node *N[i]* of the graph is then traversed in order from the start node, with the following four steps being performed for each node:

1. If node *N[i]* is a dead-end, its marker is set to 'non-productive', and the following recursive method is carried out:
    - Each predecessor node *P[j]* of node *N[i]* not marked as 'excluded' and having no predecessors in the graph marked 'productive' or 'excluded', is marked as 'non-productive' and its *np_time* field is set to the sum *p_time* + *np_time,* where *p_time* is set to 0. This same

University of Cape Town

2. If node N[i] is not a dead-end, then:
    - If *p_time* field of N[i] is greater than 0, it is marked as 'productive', otherwise it is marked 'non-productive'.
3. The following computation is done:
    - Six accumulator variables are set to 0; these are named: *sum_pd*, *sum_pp*, *sum_pi*, *sum_npd*, *sum_npp*, and *sum_npi*. These accumulators respectively track time spent acquiring: productive data, productive process, productive innovation, non-productive data, non-productive process and non-productive innovation knowledge.
    - The entire graph is traversed from the start node. For each node *X[i]* in the graph that is not marked 'excluded', the following is done:
        - If field *k_type* of *X[i]* is set to 'data knowledge', then the value of field *p_time* is added to *sum_pd*, and the value of field *np_time* is added to *sum_npd*.
        - If field *k_type* of *X[i]* is set to 'process knowledge', then the value of field *p_time* is added to *sum_pp*, and the value of field *np_time* is added to *sum_npp*.
        - If field *k_type* of *X[i]* is set to 'innovation knowledge', then the value of field *p_time* is added to *sum_pi*, and the value of field *np_time* is added to *sum_npi*.
4. The values of the accumulators are added to the output file.

If the knowledge register input to the program has *n* knowledge events, then the output file of the program comprises *n* rows, with each row comprising six values that indicate the totals for productive time and non-productive time divided according to the categories of data, process and innovation knowledge. The time complexity of this algorithm is $O(n^3)$, considering that, in the worst case, each iteration of the first step could involve $O(n^2)$ operations.

Output from the PTHC was used to produce *productivity graphs*, which are graphs that represent changes in productive versus non-productive time over the sequence of knowledge events for the project. The output provided by the program allowed the results to be separated into the data, process and innovation knowledge categories, which enabled separate graphs to be plotted for each knowledge category.

University of Cape Town

The sections that follow provide results for the trend analysis process. The results are divided according to the two projects of Experiment 1: Project P1-1 results are reviewed in Section 4.4.1, and Project P1-2 results are presented in Section 4.4.2. Section 4.4.3 discusses commonalities of the results, which leads into a revision of the data analysis methods for Experiment 2 and the construction of ESAOA KMS version 1.

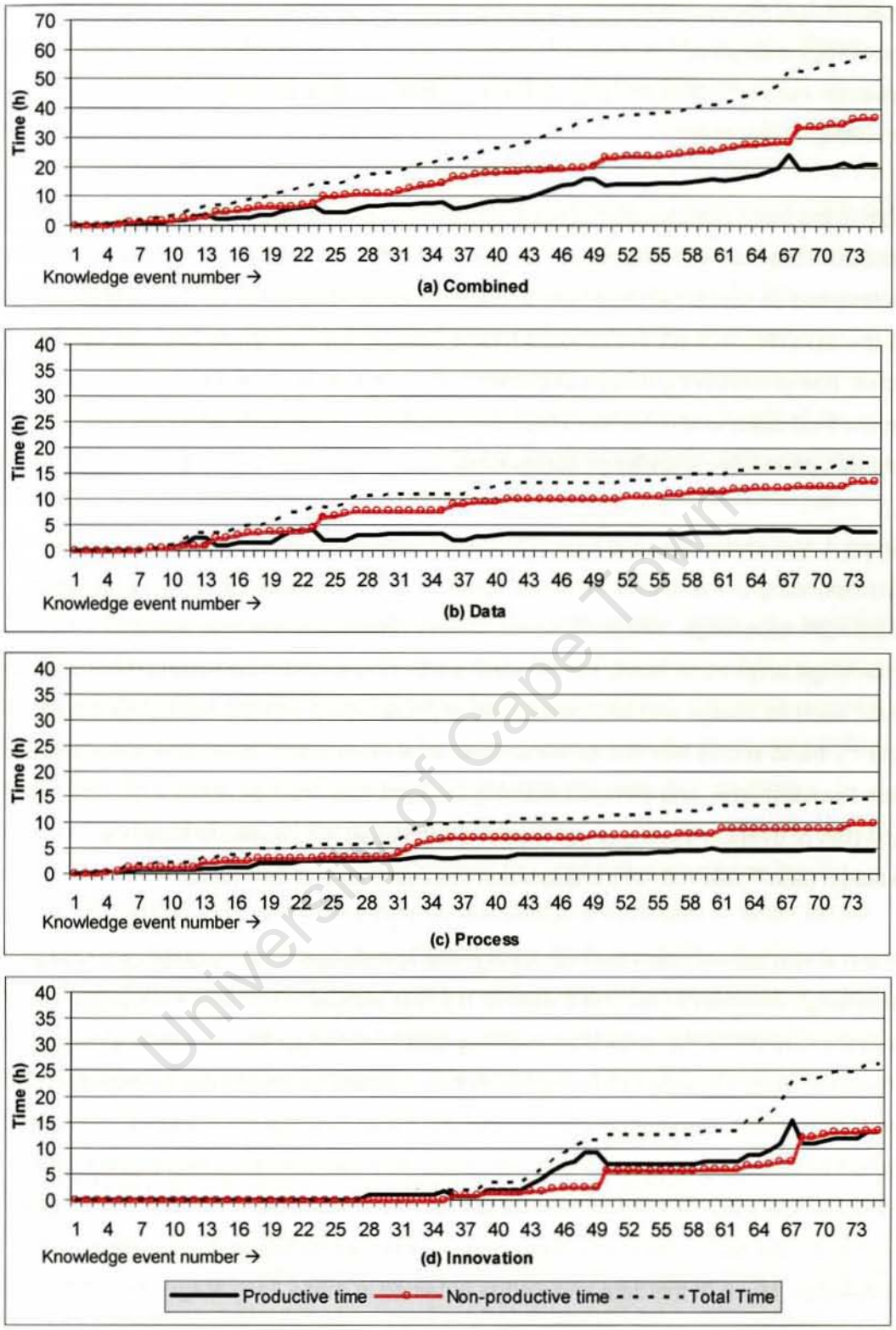
4.4.1 Results of P1-1 (SoSiG)

The productivity graphs and summary tables for Project P1-1 are separated into the two subsections that follow.

4.4.1.1 Productivity graphs

The productivity graphs for P1-1 (SoSiG) are shown in Graph 4.1. The *x*-axis for each graph shows a progression of knowledge acquisition events in chronological order. The *x*-axis labels are the knowledge event numbers. The *y*-axis shows the accumulating sum of hours spent on these knowledge production tasks, from event 1 to the n^{th} event (corresponding to the last event on the *x*-axis). Note that the graph is essentially showing the integral of time with respect to the learning event, which means that the last events are not necessarily taking considerably longer than earlier events.

University of Cape Town



Graph 4.1: Productivity graphs for Project P1-1 (SoSiG) showing accumulation of productive and non-productive times.

University of Cape Town

Graph 4.1(a) shows the sum of productive time versus non-productive time for all knowledge categories combined. Graphs 4.1(b), 4.1(c), and 4.1(d) respectively separate out productive and non-productive time for data, process, and innovation knowledge categories.

Dips in the trend lines for productive time and non-productive time can occur because time that was originally classified as productive in one event can become reclassified as non-productive time because of a later event, which is accounted for by the algorithm of the PTHC program (see Section 4.4). Similarly, the reverse can occur: non-productive time can become productive time when a previously determined dead-end is revisited (see Section 4.2.8 for more detail on the productive and non-productive knowledge categories).

Graph 4.1(a) shows the total time (TT) trend line ending at the value 58, corresponding to the total of 58 hours spent in knowledge events for the acquisition of ESAOA knowledge. These 58 hours of work reflect only the time spent in knowledge acquisition tasks; the remaining effort involved in the development project itself, such as design and documentation, is not factored into this total of 58 hours. The TT trend shows that few of these tasks took more than 1 hour. Event number 66 took the most time, and involved learning to implement an interrupt routine; this event was part of event chain 25, which involved solving an 'I2C software interface' problem (see Table 4.8, which describes the event chains for Project P1-1).

Graph 4.1(b) shows time spent acquiring data knowledge. Little acquisition of data knowledge occurred in the first 6 events; the first acquisition of data knowledge was in event 7. In these first events concerning data knowledge, the developers read about particular Linux-based tools, which they needed for testing the embedded evaluation board. Between events 10 and 26, the knowledge produced in the events shifted between the productive and non-productive categories. During these events, the developers learned about the hardware platform, chose development tools, and learned to use tools; these activities primarily involved finding and assimilating data knowledge. Much of the data knowledge obtained in these early stages was not used, or led to dead-ends, which accounts for the shifts of time between the productive and non-productive categories. For example, the developers spent time reading about certain tools (events 11 and 13) only to disregard them later.

University of Cape Town

Graph 4.1(c) shows time spent acquiring process knowledge. Productive time spent formulating process knowledge increased gradually throughout the project. These events involved testing development steps (e.g., changes in command-line parameters in event 50). Acquiring knowledge of new development processes (i.e., revising steps used in development) was accomplished speedily, as reflected by the average event duration of 10 minutes in the TT trend. However, many of the changes were reversed shortly afterwards, either in favour of a better approach, or because the new method interfered with an existing constraint. This accounts for the comparatively large increases in non-productive knowledge associated with comparatively small increases in productive knowledge. The most noticeable case, comprising events 31 to 36, involved determining how to compile and upload an experimental program to the embedded platform. Multiple attempts were made to solve this problem (an average of 1.3 hours for each event), but only one of the attempts was deemed successful.

Graph 4.1(d) shows time spent acquiring innovation knowledge, which only started at event 30. This first event involved writing an experimental program to test the design of a communications protocol. The TT trend line for innovation is flat until it reaches this point, indicating that the developer first had to learn about the hardware platform and tools before being in a position to write experimental code that could be executed on the platform. Of the four graphs in Graph 4.1, only the innovation graph shows the non-productive time trend line below the productive time trend line for most of the project. The productive time for this category increases in larger jumps than the other categories. These characteristics are likely due to the developers having formulated an effective development process with which design ideas could be efficiently tested and then refined. Many of these tests were deemed successful; therefore, the time spent performing these experiments remained classified as productive time.

University of Cape Town

4.4.1.2 Productive and non-productive time summary tables

Table 4.14 is a summary table for Project P1-1. The table shows that the developers spent a total of 17 hours (29 percent of the total 58 hours) acquiring data knowledge, 14 hours (24 percent) acquiring process knowledge, and 27 hours (47 percent) acquiring innovation knowledge. Significantly more time was involved in acquiring innovation knowledge than either data or process knowledge. Of the total 58 hours spent on knowledge acquisition, there were 14 hours of non-productive data time, 10 hours of non-productive process time, and 13 hours of non-productive innovation

time, amounting to 37 non-productive hours in total (64 percent of 58 hours). In total, there were 21 productive hours (36 percent of 58 hours), showing that more time was spent obtaining non-productive knowledge than productive knowledge.

Table 4.14: Breakdown of knowledge acquisition times for P1-1 (SoSiG).

Knowledge Acquired	Total Time (h)	% Project Time	Productive Time (h)	Non-productive Time (h)
Data knowledge	17	29%	3	14
Process knowledge	14	24%	4	10
Innovation knowledge	27	47%	14	13
TOTAL	58	100%	21	37

4.4.2 Results of P1-2 (ANTCON)

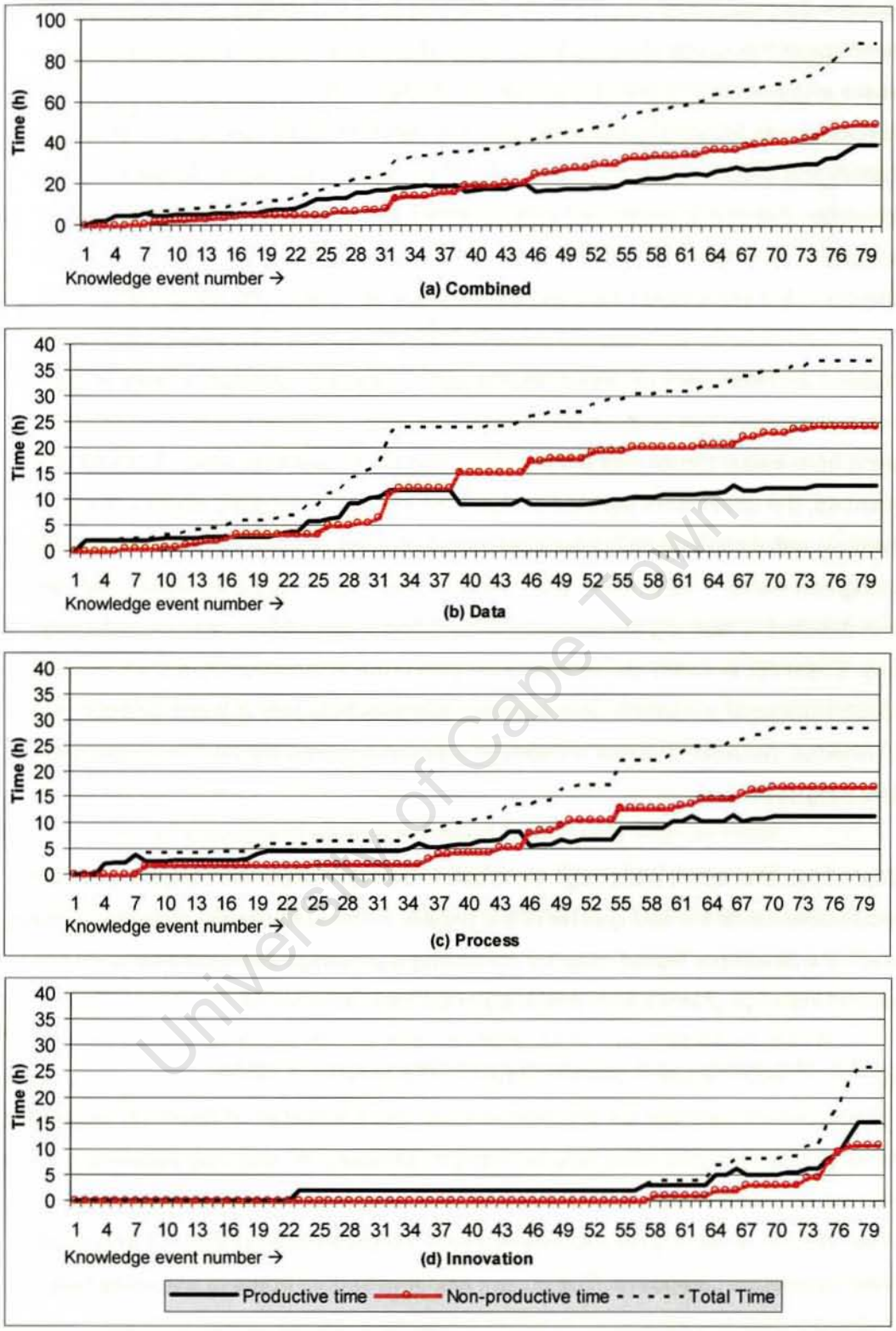
The productivity graphs and summary tables for Project P1-2 are separated into the two subsections that follow.

4.4.2.1 Productivity graphs

The productivity graphs for Project P1-2 are shown in Graph 4.2. As in the corresponding graphs for SoSiG, Graph 4.2(a) shows changes in productive time versus non-productive time for all knowledge types combined, and Graphs 4.2(b), (c), and (d) respectively separate out productive and non-productive time for the data, process, and innovation knowledge categories.

The TT trend in Graph 4.2(a), the combined graph for P1-2, has a different shape to the one for P1-1. In the P1-1 graph, there is one significant step around event 65; but for the P1-2 graph, there are multiple steps, in particular around events 32, 53, and 75. This is due to the P1-2 developers taking a longer time to complete some of these tasks. In the P1-1 graph, the PT trend remained below the NT trend from event four; but in the P1-2 (ANTCON) graph, the PT trend remained above the NT trend from event 38 – showing that the total amount of productive time for P1-2 outweighed the total non-productive time for almost the entire first half of the project. The graph of Graph 4.2(a) also shows that a total of 90 hours were spent on knowledge events, which is 32 hours more than P1-1.

University of Cape Town



Graph 4.2: Productivity graphs for Project P1-2 (ANTCON) showing accumulation of productive and non-productive times.

University of Cape Town

Graph 4.2(b) illustrates the trends for acquiring data knowledge. The productive time trend line of this graph shows a focused effort spent between events 23 and 32, where an increase of 8 productive time hours occurred. For these events, the developer was focused primarily on learning about an embedded Linux operating system and searching for solutions to TFTP networking problems. At event 38, the developer decided to reject the first embedded operating system, resulting in productive time spent on the first operating system becoming non-productive time (the dip in the productive time trend line at event 38 reflects this decision).

Graph 4.2(c) illustrates the trends in acquiring process knowledge. Based on the graph, it appears that process knowledge was acquired in two parts: the first part going from event 1 to 44, while the second part continued from event 45 to 78. In event 45, the developers decided to reject the second embedded operating system together with its associated cross-compiling toolchain, which had been installed and configured earlier – as a result, both the time spent learning the toolchain and the time devoted to learning the embedded operating system became non-productive time. Event 38, in which the developers rejected the first operating system, caused a transformation of productive time to non-productive time only in terms of data knowledge, because process knowledge was not acquired for the first embedded operating system.

Most of the time spent acquiring innovation knowledge, shown in Graph 4.2(d), was accumulated over the last quarter of the project, with the exception of event 22 during which the developer tested code for controlling a prototype interface board. The second instance of innovation only happened later, at event 56.

University of Cape Town

4.4.2.2 Productive and non-productive time summary tables

Table 4.15 indicates that the P1-2 developers spent a total of 36 hours (40 percent of the total 90 hours) acquiring data knowledge, 28 hours (31 percent) acquiring process knowledge, and 26 hours (29 percent) acquiring innovation knowledge. Thus, in P1-2, acquiring data knowledge took the greatest proportion of time of all three knowledge categories. The project also involved 23 hours of acquiring non-productive data knowledge, 17 hours of acquiring non-productive process knowledge, and 11 hours of acquiring non-productive innovation knowledge. Accordingly, the knowledge events collectively consumed 39 productive hours (43 percent of the total 90 hours), compared to the total of 51 non-productive hours (57 percent of the total time).

Table 4.15: Breakdown of knowledge acquisition times for P1-2 (ANTCON).

Knowledge Acquired	Total Time (h)	% Project Time	Productive Time (h)	Non-productive Time (h)
Data knowledge	36	40%	13	23
Process knowledge	28	31%	11	17
Innovation knowledge	26	29%	15	11
TOTAL	90	100%	39	51

4.4.3 Synopsis of Experiment 1 results

The overall results for the Experiment 1 projects show that, for both projects:

1) Productive time accounted for less than half the total time spent in learning tasks for each case study;
2) Each hour of productive time was generally achieved after multiple hours of non-productive time; and
3) The ratios between productive and non-productive time were neither consistent between knowledge categories, nor stable over time. The ratios of time spent on the different learning tasks for the three categories of productive time, viz. data, process, and innovation, were 3:4:14 for Project P1-1, and 13:11:17 for Project P1-2.
4) Production of innovation knowledge tended to start only after an initial accumulation of data and process knowledge; in Project P1-1 innovation started about halfway through the project, while in Project P1-2 innovation started only in the last quarter of the project.

Acquisition of innovation knowledge was fundamental to the completion of both projects, but the acquisition of this knowledge started at a late stage in both projects. Time spent acquiring productive innovation knowledge accounted for a relatively small part of the time spent in learning tasks, viz. less than a quarter of the time in both cases: 14 of 58 hours for Project P1-1, and 17 of 90 hours for Project P1-2.

The amount of time spent acquiring data and innovation knowledge differed between projects. However, almost the same amount of time was spent acquiring innovation knowledge in both projects; 27 hours in Project P1-1 and 28 hours in Project P1-2. Of the time spent obtaining process knowledge, 24% of the Project P1-1 time was productive, whereas 31% the Project P1-2 time was productive. This result suggests that a team of two novices needs to spend a certain minimum time acquiring process

University of Cape Town

knowledge to achieve effective development strategies for the hardware platform prescribed.

In both projects, the acquisition of data knowledge was often followed by the acquisition of either process knowledge or innovation knowledge, or by a dead-end. For example, in event chain 6 there were three knowledge events concerning data knowledge followed by one more event in which process knowledge was acquired. In Project P1-1, there were relatively few event chains in which only data knowledge was acquired (these were event chains 8, 13, 15 and 16). Project P1-2 had a larger number of event chains (a total of 13) in which only data knowledge was obtained; but, unlike Project P1-1, most of these event chains comprised few (one or two) knowledge events, many of which resulted in non-productive knowledge. This finding suggests that P1-2 developers may have lost their focus more often than P1-1 developers did (e.g., reading up on information that was of little use to the project or not of immediate use to solving the problems they encountered).

These results of analysing the data obtained from Experiment 1, and the artefacts produced by the developers working on the projects, were used in constructing ESAOA KMS version 1. The next section (Section 4.5) describes the modifications that were made to the framework analysis strategy, and it is followed (in Section 4.6) by a description of ESAOA KMS version 1 that was later tested in Experiment 2.

4.5 Design of the second iteration of framework analysis

Modifications were made to the initial data capture, data analysis, and trend analysis methods that had been established during the preliminary study and applied to the Experiment 1 data. These modifications are a response to insights gained while obtaining the initial Experiment 1 results. More specifically, the modified analysis methods were applied to the Experiment 1 data, whereas the modified data capture and analysis methods were applied to Experiment 2. Section 4.5.1 details the changes made to the data capture procedures, while Section 4.5.2 describes the changes made to the data analysis methods.

University of Cape Town

4.5.1 Refinements to data capture methods for Experiment 2

The main difficulties encountered in data capture for the first experiment were:

(1) Much of the data were not useful in identifying ESAOA KM activities;
(2) The large amount of time demanded from both researcher and ES developers in terms of capturing data; and
(3) Duplication (the same information was represented multiple times in the data).

Each point above is elaborated on in the subsections below.

4.5.1.1 Focusing on the knowledge-rich data sources

Developer logs, code reviews and email archives were found to be the most useful (and 'information rich' [Kitamura *et al.*, 2006]) data sources because these data were created close to the time at which the ESAOA activities occurred. Email was particularly useful, as it automatically recorded time and thread information (i.e., replies to questions). Consequently, a group forum was planned for Experiment 2 as similar features are available with forums postings.

Analysing logs and email archives was easier and faster than analysing the minutes of meetings. Minutes were in many cases handwritten, some (generally those recoded by the novice engineers) containing cryptic short-hand, all of which took time to decipher. Data obtained from both meeting minutes and design and code review meetings resulted in identifying few additional knowledge events from those identified in the other data sources. Code reviews, which entailed looking through code produced by the developers, were beneficial to tracking solutions (or confirming which solutions were not applied to the final product, thus establishing dead-ends). However, looking at the designs (e.g., schematics and block diagrams) generally provided no insight into ESAOA activities.

During meetings, the developers seldom remembered past learning tasks accurately and frequently used their logs and project artefacts to jog their memory. For these reasons, minutes from meetings and product demonstrations were not used as primary data sources for Experiment 2.

Although product demonstrations in Experiment 1 were found to be of little benefit to data collection, demonstrations were retained in Experiment 2 as they were used to evaluating final products and the quality of the knowledge produced by the teams.

University of Cape Town

Interviews with the developers were necessary to refine the interdependency between ESAOA activities, thus providing insights not easily obtained from the other data sources.

4.5.1.2 Changing the unit of analysis to event chains

The developers found it faster and more beneficial to their work, to log keywords describing a problem, and to give brief descriptions of solution attempts, related to ESAOA activities and associated knowledge-seeking tasks. They found it too laborious to capture detailed descriptions of their activities as originally requested of them. Thus, a log tended to become a map of problems and corresponding solution attempts. This structure was similar to that of email conversations. A similar structure was observed in event chains (which were used to determine dead-ends and non-productive knowledge events). For this reason, a coarser level of 'granularity' [Nyerges *et al.*, 2002] was used for the data analysis phase, by making use of event chains as a unit of analysis instead of independent knowledge events. This change was expected to speed-up the data analysis method without significantly changing the overall results of the analysis process (Section 4.5.3 compares results of the old and new analysis methods).

Experiment 1 contained two projects, while 13 projects were planned for Experiment 2. In addition, with regard to the duration of projects, the first experiment projects took approximately three months to complete, while those of the second experiment were planned for the duration of eight months. This caused additional concern with regard to the large amount of data that would be generated across the thirteen projects over the longer period. For Experiment 1, the developers were requested to record how long each of the knowledge-seeking tasks took. However, the developers frequently neglected to complete these time entries, and often did not review previous entries to record whether they had been completed – the developers complained that this need to track their time and review entries took too long and tended to diverted them from their main priority of completing their project. This resulted in significant amounts of time spent filling in missing details, which, if the same problem occurred in Experiment 2, would have made the analysis procedure excessively lengthy. Consequently, the decision to maintain timing records for each knowledge events was abandoned in Experiment 2. Event chains used in the revised analysis method were thus associated with a count of knowledge events, instead of with hours of productive or non-productive time.

University of Cape Town

4.5.1.3 Data capture supporting event chains

These changes led to the coarser, but less time-consuming, strategy for analysing knowledge events in terms of event chains and *knowledge occurrences* (see Section 4.5.2.1) per event chain. Based on the changes discussed above, the data capture method for use in Experiment 2 focussed on:

(1) Developer logs that followed the problem/solution mapping schema;
(2) Email and online forum correspondence archives; and
(3) Code reviews

These changes supported representation of the process of ESAOA knowledge acquisition and production in terms of event chains. This necessitated a change to the data analysis method.

4.5.2 Changes to the analysis methods

4.5.2.1 Refinements of data synthesis methods – using knowledge occurrence

The data from Experiment 1 included the number of hours spent in each knowledge event. As argued in Section 4.5.1, gathering data on the time spent for each knowledge event involved a significant effort, and the data capture method was consequently streamlined for Experiment 2, focusing on problem/solution mapping and maintaining the number of knowledge events per event chain.

The knowledge register was modified to systematise data according to *knowledge occurrences* and event chains. The term *knowledge occurrence* is defined as the existence of a knowledge event; a knowledge occurrence is thus a further abstraction of a knowledge event, indicating the existence of an event without providing specific details about what the event involved. Knowledge occurrences provided a quantitative technique to associate each event chain with the number of knowledge events it comprised per knowledge category – event chains were thus measured according to knowledge occurrences instead of hours.

The classification of knowledge occurrences into the categories of productive and non-productive knowledge were all done in respect to the final product, in other words, once the last entry had been added to the knowledge register. This eliminated the laborious *back tracing* process done for the earlier version of the trend analysis method (in the earlier version of trend analysis, each time a knowledge events was

University of Cape Town

added to the graph, all previous knowledge events were back traced to determine if they had become non-productive in respect to the version of the product at that time).

In the revised data analysis method, a brief description of each event chain was entered into an event chain table for the project concerned. The rows of the knowledge register maintained information for each knowledge occurrence, the event chain of which it was a part, and the main category of knowledge that was produced during the event. Each knowledge occurrence entry was added into the knowledge record chronologically in the order of the event chain (i.e., the knowledge register grouped events into event chains, and sorting these by event chain number, and then by event number). Keywords indicating artefacts and tools used during the association knowledge events were also added to the knowledge register, as this information was expected to be potentially useful when refining the KMS.

Table 4.16 shows how the event chain table (see Table 4.8) and knowledge register (Appendix A.1) for Project P1-1, produced by the earlier analysis method, was converted into the new version of the knowledge register according to the refined data analysis method. The table has twelve columns. The first four columns display: the event number (where 1 corresponds to the first knowledge occurrence for the first event chain); the data source type (e.g., 'M' for meeting, 'L' for log); the project number; and the event chain number. The 'KD', 'KP', 'KI', 'PK' and 'NPK' columns are used to represent the category of knowledge produced, respectively: data knowledge, process knowledge, innovation knowledge, productive knowledge or non-productive knowledge. Each of these columns either has the value '1', indicating that most of the knowledge produced in the event was in that knowledge category, or it is blank, to indicate that little knowledge produced in the event fell into that category. The last three columns were used to record: names of tools, names of artefacts or components, and other comments relating to the knowledge event.

University of Cape Town

Table 4.16: Excerpt from knowledge register for Project P1-1 (SoSiG).

Event No.	Type	Project	Event Chain	KD	KP	KI	PK	NPK	Artefacts: Tools	Artefacts: components	Comments
1	M	1	1		1		1				
2	M	1	1								
...	...	...	...	...	...	...	...	...	...	...	...
11	L	1	4	1			1			umon source, umon manual	Download umon source, too much reading to get through.

4.5.2.2 Refinements to graphing methods – knowledge occurrence graphs

As explained above, knowledge registers were changed to maintain knowledge occurrences per event chain. Trend analysis was consequently changed to perform graphing of knowledge occurrences instead of productive and non-productive time. For each of the 13 projects in Experiment 2, the knowledge register was used to plot knowledge occurrence graphs, which show trends of productive or non-productive knowledge production, for event chains, separated into the data, process and innovation knowledge categories.

In Section 4.5.3, the list of events and event chains produced in this experiment are used to produce knowledge occurrence graphs for the Experiment 1 projects; these graphs are then used to compare and contrast the results of the initial data analysis method with those using the revised method.

4.5.3 Establishing a basis for comparison between experiments using knowledge occurrences

The original graphs for Projects P1-1 and P1-2 represented time as either productive or non-productive relative to the state of the prototype at that particular event. As detailed above, this data analysis method was revised to use knowledge occurrences (explained in Section 4.5.2.1). The knowledge occurrences were now classified as productive and non-productive knowledge in reference to the final prototype (instead of in reference to the state of the product at the time the knowledge event occurred).

The strategy described in Section 4.5.2 was applied to the data captured for projects P1-1 and P1-2 to produce a list of knowledge occurrences and then knowledge occurrence graphs for the projects. Sections 4.5.3.1 and 4.3.5.2 present the results and compare them with those obtained using the earlier analysis method. These results provide an accurate basis for comparing the results of the two experiments.

4.5.3.1 Knowledge occurrence tables and graphs for P1-1 (SoSiG)

For Project P1-1, there were 74 knowledge occurrences, i.e., one knowledge occurrence for each knowledge event (as per the analysis of data performed in Section 4.3.1). This total was divided into 23 data knowledge occurrences, 31 process knowledge occurrences and 20 innovation knowledge occurrences (see Table 4.17). A percentage breakdown in respect of data, process and innovation knowledge is shown in Table 4.18 (calculated from the statistics in Table 4.17).

University of Cape Town

Table 4.17: Knowledge occurrences per knowledge type for P1-1.

Knowledge Type: P1-1	PK	NPK	Tot
Data knowledge	5	18	23
Process knowledge	10	20	30
Innovation knowledge	8	12	20
Combined	23	50	73

Table 4.18: Productive and non-productive knowledge within knowledge types for P1-1.

Knowledge Type: P1-1	PK	NPK	Tot
Data knowledge	22%	78%	100%
Process knowledge	33%	67%	100%
Innovation knowledge	40%	60%	100%
Combined	32%	68%	100%

Based on the results shown in Table 4.10 (which used the earlier analysis method), percentages for productive and non-productive knowledge were determined based respectively on the amount of time spent on productive and non-productive tasks. This percentage breakdown is reproduced in Table 4.19.

Table 4.19: Productive and non-productive time percentages for P1-1.

Knowledge Type: P1-1	Productive Time (h)	Non-productive Time (h)	% Productive Time	% Non-productive Time	Total
Data knowledge	3	14	17 %	82 %	100 %
Process knowledge	4	10	29 %	71 %	100 %
Innovation knowledge	14	13	52 %	48 %	100 %
TOTAL	21	37	36 %	64 %	100 %

In order to compare the knowledge occurrence strategy with the productive and non-productive time method, absolute differences between the percentage breakdowns for the knowledge categories were calculated. Table 4.20 shows the differences between the percentage breakdown results of applying the two strategies.

Table 4.20: Differences of results between analysis methods for Project P1-1.

Knowledge Type: P1-1	ΔPK	ΔNPK
Data knowledge	5%	4%
Process knowledge	4%	4%
Innovation knowledge	12%	12%
Average of above	7.0%	6.7%
Combined	4%	4%

The average change in the percentage breakdown for the categories was 7%. Despite the change in the vertical and horizontal axes, and the associated disparity between knowledge occurrences and productive / non-productive time, there are similarities in the percentage breakdowns for productive and non-productive knowledge. The differences in percentages for productive and non-productive data

University of Cape Town

and process knowledge were between 4% and 5%. In terms of innovation, the difference was 12% for both productive and non-productive knowledge. Since the two methods produced very similar results, it was decided that the knowledge occurrence strategy (and its significant savings in terms of effort for analysing data) would be applied in Experiment 2.

The percentage breakdown of productive and non-productive knowledge occurrences in relation to the total number of knowledge occurrences is shown in Table 4.21. These statistics are used in Chapter 5 to determine the effect that the ESAOA KMS has on projects in Experiment 2 in terms of the proportions of knowledge produced in each category in comparison to the Experiment 1 results.

Table 4.21: Proportions of data, process and innovation knowledge for P1-1.

Knowledge Type: P1-1	PK/Total	NPK/Total	Total
Data knowledge	7%	25%	32%
Process knowledge	14%	27%	41%
Innovation knowledge	11%	16%	27%
TOTALS	32%	68%	100%

The knowledge occurrence graph for Project P1-1 is illustrated in Graph 4.3. The vertical axis of the graph refers to knowledge occurrences. Unlike Graph 4.1 (the productivity graph for Project P1-1, which had its vertical axis as time), Graph 4.3 shows only increases and plateaus. It does not show downward movement. Downwards movement was seen in Graph 4.1 and was caused by an event that resulted in a dead-end, thereby causing terminated predecessor events in the event chain (i.e., events which had only non-productive successors) to be reclassified as non-productive. This difference between the two graphs is a result of knowledge occurrences in the revised data analysis method being classified only once the project had been completed; moreover, the knowledge occurrences were classified in relation to the final prototype.

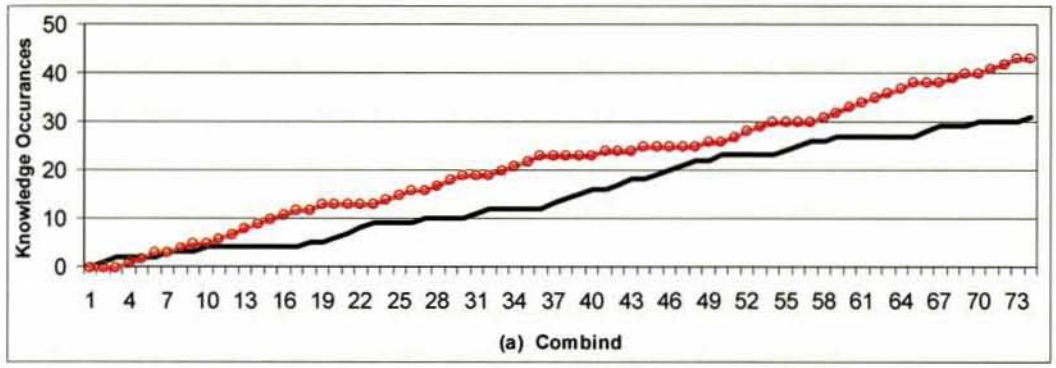
The horizontal axis for Graph 4.3 is the event chain (whereas the horizontal axis for Graph 4.1 is the event number). Graph 4.3 is therefore organised by event chains rather than chronologically or by knowledge events. For example, event chain 1 typically has to do with role allocation (according to the Experiment 2 results). If the team members continue to generate productive or non-productive knowledge with regard to roles half-way (related to issues in event chain 1) through the project, this is still added as a knowledge occurrence in event chain 1. This method results in

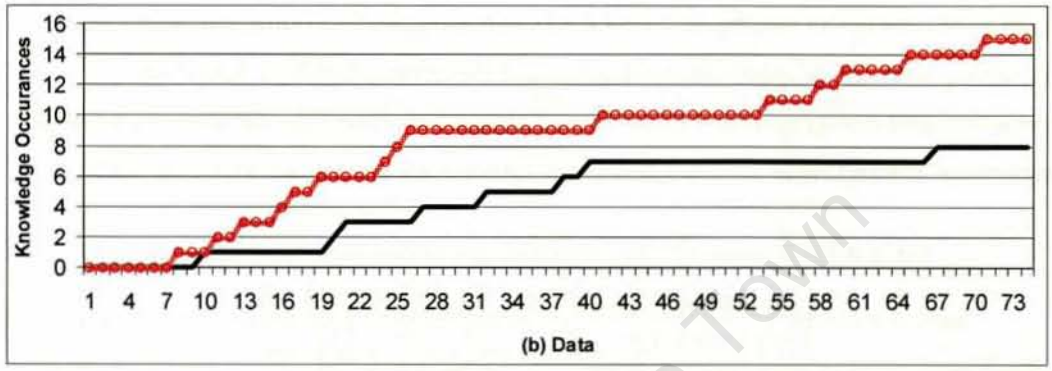
University of Cape Town

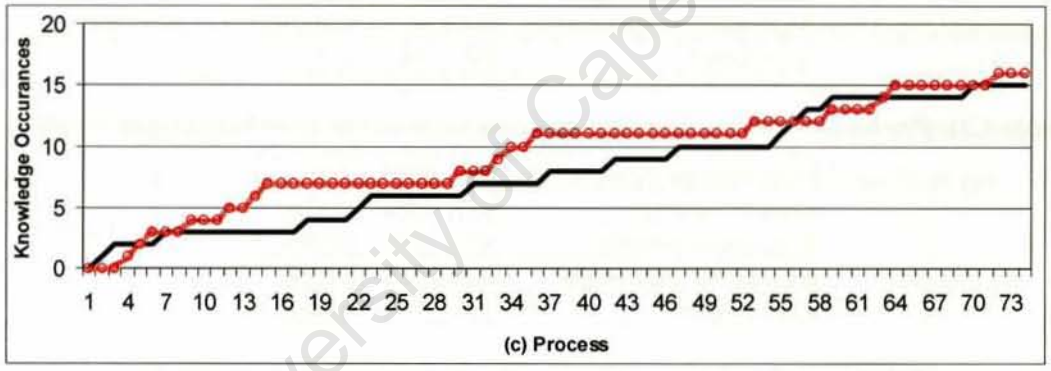
stepped increases, as is visible in Graph 4.3. Generally, however, event chains are in chronological order because they are numbered according to their occurrence in the course of the project. For example, if event chain 1 starts at event 1, event chain 2 starts at event 6, and event chain 3 at event 10, then events 7 to 9 would be part of either event chain 1 or 2).

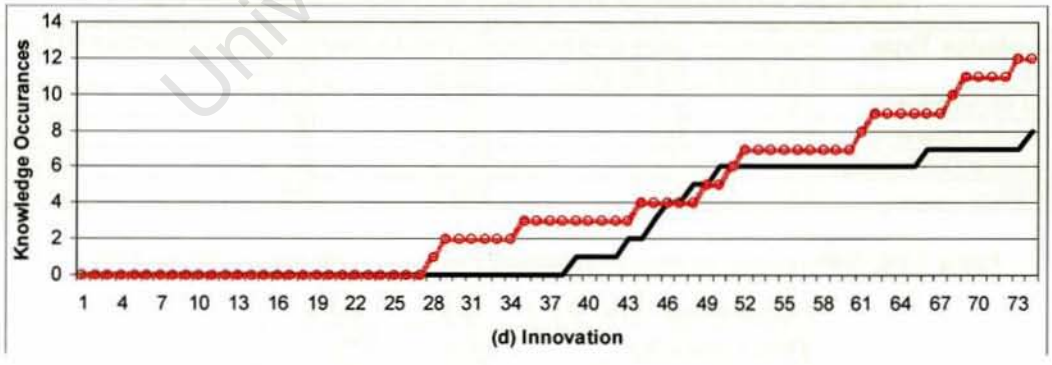
Despite the differences in vertical and horizontal axes and despite the difference between knowledge occurrences and productive / non-productive time, there are similarities in the trends show in the two types of graphs.

University of Cape Town









Graph 4.3: Productive and non-productive knowledge occurrences for P1-1 (SoSiG).

University of Cape Town

4.5.3.2 Knowledge occurrence tables and graphs for P1-2 (ANTCON)

For Project P1-2, there were 77 knowledge occurrences; one knowledge occurrence for each event was determined from the analysis of data performed in Section 4.3.2. This total was split into 35 data knowledge occurrences, 32 process knowledge occurrences and 10 innovation knowledge occurrences (see Table 4.22).

As in the case of Project P1-1, a percentage breakdown for data, process and innovation knowledge was calculated (see Table 4.23) from the statistics provided in Table 4.22. Using the results in Table 4.11, percentages for productive and non-productive knowledge were determined, using the amount of time spent on productive and non-productive tasks (see Table 4.24). Absolute differences between percentage breakdowns for the knowledge categories are shown in Table 4.25.

Table 4.22: Knowledge occurrences per knowledge type for P1-2.

Knowledge Type: P1-2	PK	NPK	Tot
Data knowledge	12	23	35
Process knowledge	15	17	32
Innovation knowledge	6	4	10
Combined	33	44	77

Table 4.23: Productive and non-productive knowledge within knowledge types for P1-2.

Knowledge Type: P1-2	PK	NPK	Tot
Data knowledge	34%	66%	100%
Process knowledge	47%	53%	100%
Innovation knowledge	60%	40%	100%
Combined	43%	57%	100%

Table 4.24: Breakdown of knowledge acquisition times for P1-2.

Knowledge Type: P1-2	Productive Time (h)	Non-productive Time (h)	% Productive Time	% Non-productive Time	Total
Data knowledge	13	23	36 %	64 %	100 %
Process knowledge	11	17	39 %	61 %	100 %
Innovation knowledge	15	11	58 %	42 %	100 %
TOTAL	39	51	36 %	64 %	100 %

Table 4.25: Differences of results between analysis methods for Project P1-2.

Knowledge Type: P1-2	ΔPK	ΔNPK
Data knowledge	2%	2%
Process knowledge	8%	8%
Innovation knowledge	2%	2%
Average of above	4.0%	4.0%
Combined knowledge	7%	7%

For this project, the percentage breakdowns for productive and non-productive knowledge using the knowledge occurrence method and the productive / non-

University of Cape Town

productive time method were very similar; there was an average difference of 4%. The difference in productive knowledge was slightly higher in Project P1-2 than in Project P1-1.

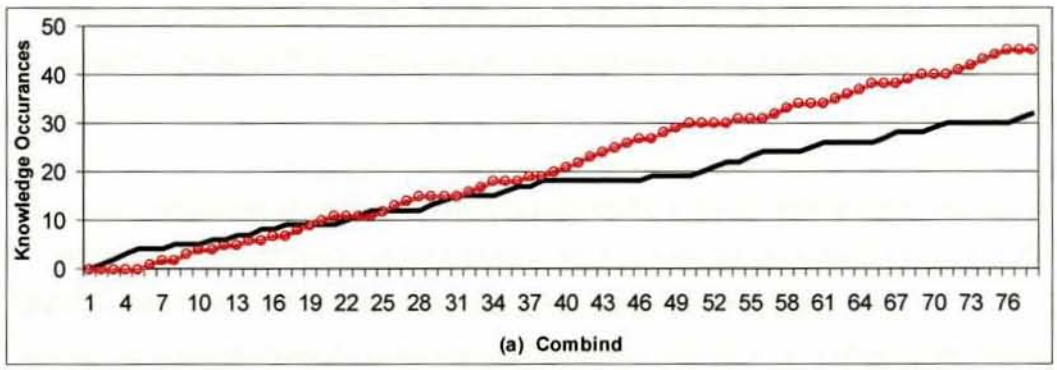
The percentage breakdown of productive and non-productive knowledge occurrences in relation to the total number of knowledge occurrences is shown in Table 4.26. These statistics are used in Chapter 5 to see the effect the ESAOA KMS has on Experiment 2 in comparison to it not being used in Experiment 1.

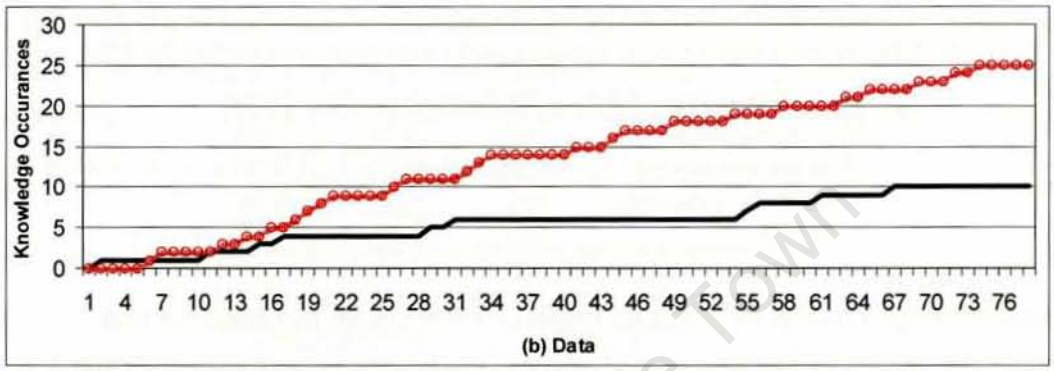
Table 4.26: Proportions of data, process and innovation knowledge for P2-2.

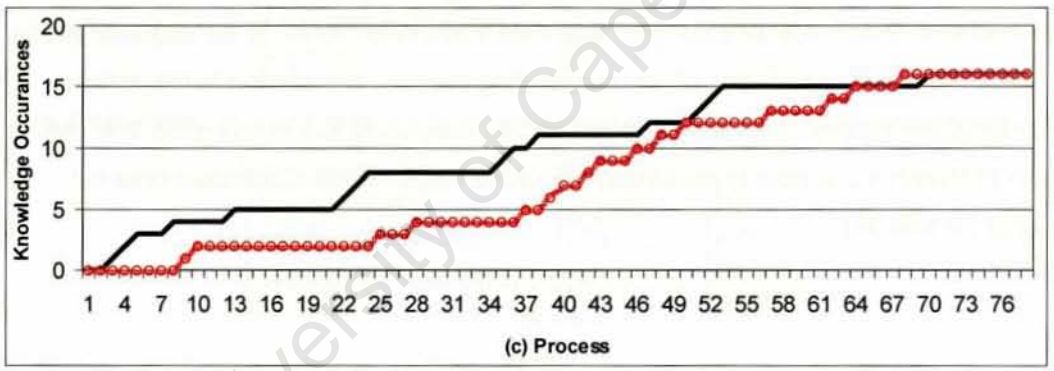
Knowledge Type: P1-2	PK/Total	NPK/Total	Total
Data knowledge	16%	30%	45%
Process knowledge	19%	22%	42%
Innovation knowledge	8%	5%	13%
TOTALS	43%	57%	100%

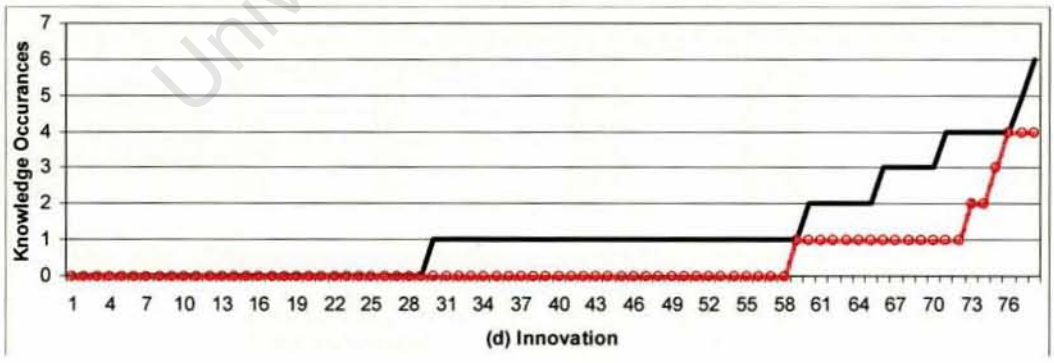
The knowledge occurrence graph for Project P1-2 is shown by Graph 4.4 (the productivity graph for this project is shown by Graph 4.2). As was seen in Graph 4.2, the trends in Graph 4.4 also suggest that data knowledge tends to be acquired earlier in the project, with a focused effort on acquiring process knowledge in the middle quarter of the project. The dip in the productive time trend line at event 38 that was seen in Graph 4.2 is now represented as earlier steps in non-productive time (at events 28 and 37).

University of Cape Town









Graph 4.4: Productive and non-productive knowledge occurrences for P1-2 (ANTCON).

University of Cape Town

4.5.4 Overall results of Experiment 1 in knowledge occurrences

The differences in the percentage breakdowns shown in Table 4.20 and Table 4.25 show that the results produced by the two different data analysis methods were very similar. Since the two methods produced similar results, it was decided that the knowledge occurrence strategy (and its significant savings in terms of effort for analysing data) would be applied in Experiment 2. Combined averages for Project P1-1 and P1-2 are given in this section in terms of knowledge occurrences, which will later be used to compare the averaged results for Experiment 2.

Overall, the average percentage of productive knowledge occurrences was 36% for Experiment 1, and non-productive knowledge occurrences amounted to 64%. The percentage breakdowns of averaged knowledge occurrences separated into the knowledge categories are summarised in Table 4.27. Table 4.28 shows the percentage of productive and non-productive knowledge occurrences per project.

Table 4.27: Average of productive and non-productive knowledge for knowledge categories for Experiment 1.

Knowledge Type	% Productive knowledge	% Non-productive knowledge	Total
Data knowledge	11%	29%	40%
Process knowledge	16%	25%	41%
Innovation knowledge	10%	11%	20%
TOTALS	36%	64%	100%

Table 4.28: Total productive and non-productive knowledge for Experiment 1.

Project	% Productive knowledge	% Non-productive knowledge	Total
P1-1	32%	68%	100%
P1-2	40%	60%	100%
Average	36%	64%	100%

4.6 First application of framework construction: ESAOA KMS version 1

The ESAOA KMS was designed around the generic structure of a KMS as presented in Section 2.6.4; this structure comprises the six parts of: roles, groups, desires, work, workflows, and artefacts [Van der Spek & Spijkervet, 1997]. The choice of roles, desires, and other design aspects of the ESAOA KMS version 1 are based on Experiment 1, and builds on the literature (see Chapter 2).

University of Cape Town

An overview of the design and operation of ESAOA KMS version 1 is provided in Section 4.6.1. Section 4.6.2 explains the concept of ESAOA workspaces and ESAOA workstations, which were used to integrate aspects of the KMS as a unified system. The ESAOA knowledge ontology is described in Section 4.6.3. The roles, groups and their desires are detailed in Section 4.6.4. Artefacts and artefact classification mechanisms of the ESAOA KMS are explained in Section 4.6.5. The workflows and processes of the KMS are documented in Section 4.6.6, and they are arranged according to the roles that perform them. Section 4.6.7 presents the software design of the KM tools that form part of the ESAOA workspaces. Section 4.6.8 explains how the ESAOA workspaces were implemented, packaged and distributed using *ESAOA workspace distributions*.

4.6.1 Overview of ESAOA KMS version 1

The high-level design of ESAOA KMS version 1 comprises interlocking systems called *ESAOA workspaces* (or just *workspaces*) for managing data, process and innovation knowledge. These workspaces are designed to operate together in order to drive an ES development team towards sharing information and producing innovation knowledge early on in the project. An ESAOA workspace is a computer-based work area that comprises a shell environment and a collection of *soft artefacts* (i.e., digital, computer-based files) organised into a directory structure, which follows a specific layout and classification schema, and has an integrated knowledge base, ESAOA support tools, and a related collection of externally stored and maintained development tools. An ESAOA workspace is accessed via an *ESAOA workstation*. The soft artefacts of the workspace are accessed, organised, modified and supplemented by the *roles* guided by *processes* of the ESAOA KMS. The above mentioned aspects of the ESAOA KMS are summarised as follows:

- **ESAOA workspace:** digital work area, which integrates software tools and digital artefacts, that developers work on;
- **ESAOA workstation:** the combination of a computer system, the physical work area in which an ES is worked on, and the software tools (such as SSH client programs) that provides the human/computer interface to an ESAOA workspace;
- **ESAOA roles:** a description of responsibilities and characteristics that individuals take on at certain times during development;
- **ESAOA processes:** provide KM guidelines for commonly occurring activities carried out by one or more roles.

University of Cape Town

The ESAOA procedures for the first version of the KMS (Section 4.6.6) are based on Experiment 1, and chosen in accordance with the work that developers are likely to perform in a project. These procedures are expressed in the form of models using the *ESAOA conceptual modelling language* (defined in Section 3.11). Each ESAOA process generally involves one or more roles using, modifying, or constructing artefacts in ESAOA workspaces.

The *ESAOA conceptual modelling language* was developed as a means of modelling aspects of the ESAOA KMS and their interrelations (Section 4.2.1 describes how the modelling language came about). The modelling language applies techniques used in the Unified Modelling Language (UML) [Rumbaugh *et al.*, 2005]. The ESAOA conceptual modelling language comprises modelling *atoms* and *connectors*. Atoms relate to KM aspects (i.e., glyphs for roles, artefacts and so on) and other parts of the KMS. *Connectors* represent defined associations between atoms. Section 3.11 defines the ESAOA conceptual modelling language for ESAOA KMS version 2, which includes a visual reference table for all the atoms and connectors (note that the second version of the modelling language is a refined version of the first version, and as such only has a few added atoms and connectors that were not found in the first version of the modelling language).

Figure 4.7 provides a UML class diagram that illustrates the design of the KMS as described above. The *ESAOA workspace* and *ESAOA workstation* blocks in the diagram are highlighted as they are high level components of the KMS design in which the other items reside.

University of Cape Town

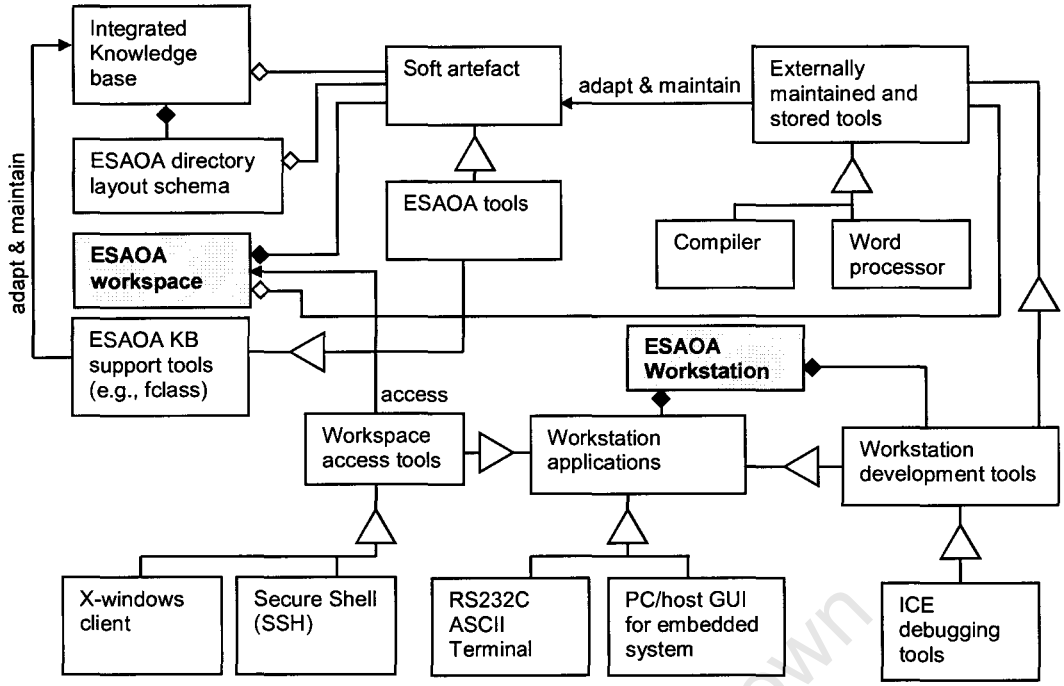


Figure 4.7: ESAOA workspace and ESAOA workstation.

As Figure 4.7 shows, soft artefacts that developers work on reside within an ESAOA workspace. The ESAOA workstation is a collective of both soft artefacts, such as workstation applications (i.e., software programs) and physical tools (i.e., hard artefacts) that exist outside the computer. The diagram shows that *workspace access tools* are a form of workspace application that are used specifically to access software tools and other soft artefacts that reside within an ESAOA workspace – an SSH client program (e.g., Putty [Tatham, 2009]) is an example of such a tool. Note that workspace access tools reside on the ESAOA workstation (the computer used to access the central server on which the ESAOA support tools, cross-compilers and similar tools are stored). Software tools such as cross-compilers, IDEs and word processing programs (which often take large amounts of disk space, are installed, and not themselves adapted) are termed *externally maintained and stored tools* – as Figure 4.7 shows, these tools are used to adapt soft artefacts within an ESAOA workspace. The *integrated knowledge base* is a combination of the directory structure, classifications applied to artefacts, developer logs, and documentation stored in the workspace. The integrated knowledge base, and its operation, is further described in Section 6.3.3.

University of Cape Town

4.6.2 ESAOA workspaces and workstations

In order to construct the ESAOA KMS and to make it usable, a strategy was needed to unify the roles, processes and artefacts of the KMS and to present these to knowledge workers as a functional system, specifically a system providing a human interface aspect and a defined internal interrelated structure and functionality [Weinberg, 1975; Brown, 1999]. This objective was achieved using the combination of ESAOA workspaces and ESAOA workstations.

4.6.2.1 ESAOA workspaces

An ESAOA workspace is a digital, computer-based work area that comprises the following parts:

1) A shell environment (an extended version of the Bash shell[3]);
2) Soft artefacts (i.e., computer files) organised into an ESAOA directory structure (following a specific layout and classification schema);
3) ESAOA support tools; and
4) A related collection of externally stored and maintained development tools (e.g., compilers and CAD software).

Soft artefacts for the ESAOA version 1 workspaces (see Section 4.6.5) were taken from project repositories produced by Experiment 1 teams, and these artefacts were improved and supplemented in the course of creating the workspaces. Figure 4.8 provides a screenshot illustrating the ESAOA environment, accessed via an X-terminal. Section 4.6.2 provides more detail concerning the composition and organisation of ESAOA workspaces and their artefacts.

The development teams start from a baseline ESAOA workspace, which incorporates a pre-selected set of *soft artefacts* (i.e., artefacts in digital form, such as software and other artefacts needed by the ESAOA KMS). These artefacts are then extended or supplemented by a *role* in the course of using the KMS. The work and workflow aspects of the KMS describe methods by which one or more roles acquire ESAOA knowledge (i.e., how they learn to use, modify and construct ESAOA artefacts).

[3] The Bourne-Again Shell (Bash)

University of Cape Town

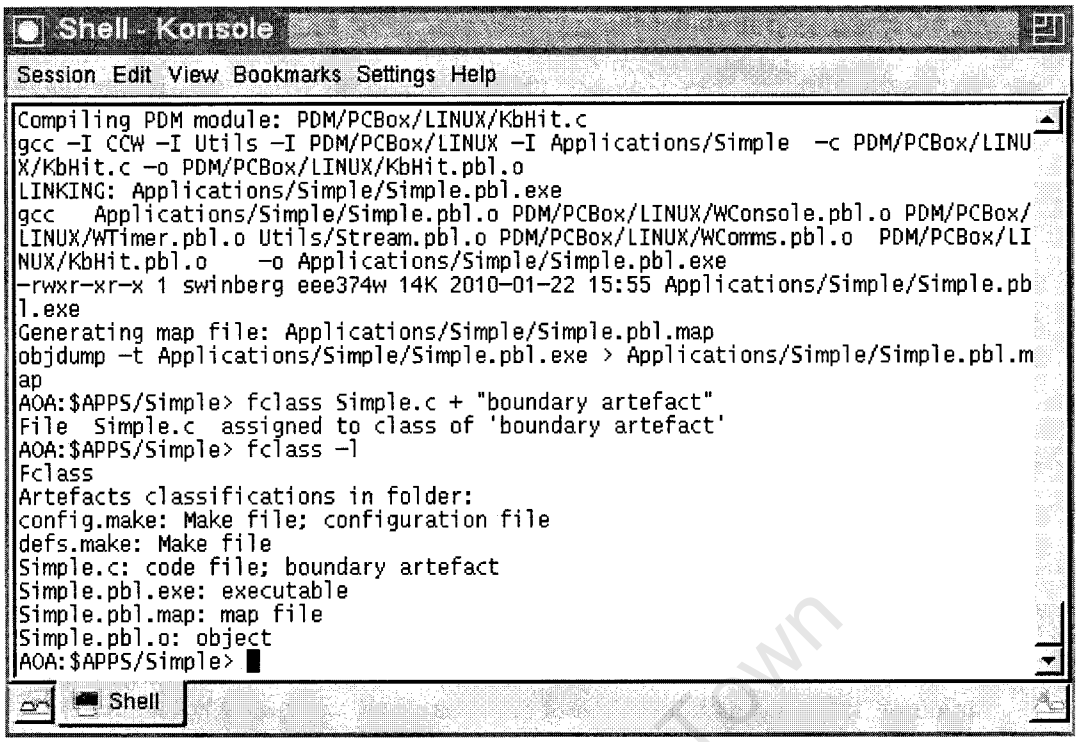


Figure 4.8: Screenshot of the Bash shell environment of an ESAOA workspace.

ESAOA KMS version 1 had three ESAOA workspaces: 1) a *communal workspace*, 2) a *team workspace*, and 3) a *personal workspace*. Workspaces were accessed using *workstations*. The communal workspace was shared between all projects. The team workspace was used to maintain master versions of soft artefacts for a team, and to share artefacts among the team members. Each team member worked on their own personal workspace, which was synchronized with the team workspace (for version 1 of the ESAOA KMS, this synchronization process was left as a manually operation, but some Experiment 2 teams used version control tools to facilitate this task). The second version of the KMS has the same three workspaces, but they are extended version of those developed for the first version of the KMS (see Section 6.1.1).

University of Cape Town

4.6.2.2 ESAOA workstations

The term *ESAOA workstation* refers to the combination of a computer system (termed the *workstation computer*), which provides the human/computer interface to an ESAOA workspace, together with the surrounding physical artefacts that developers use during the development of an ES (such as printouts of datasheets, books, testing equipment and the ES hardware worked on). Essentially, an ESAOA workstation is much the same as any normal work area with a computer that is used during ES development. The term ESAOA workstation is used in relation to the

ESAOA KMS simply as a means to emphasise that a particular workstation is intended for use with an ESAOA workspace.

In addition to software tools needed to access ESAOA workspaces, the ESAOA workstation computer is likely to need a variety of other tools in order to enable developers to complete development activities. Such tools include text editors, web browsers and communication programs to interact with the ES products. Figure 4.9 shows an annotated screenshot of a typical ESAOA workstation, showing the standard software applications (e.g., Microsoft Excel, Explorer and HyperTerminal) that are commonly used on ESAOA workstations.

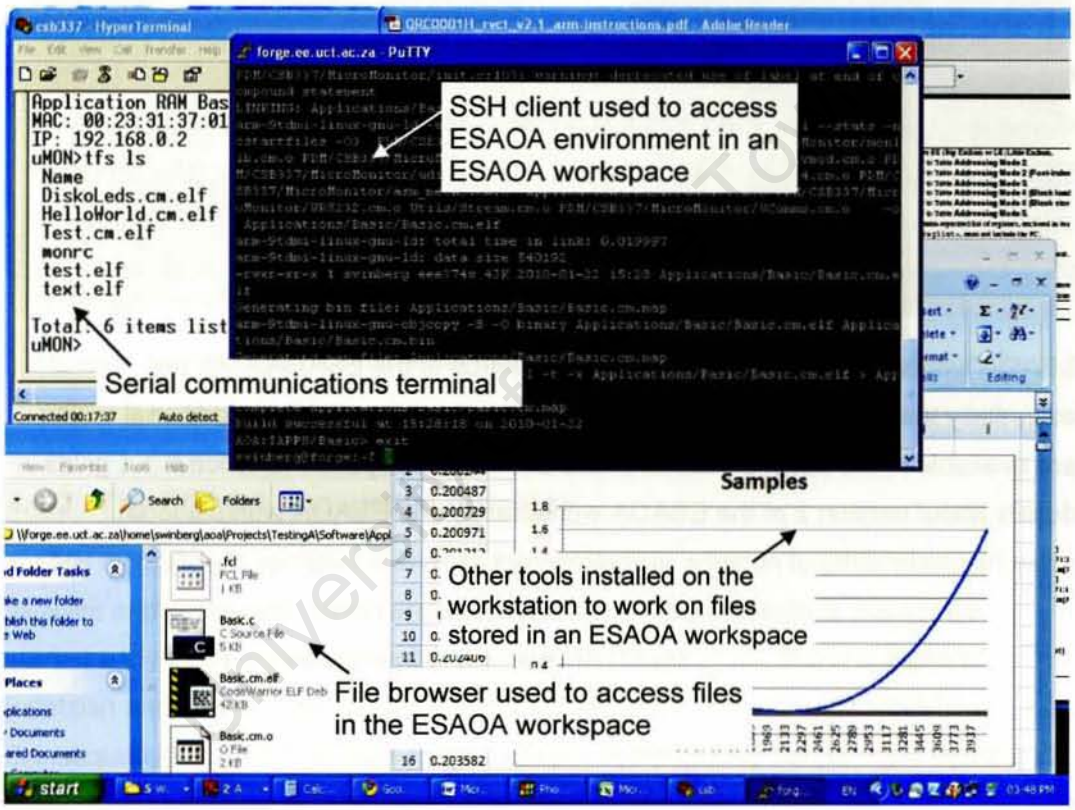


Figure 4.9: Annotated screenshot of an ESAOA workstation.

The workstation setup is expected to vary from one team member to another depending on the needs of the team concerned. Each workstation is expected to include at least the following software allocations:

- A SSH client program for accessing remote computer (or computers) that store the team and communal workspaces;

- A serial communications terminal (e.g., Windows HyperTerm); and
- Standard networking software (e.g., web browsers).

A personal workspace, which is a working copy of the team's workspace, also needs to be accessible from an ESAOA workstation. The personal workspace can be kept on a remote computer (e.g., the same central server that hosts the team workspace), or the personal workspace can be stored on the local computer workstation. In the case that personal workstations is stored locally, then the ESAOA workspace environment, together with software it depends on (i.e., the Bash shell and its dependencies) also need to be installed on the workstation computer.

As mentioned above, the concept of an ESAOA workstation is intended to refer to the broader space in which artefacts relating to development are located. The screenshot shown in Figure 4.9 only illustrates soft artefacts (e.g, software tools) related to an ESAOA workstation. ESAOA workstations include a variety of hard artefacts that are also used during development, such as books and hardware computers; Figure 4.10 provides a photograph illustrating the broader aspects of an ESAOA workstation.

ESAOA workspaces used with the second version of the ESAOA KMS are essentially the same as those used in the first version, except that additional tools are available in the second version of the ESAOA workspaces (Section 6.1.1 gives details about version 2 of the ESAOA workspaces and ESAOA workstations).

University of Cape Town



Figure 4.10: Photograph illustrating the broader concept of an ESAOA workstation.

4.6.3 The ESAOA knowledge ontology

A KMS typically incorporates a *knowledge ontology* in addition to other aspects (such as roles and processes, which are described later). The ontology is likely to be started early in the establishment of a new KMS [Staab *et al.*, 2001] in order to establish and share a specialised vocabulary needed to describe the KMS itself. Consequently, the ESAOA knowledge ontology is introduced before more detail is given on the other aspects of the ESAOA KMS.

In the information sciences and computer science fields, the term *ontology* generally refers to "a specification of a representational vocabulary for a shared domain of discourse" [Gruber, 1993, pg. 199]. The term *ESAOA knowledge ontology* in this thesis is based on the preceding definition of ontology, and accordingly refers to a specialised terminology structure used to specify the ESAOA KMS (i.e., a specific 'domain of discourse'). The *ESAOA conceptual modelling language* should not be confused with the ESAOA knowledge ontology: the former describes the notation for visual models that aids textual explications using the ESAOA knowledge ontology.

4.6.3.1 Levels of the ESAOA knowledge ontology

While the ontology is interpreted abstractly as an interrelated whole, the representation of the ESAOA knowledge ontology is in two parts: a higher-level part and a lower-level part.

The higher-level part is essentially a dictionary of terms (i.e., a set of terms and corresponding definitions); except that the individual terms are represented and handled following an object-oriented approach whereby terms can be specialisations of another term. Accordingly, this higher-level part is displayed and browsed in a tree structure (making it easy to determine which terms are specialisations of other terms). This higher-level part is intended only for use by human users of the KMS, and is not intended to be interpreted in some way by programs.

The lower-level part of the ontology is maintained within the knowledge base that is integrated into an *ESAOA workspace* (this data is in the form of CSV files within project directories). This part of the ontology predominantly relates to the classification of artefact, relations between artefacts, and relations between artefacts and which roles are responsible for maintaining them. The ESAOA KMS ontology is intended to change from one project to another, as different projects will not necessarily need the same set of terms and artefact classifications; this is a reason for the lower-level part of the ontology being managed in a different manner.

4.6.3.2 Top-level terms of the ESAOA knowledge ontology

The starting point of the ESAOA knowledge ontology has five top-level terms, namely: role, activity, process, artefact and space. Top-level are not a specialised form of any other term in the ontology. All the other terms in the ontology are specialisations of these five top-level terms or specialisations of terms lower in the hierarchy. Additional top-level terms can be added as needed when the KMS is in use. Figure 4.11 shows a UML class diagram that visualizes these terms (and a selection of commonly used specialised terms). The terms shown in Figure 4.11 are explained in Table 4.29.

University of Cape Town

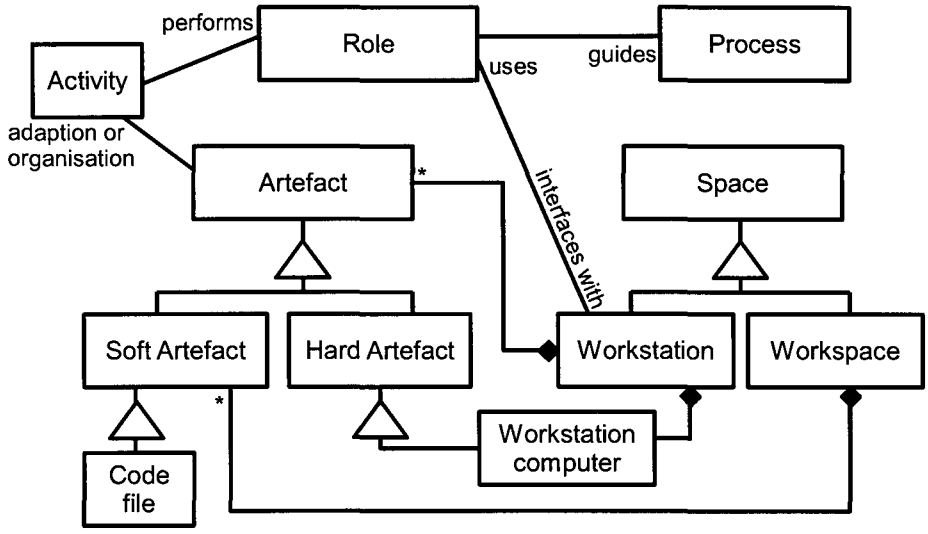


Figure 4.11: UML diagram visualizing part of higher-level ESAOA knowledge ontology.

Table 4.29: Roles of ESAOA version 1.

Term	Specialisation of	Description
Role	-	Describes the behaviour, responsibilities, characteristics and needs a developer takes on while performing certain types of development operations.
Activity	-	One or more related actions, carried out by a role (or roles), related to development of an embedded system.
Artefact	-	A physical resource (e.g., lab equipment) or digital resource (e.g., software tools and data files) used in *activities*.
Space	-	A location (digital or physical) in which *artefacts* reside.
Process	-	A description of related *activities* provided as a guideline to help *roles* carry out a development objective.
Soft artefact	Artefact	A digital *artefact* stored in a computer file system (e.g., a code file or application program).
Hard artefact	Artefact	A physical *artefact* that resides outside a computer file system (e.g., books and laboratory equipment).
Workspace	Space	A *soft space* where development *activities* that involve changes to *soft artefacts* take place, comprising tools and other *artefacts* used to carry out these activities.
ESAOA workspace	Workspace	A computer-based work area comprising a shell environment, soft artefacts, a knowledge base and support tools. An ESAOA workspace also depends on a selection of externally stored software applications.
Workstation (or ESAOA workstation)	Space	Combination of a computer system, the physical work area where an ES is worked on (including printouts and the product being worked on), and other software tools (such as SSH client programs) that provides the human/computer interface to an ESAOA workspace.
Workstation computer	Space; Hard artefact	This term refers specifically to the computer in a workstation.
Code file	Soft artefact	A file that contains code.

University of Cape Town

4.6.3.3 Knowledge artefacts and boundary artefacts

The terms *knowledge artefact* and *boundary artefact* have specific meaning in the ESAOA ontology, and although they are not defined as top-level terms, they are commonly used in reference to aspects of the KMS. Both of these terms are *artefact* specialisations.

The transfer and production of knowledge generally involves digital or physical artefacts referred to as *knowledge artefacts* [Knorr-Cetina, 2001]. These artefacts are used both for creating knowledge and for transferring knowledge. The term *boundary artefact* [Arias & Fischer, 2000] refers to an object used as part of a discussion point, or something that is manipulated in some way to help the transfer of knowledge in a context of discourse (e.g., a meeting between engineers). Within an ESAOA workspace, soft artefacts can be demarcated as knowledge artefacts, in order to emphasise that they are used in building knowledge. Similarly, artefacts can be marked as boundary artefacts to highlight that viewing and manipulation of these artefacts may facilitate knowledge sharing during demonstrations and meetings. The demarkation of soft artefacts as boundary or knowledge artefacts is accomplished using the *fclass* tool (discussed in the next section).

4.6.3.4 Evolving the ESAOA knowledge ontology

The ESAOA knowledge ontology is adapted and grown as needed while the ESAOA KMS is in use; this is done by changes to the *functionality*, *workspace*, and *maintainer* classifications of artefacts in a project (explained in Section 4.6.4; form classifications cannot be changed), or by editing a copy of the original ESAOA knowledge ontology document (shown in Appendix C.1) that is shared between team members. The artefact classifications are altered by using the *fclass* ESAOA support tool (for modifying functionality or maintainer classifications), or by moving or copying files between workspaces (to alter workspace classifications).

The artefact classifications can be considered as being at a lower level within the ESAOA knowledge ontology, and are consequently not recoded in the ontology document because they are maintained by the *fclass* tool within the knowledge base integrated into an ESAOA workspace. Thusly, the ESAOA knowledge ontology effectively changes from one ESAOA workspace to the next. For ESAOA KMS version 1, individual team members make changes to the artefact classifications, and the rest of the team is automatically updated when that individual synchronizes his or her personal workspace with the team workspace). A team member makes higher

University of Cape Town

level ontological changes, such as the addition of new terms that are not used in classifying development artefacts, by changing his or her own copy of the ESAOA knowledge ontology, and needs to manually propagate these changes to the team.

The ESAOA knowledge ontology document produced for ESAOA KMS version 1 was extended in the second version. In order to save space, only version 2 of the full ESAOA ontology is provided in this thesis (see Appendix C.1).

4.6.4 Roles

As defined in the ESAOA ontology, a role describes the characteristics, behaviour and needs that individuals take on for certain types of development operations. The roles of the ESAOA KMS are provided as starting points for team member responsibilities and as a means to partition knowledge work amount team members. As a particular development project progresses, the responsibilities of particular roles are likely to change and adapt to the specific needs and behaviour of the teams.

Observations of Experiment 1 indicate that innovation happens through the availability of effective process knowledge needed to experiment with ideas, and, in a similar way, process knowledge is dependent on data knowledge (see Section 4.4.3). The design of ESAOA KMS version 1 and its roles is based on this flow of knowledge. Accordingly, three of the knowledge worker roles for the KMS are based on these dependencies between data, process and innovation knowledge observed in Experiment 1. An additional three roles are included for facilitation of teams and maintenance of the KMS. The role names, and their corresponding acronyms, are as follows: *chief knowledge officer* (CKO), *communal knowledge steward* (CKS), *team leader* (TL), *data steward* (DS), *process engineer* (PE), and *innovation engineer* (IE). The roles are described in Table 4.30. The *team member* (TM) role (entry seven in the table) is an abstraction used to refer collectively to all types of team members.

The CKO, CKS and DS roles are all inspired by the generic KM roles reviewed in Section 2.7. The CKO of an ESAOA KMS essentially has the same responsibility as those described for a generic CKO [Holsapple, 2003] (see Section 2.7.2). The DS has responsibilities similar to those of a 'knowledge steward' [Tsui, 2002], and the CKS role has elements of a 'knowledge engineer' [Borghoff & Pareschi, 1998], 'change agent' [Groff & Jones, 2003] and 'knowledge steward'.

University of Cape Town

Table 4.30: Roles of ESAOA version 1.

Role name	Acronym	Description
Chief knowledge officer	CKO	The individual responsible for guiding users of the ESAOA KMS, implementing or preparing new communal ESAOA artefacts and procedures for distribution, and maintaining the communal ESAOA workspace.
Communal knowledge steward	CKS	The CKS assists the CKO in assisting users of the ESAOA KMS, explaining procedures, organising new communal artefacts, and maintaining the communal ESAOA workspace.
Team leader	TL	The team leader is responsible for managerial aspects of the team, such as ensuring that team members are performing.
Data steward	DS	The DS obtains and produces data artefacts (such as datasheets and meeting minutes), comprehends them, and places them in the ESAOA team workspace. The DS is responsible for maintaining the organisational structure of the ESAOA team workspace and for supporting the PE.
Process engineer	PE	The PE experiments with processes using data knowledge and support provided by DS. The PE is responsible for creating process artefacts and for supporting the IE.
Innovation engineer	IE	The IE applies processes to test ideas and to produce innovations. The IE is supported by the PE.
Team member	TM	An abstract aggregation role name, used to refer to all team members collectively, i.e. the TL, DS, PE and IE.

4.6.4.1 Representation of roles in the ESAOA modelling language

Roles are represented by the ESAOA conceptual modelling language as circles containing the acronym for the role name. The model presented in Figure 4.12 shows the six ESAOA KMS roles, in addition to the abstract TM role. The connectors (i.e., the arrows in Figure 4.12) are referred to as role interaction associations; these connectors represent an association between two roles, the label providing keywords describing the association. The inheritance association is represented as in standard UML, with lines leading from an element through a triangle to an inherited element.

4.6.4.2 General relations between the roles

People involved with the KMS can perform one or more roles; one person can also take on different roles at different times. For example, a particular team member might start performing a DS role, and later change to an IE role. Furthermore, a particular team might have multiple team members filling the same roles (e.g., three DS roles, a PE role, and an IE role).

ESAOA KMS version 1 has five distinct knowledge worker roles (modelled in Figure 4.12). These roles are largely non-hierarchical, with the exception of the chief knowledge officer (CKO), who is in charge of the KMS. In the case of Experiment 2 (where ESAOA KMS version 1 was applied), the researcher played the part of the

University of Cape Town

CKO, and a separate CKS was hired. Each development team had a team leader, who was responsible or managing the rest of the team members (as illustrated in Figure 4.12) and also performed development, taking on other roles while doing so.

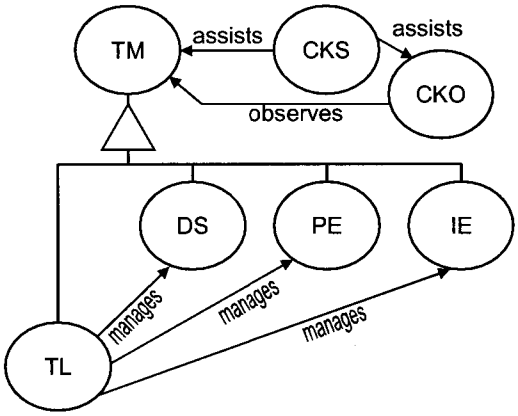


Figure 4.12: Roles of ESAOA KMS version 1.

4.6.4.3 Maximising support for the IE using a feed-forward approach

The roles of the ESAOA KMS version 1 are structured to maximise support for the IE. Ideally, the IE will be able to focus on testing design ideas, without the need to read the detail of particular components and determine how they are controlled and connected. The TL is in charge of managing the members of a team and for liaising with the CKO to ensure that the ESAOA KMS is supporting the team. The individual who has been assigned the team leader role also performs at least one of the DS, PE or IE roles. At a high level, this version of the KMS is designed around providing support to the IE, essentially establishing a strategy by which the project teams are guided towards producing innovation knowledge early on in the project. The KMS thus follows a *feed-forward* flow of knowledge production, around which the roles and processes are structured.

Figure 4.13 models the interrelation of the roles defined for the KMS (the modelling atoms and connectors shown in the figure are defined in Section 3.11). The model indicates that the CKO and CKS chiefly operate in the communal workspace. The IE, PE, and DS chiefly operate in their individual personal workspaces, or their shared team workspace, although they also use the communal workspace. The IE, PE and DS are supported by the CKO and CKS, as implied by the *role support connector* line (shown as a line ending in a dot that links the two workspaces). The model also indicates that, in terms of supervising KM strategies, the DS, PE and IE roles all report to the TL and that the TL consults the CKO regarding KM strategies.

University of Cape Town

The CKS assists the CKO, performing tasks such as maintaining communal knowledge artefacts. The team member roles were supported by individuals filling the CKO and CKS roles. The CKO was responsible for administering the KMS and training knowledge workers to use the system. KM responsibilities of the TL were not defined for this version of the KMS because Experiment 1 findings did not show a need for this role to be involved in managing ESAOA knowledge. Section 4.6.4 describes the roles of this KMS, which implement this design strategy.

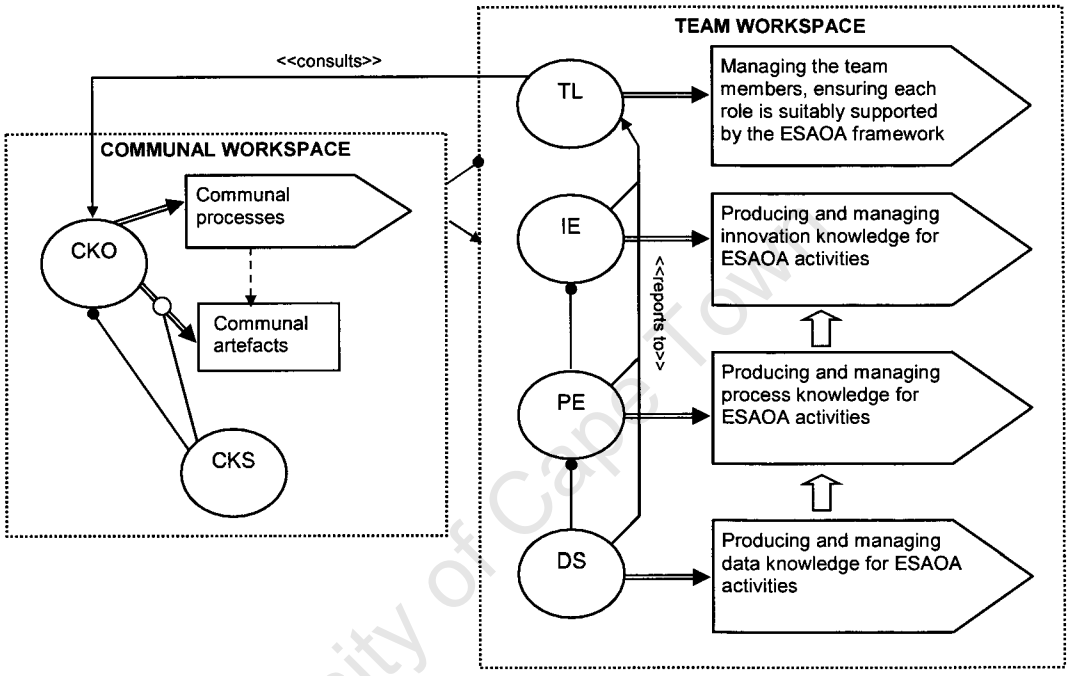


Figure 4.13: Role support structure for ESAOA version 1.

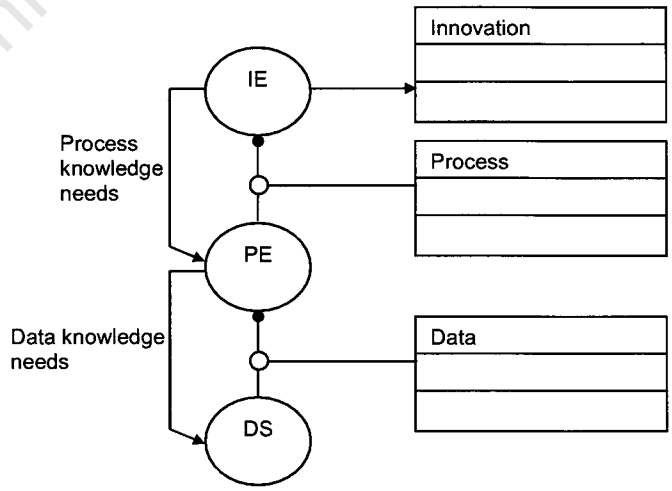


Figure 4.14: The feed-forward flow from the DS, to the PE, ending at the IE.

University of Cape Town

The feed-forward flow of ESAOA KMS version 1 is shown by the way in which the IE draws on the PE to provide methods to test innovations, and the PE in turn depending on the DS to provide data knowledge needed to develop process knowledge. Figure 4.14 models this interdependence between the IE, PE and DS.

4.6.5 ESAOA artefacts for knowledge representation and transfer

While knowledge exists in the mind of a person, the transfer and production of knowledge typically involves physical artefacts, often referred to as *knowledge artefacts* or "knowledge objects" [Knorr-Cetina, 1997, pg. 1][4]. These artefacts are used both to create knowledge and to transfer knowledge. In the case of transferring knowledge, particularly in a discourse context (e.g., in a meeting or "extreme programming" situation [Wake, 2002, pg. 1]), the term "boundary object" [Arias & Fischer, 2000, pg. 2] refers to an object used as part of a discussion point, or to something that is manipulated in some way to help the transfer of knowledge.

These concepts of knowledge objects and boundary objects apply to knowledge in general, and they are refined in the design of the ESAOA KMS. In the ESAOA KMS, the terms *knowledge artefact* and *boundary artefact* are used instead of 'knowledge object' and 'bounding object' to avoid confusion with the more common definitions of these concepts.

In Experiment 1, different types of knowledge artefacts and boundary artefacts were used for producing, capturing and expressing the different forms of ESAOA knowledge. The artefacts and artefact classifications for ESAOA KMS version 1 are based on artefacts producing by developers in Experiment 1.

In ESAOA KMS version 1, each artefact has four classifications, namely classifications of: *form*, *functionality*, *workspace*, and *maintainer*. These classifications are described below:

- **Form classification** is based on whether the artefact is in a digital or non-digital form, and whether represents explicit knowledge, among other

[4] Knorr-Cetina defines a knowledge object as a document or collection of documents in which knowledge is maintained.

University of Cape Town

properties. The terms *soft*, *hard*, *boundary* and *knowledge* characterise the form of an artefact.

- **Functionality classification** is based on what the artefact provides or is used for during the project (e.g., documentation or code) – the functionality classification essentially indicates what the actual artefact is.
- **Workspace classification** indicates where the particular artefact resides, and where it comes from (e.g., the artefact is in a communal workspace and comes from the web).
- **Maintainer classification** indicates the role that is responsible for maintaining the artefact (this classification is partly implied by the workspace classification; for example, artefacts in the communal workspace are maintained by the CKS).

The workspace and maintainer classifications are more abstract than the functionality classifications, since they do not indicate what an artefact is, but rather where it was obtained and located, and who maintains it. However, workspace and maintenance classifications provide an effective means of referring to collections of artefacts (e.g., communal artefacts refer to all artefacts residing in the communal workspace).

The artefact classifications are used in Section 4.6.6, in which the ESAOA KMS workflows are presented; that section also highlights the particular knowledge artefacts that are used to capture various forms of ESAOA knowledge, and the use of boundary artefacts in situations of knowledge transfer.

The ESAOA KMS artefacts are presented in the subsections that follow. The form classifications for artefacts are described in Section 4.6.5.1. The functionality classifications are presented thereafter in Section 4.6.5.2, which serves the purpose of indicating the specific artefacts of the KMS and related KM tools. The workspace and maintenance classifications are presented together in Section 4.6.5.3 as workspace classifications often lead to particular maintenance classifications. Section 4.6.5.4 involves organisation of artefacts in ESAOA workspaces, and Section 4.6.5.5 concerns specialised KM artefacts (used by teams in Experiment 1), and included in the ESAOA knowledge ontology.

4.6.5.1 Artefact form classifications: hard and soft artefacts

The form classifications are implicit, being based on the nature of the artefact, in other words, whether it is digital or not, whether it contains information or whether it is

University of Cape Town

a physical item (e.g., a piece of equipment). Each artefact has one of two fundamental, top-level form classifications; each artefact is either a *hard artefact* or a *soft artefact*. Additional form classifications of *boundary artefact* and *knowledge artefact* can be overlaid on the top-level classification. For example, an oscilloscope could be classified as a 'hard boundary artefact'. The definition of the form classifications are elaborated upon below:

- **Soft artefacts** are digital files stored on a memory device and accessible by a computer. Examples include: a Microsoft Word document, an executable program, a code module, or any other digital file. A paper document (i.e., a hardcopy of a digital document or notes on a piece of paper) is also considered a soft artefact.
- **Hard artefacts** refer to physical items, excluding digital files, paper notes or hardcopies of digital documents. Some examples of *hard artefacts* are microprocessor chips, wires and multi-meters.
- **Knowledge artefacts** are a type of *soft artefact* that contains explicit knowledge relating to another soft artefact or hard artefact. An example is a microprocessor datasheet.
- **Boundary artefacts** are used in a 'discourse' [Arias & Fischer, 2000] in which two or more engineers are involved in knowledge exchange (in which engineers either discuss or adapt the boundary artefact to facilitate the transfer of knowledge). A boundary artefact is usually a knowledge artefact, but not always. An example of a soft boundary artefact could be a printout of a schematic. An example of a hard boundary artefact is a multimeter, which is used by engineers to view and discuss a character of an ES product. Hard boundary artefacts typically do not contain easily extractable information about other artefacts, but may be valuable in knowledge transfer.

4.6.5.2 Artefact functionality classifications and functionality hierarchy

Artefact *functionality classifications* are directly based on what artefacts are used for. Functionality classifications are applied to an artefact by designating the artefact as being in one or more *functionality classes*. These functionality classes are generally in the form of keywords describing functionality characteristics of an artefact; examples include: 'code module', 'datasheet' and 'drawing'.

The functionality classes used in functionality classification are based on a *functionality hierarchy*. This *functionality hierarchy* is a KM tool of the ESAOA KMS.

University of Cape Town

The functionality hierarchy is similar to that of a 'class hierarchy' used in the object-oriented software engineering paradigm [Kroll & Kruchten, 2003]. However, instead of providing a software models, the *functionality hierarchy* of the ESAOA KMS provides guidelines to assist developers in keeping track of functionality classes and in deciding which functionality classification to apply to artefacts. Each person using the ESAOA KMS is provided with a soft and hardcopy form of the functionality hierarchy (an except of which is given in Table 4.31). The master version is maintained by the CKO and resides in the communal workspace. Additional functionality classifications are added by the CKO and team members as needed. An automatic functionality hierarchy, in the form of a *functionality index file*, is maintained by the *fclass* program (an ESAOA support tool). The remainder of this section details the initial functionality classes of the ESAOA KMS and the *fclass* program.

The *functionality hierarchy* includes five top-level classes, namely: documentation, software, tools, templates and misc (for miscellaneous). Table 4.31 indicates how additional classes are added to the hierarchy below these top-level classes. Sub-categories (e.g., 'concept sketch') are indented. Entries marked with an asterisk indicate that files of that classification are collected into a folder of that classification name (e.g., all design documents are placed in folder 'Documentation/Design'). The table also indicates whether particular artefacts of a functionality class tend to occur in the form of: *boundary artefacts* (indicated by the label '(B)') or *knowledge artefacts* (identified by the label '(K)'). These boundary or knowledge artefact classifications are indicated because boundary artefacts are often accessed by multiple roles possibly at the same time or close to the same time to aid knowledge sharing; knowledge artefacts, in contrast, are generally involved in sharing knowledge, but usually not worked on jointly.

University of Cape Town

Table 4.31: Excerpt of artefact functional classes.

Artefact type	Description
Documentation	
Datasheet (K) *	A datasheet for an electronic component
Manual (K) *	A manual for a product
Design (B) *	A design document, such as a schematic for the hardware.
Diagram (B) *	A diagram used for capturing information (does not described the design)
Artefact organisation diagram (AOD) (B)	A diagram used to represent the organisation of an ESAOA workspace.
Concept sketch (B)	A visual explanation of an idea, used by the IE to communicate ideas graphically. The name given to this drawing type is adapted from Healey, *et al.* [2002].
Concept drawing (B)	A formal, more detailed and professionally drawn version of a concept sketch. Used by the IE to capture an important concept.

Report (K) *	A report (e.g., team status report)
Log (K) *	A log file
Software	
Code (B)	A code module or code-related file
Application *	Files related to the implementation of a software application. Application modules use only CCW or Utility modules (or standard C library calls).
Executable	An executable file (such as a binary image)
PDM (B) *	Platform deployment module (PDM) – these are platform dependent modules used to glue application code to lower level code (e.g., an operating system or underlying hardware).
CCW (B) *	Common component wrapper (CCW) module. A platform-independent software interface to a component available on multiple embedded platforms.
Utility (B) *	A platform-independent utility module (uses basic C library calls, CCWs or other Utility modules).
Test case	A file or collection of files representing test case(s), e.g. for use in regression testing.
Tools	
Script (K) *	A script that automates part of an ESAOA process
Program (K) *	An executable C++ program that automates part of an ESAOA process
Templates	
Code Template	Code template
Diagram Template	Diagram template
Misc	
Data	A data file, e.g., sampled input

* Entries marked with an asterisk have a separate subfolder in an ESAOA workspace.

Digital artefacts (i.e., files) within ESAOA workspaces are classified using the ESAOA *fclass* program. The *fclass* program maintains classifications within a *functionality classification index* file (named '.fci'), which is a comma separated values (CSV) file that resides in the root directory of a workspace, and *functionality classification lookup* files (named '.fcl') which reside in folders within the ESAOA directory hierarchy. The *fclass* program is detailed in Section 6.4.1. Figure 4.15 provides an annotated screenshot showing how a user can assign a functionality classification to a file.

University of Cape Town

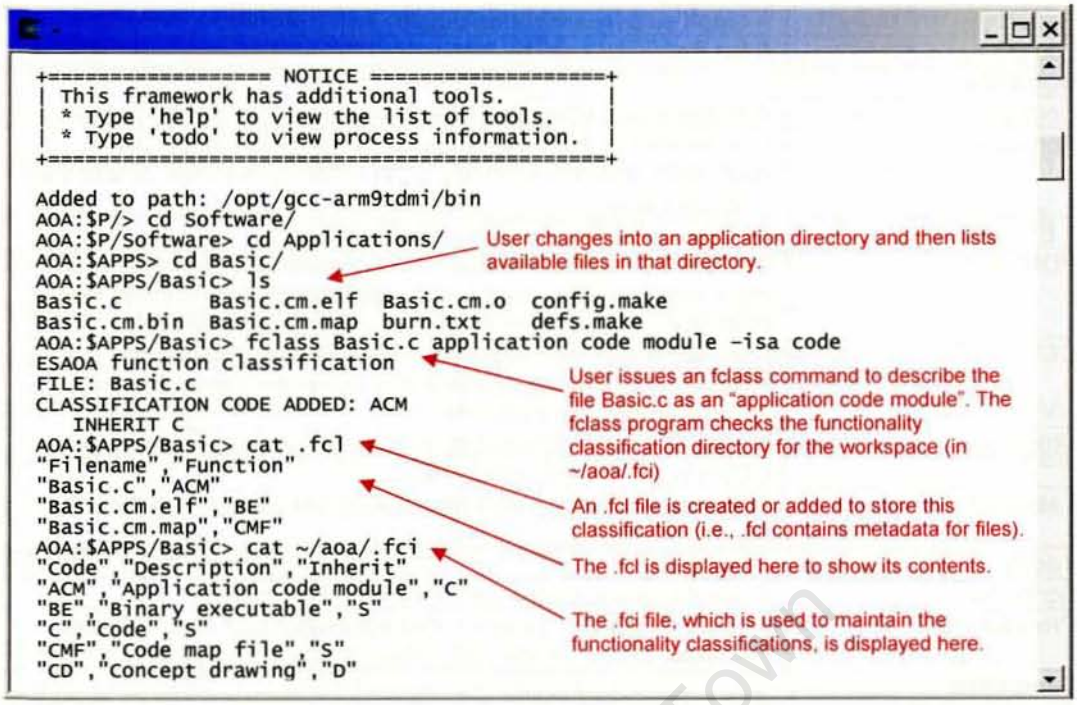


Figure 4.15: Screenshot showing use of the *fclass* program.

4.6.5.3 Artefact role and workspace classifications

Artefacts in an ESAOS team workspace or personal workstation are assigned role classifications according to the roles responsible for maintaining them (see Figure 4.16). For example, *DS artefacts* are usually maintained by the DS, *PE artefacts* are adapted by the PE, and so on. Table 4.32 lists examples of artefacts and their role classifications (the list of artefacts are largely based on artefacts found in Experiment 1 project repositories). Generally, the plan is that most downloaded documentation and data files used for experiments will be maintained by the DS. Code samples and initial code or design solutions will be maintained by the PE (as per the description of this role). Many of the artefacts (such as code files) may start of as being maintained by the PE (i.e., while attempting to establish effective development procedures), and these working files will then be handed over to the IE to take further. The CKO and the CKS will jointly be responsible for managing communal artefacts (thus Figure 4.16 does not show any artefacts exclusive to the CKO or the CKS).

Workspace classifications are assigned according to where the artefacts are stored. The three workspace classifications are: 1) communal workspace artefacts, 2) team workspace artefacts, and 3) workstation artefacts. These classifications are implicit, based on the workspace in which a particular artefact resides.

University of Cape Town

Table 4.32: List of commonly applied role classifications.

Artefact Description	Role classification
Component datasheets, e.g. datasheet for a microcontroller or analogue to digital converter device.	DS artefact
Manuals for a development tools (e.g., GCC user manual)	PE artefact
Working design document for the prototype being built	IE artefact.
Concept diagrams or concept sketches of the prototype	IE artefact
Status reports	TL artefact
Logs	DS or TL artefact, usually shared
Sample code, experimental code files used for testing components and interconnections	PE artefact
Platform deployment modules (PDMs), such as drivers modules, used to connect embedded software application code to devices on the hardware platform.	PE artefact
Final version of embedded software binary executable	IE artefact
Common component wrapper (CCW) interface files and stub code modules (SCMs).	IE artefact
Utility modules (platform-independent utility modules, e.g. C string processing routines)	PE artefact
Test case documentation or testing files	PE artefact or IE artefact
Scripts and programs (i.e., to supplement the ESAOA tools and automate manual tasks)	PE artefact
Template files (e.g., C code file templates, readme file templates, etc.)	DS artefact
Data files (e.g., images, captured signals used for testing the system)	DS artefact

Artefacts in the communal workspace are referred to as *communal artefacts*, and they are organised and updated by the CKS and the CKO. *Team workspace artefacts* are located in a team workspace, and are maintained or adapted by members of a development team (i.e., the DS, PE and IE). *Workstation artefacts* include all artefacts in the personal workspace of the workstation, in addition to the tools, such as software programs (e.g., secure shell clients, X-Windows clients and OpenOffice), installed on the computers; these artefacts are maintained by the user of the particular workstation.

University of Cape Town

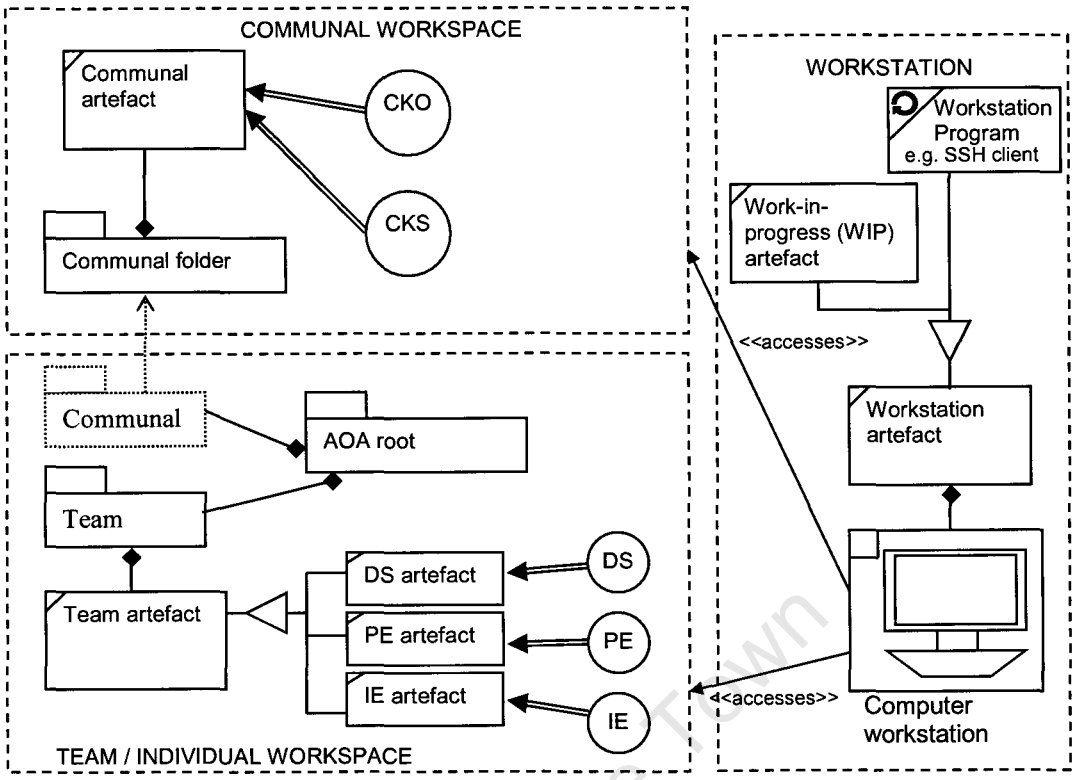


Figure 4.16: Model describing the classification of ESAOA artefacts.

In ESAOA version 1, all roles started with the same set of workstation artefacts, and then proceeded to modify and add to those initial artefacts. Section 3.11 explains the meaning of the various types of artefact atoms shown in Figure 4.16.

4.6.5.4 Artefact organisation

Artefacts are located within an ESAOA workspace according to their functionality classification, having been placed in directories or subdirectories named according to higher-level artefact functionality classifications (see hierarchy excerpt in Table 4.31). Artefacts within the 'Misc' top-level classification are not collected into subfolders. The ESAOA directory structured provided by the ESAOA team distribution (see Section 4.6.8) provides a starting point for soft artefacts used by a team, and they are arranged according to the described approach. Similarly, the ESAOA communal distribution provides an initial structure for a communal workspace, which can be further extended by the CKO and CKS. The ESAOA workstation distribution includes programs and URL links to download sites, to assist in setting up workstations; each personal workspace starts as a copy of a team workspace.

The ESAOA directory structure for soft artefact in the communal workspace is shown in Figure 4.17, and the structure for the team workspace is given in Figure 4.18.

University of Cape Town

The ESAOA team workspace starting point is a subset of the ESAOA communal workspace. It is intended to install the team workspaces and the communal workspace on the same central server computer, both being accessible from a workstation via network file sharing.

```
+-/usr/local/ESAOA-1.0/ (ESAOA communal artefacts)  ← project root
  +-Software/                ← root for software artefacts
  | +-Templates/             ← communal templates directory
  | +-Applications/          ← application code modules
  | | +-Test/                ← first application code directory
  | | +-HelloCPP/            ← second application code directory
  | +-Utils/                 ← directory pool for utility code modules
  | +-CCW/                   ← directory pool for "common component wrappers" (CCWs)
  | +-PDM/                   ← directory "platform deployment modules" (PDMs)
  | | +-Platform/            ← template PDM directory for "Platform" platform
  | |   +-Variant/           ← template PDM directory for a variant of "Platform"
  | | +-CSB337/              ← directory containing PDMs for CSB337 platform
  | |   +-MicroMonitor/      ← PDMs for MicroMonitor variant of CSB337 platform
  | |   +-uCLinux/           ← PDMs for uCLinux variant of CSB337 platform
  | +-Documentation/         ← Software documentation
  |   +-MicroMonitor/        ← Documentation for the MicroMonitor O/S
  |   +-gnu-assembler/       and related tools / drivers
  |   +-uCLinux/             ← Documentation for the uCLinux O/S
  |     +-MemDevInfo/        and related tools / drivers
  +-Tools/                   ← Communal ESAOA tools (not toolchains for compilers, etc)
  | +-Scripts/               ← Communal ESAOA Bash scripts (e.g., dt script)
  | +-Programs/              ← Communal ESAOA C++ programs (e.g., mm program)
  +-Hardware/                ← root for artefacts most strongly related to hardware
  | +-Documentation          ← Communal Hardware documentation
  |   +-Datasheets/          ← Component datasheets
  |   | +-CSB337/  ← Directory for specific component uses the component's ID
  |   | +-AT91RM9200/
  |   | +-Peripherals/← Directory for various peripherals (subdirectories not shown)
  |   | +-ARM_920/
  |   +-Overview/  ← Product overview sheets / brochures
  |   +-QuickRef/  ← Quick reference manuals (including ESAOA quick reference)
  +-Copyright/               ← Copyright / licensing information for communal artefacts
  +-Documentation/           ← Other communal documentation
    +-Forum/                 ← Documentation about using the forum
  +-Templates/               ← Other templates
```

Figure 4.17: ESAOA version 1 communal distribution directory structure.

The team workspace contains directory links (or shortcuts in Microsoft Windows™) named 'Communal' that are used to incorporate read-only communal artefacts into the team directory structure. For example, the documentation directory contains a directory link called 'Communal' (see Figure 4.18) that adds communal documentation to the team's documentation directory. ESAOA support tools, such as the *find* and classification tools (e.g., *fclass*), work across these soft links for more rapid traversal of shared files (these tools are described in Section 6.4).

University of Cape Town

```
+-ProjectX/ (ProjectX initial team workspace)  ← project root
  +-Software/                 ← root for software artefacts
  | +-Templates/              ← templates directory
  |   +-Communal/             ← soft link to communal templates
  |   +-Team/                 ← templates produced by team
  | +-Applications/           ← application code modules
  | | +-SimpleApp/            ← application code directory including sample application
  | +-Utils/                  ← directory for utility code modules
  | +-CCW/                    ← directory "common component wrappers" (CCWs)
  | +-PDM/                    ← directory "platform deployment modules" (PDMs)
  | | +-CSB337/               ← directory containing PDMs for CSB337 platform
  | |   +-MicroMonitor/       ← PDMs for MicroMonitor variant of CSB337 platform
  | |   +-uCLinux/            ← PDMs for uCLinux variant of CSB337 platform
  | +-Documentation/          ← Software documentation
  |   +-Communal/             ← Soft link to communal software documentation
  |     +-MicroMonitor/       ← Documentation for the MicroMonitor O/S
  |     +-gnu-assembler/        and related tools / drivers
  |     +-uCLinux/            ← Documentation for the uCLinux O/S
  |     | +-MemDevInfo/         and related tools / drivers
  |   +-Team/                 ← Supplementary documentation obtained by team
  +-Tools/                    ← Supplementary tools developed by team
  | +-Scripts/                ← Supplementary ESAOA Bash scripts
  | +-Programs/               ← Supplementary ESAOA C++ programs
  +-Hardware/                 ← root for artefacts most strongly related to hardware
  | +-Documentation           ← Hardware documentation
  |   +-Communal/             ← Communal hardware documentation
  |     +-Datasheets/         ← Component datasheets
  |     | +-CSB337/           ← Directory for specific component uses the component's ID
  |     | +-AT91RM9200/
  |     | +-Peripherals/      ← Directory for various peripherals (subdirectories not shown)
  |     | +-ARM_920/
  |     +-Overview/           ← Product overview sheets / brochures
  |     +-QuickRef/           ← Quick reference manuals (including ESAOA quick reference)
  |   +-Team/                 ← Team hardware documentation
  +-Copyright/                ← Copyright / licensing information for the project
  +-Documentation/            ← Product / other documentation
    +-Communal                ← Communal documentation
      +-Forum/                ← Documentation about using the forum
  +-Templates/                ← Other templates
```

Figure 4.18: ESAOA version 1 team distribution directory structure.

4.6.5.5 Specialised KM artefacts

Three types of drawings that provided graphical means of communicating and capturing ESAOA knowledge were observed in Experiment 1 and added to the KMS; these drawings are named: 1) artefact organisation diagram (AOD), 2) concept sketch, and 3) concept drawing. The developers did not originally use these specific names for their diagrams. The names 'concept sketch' and 'concept drawing' are based on the literature (specifically on work by Healey *et al.* [2002] related to

University of Cape Town

graphical communications techniques). The name 'AOD' was assigned by the researcher. Functionality classifications of the same names were added to the ESAOA KMS as a means of keeping track of these *boundary artefacts*.

Concept sketches are intended to be drawn quickly, and are thus generally pen or pencil on paper. The concept drawing captures one or multiple concept sketches including additional details. The concept sketches were informal, and used in an impromptu manner by the developers; the concept drawings were used mainly during meetings, and were thus more formal and professionally prepared. Both drawings were primarily used to exchange and capture ideas. The concept sketches in particular were used much like a whiteboard, and were modified and annotated to reflect ideas and decisions made during discussions. Figures 4.19 and 4.20 respectively illustrate a concept sketch and a concept drawing, both from Project P1-2. Figure 4.20 is based on the concept sketch shown in Figure 4.19 that was used to explain the pedestal control motor.

In ESAOA KMS version 1, concept sketches and drawings are given a title and dated (an item the P1-2 engineers neglected to do). The ESAOA team workspace distribution includes an OpenOffice Drawing template for concept diagrams.

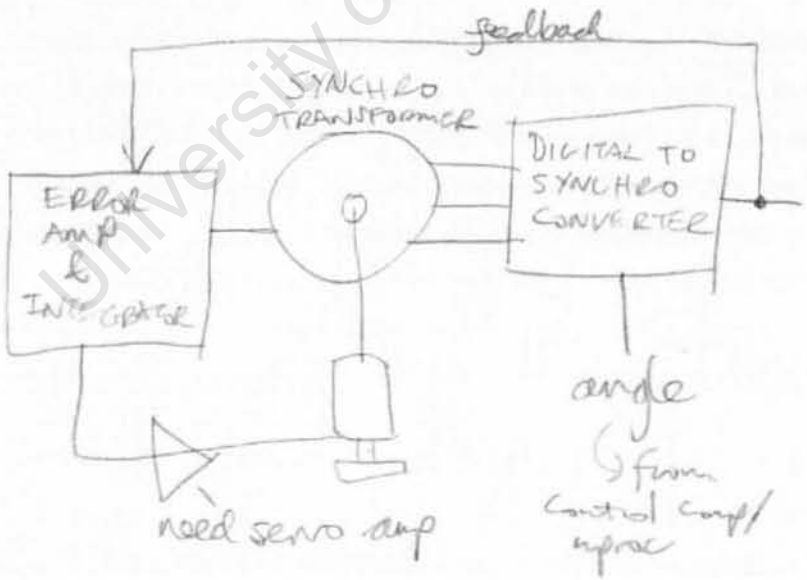


Figure 4.19: Concept sketch created in a project P1-2 meeting (event chain 44).

University of Cape Town

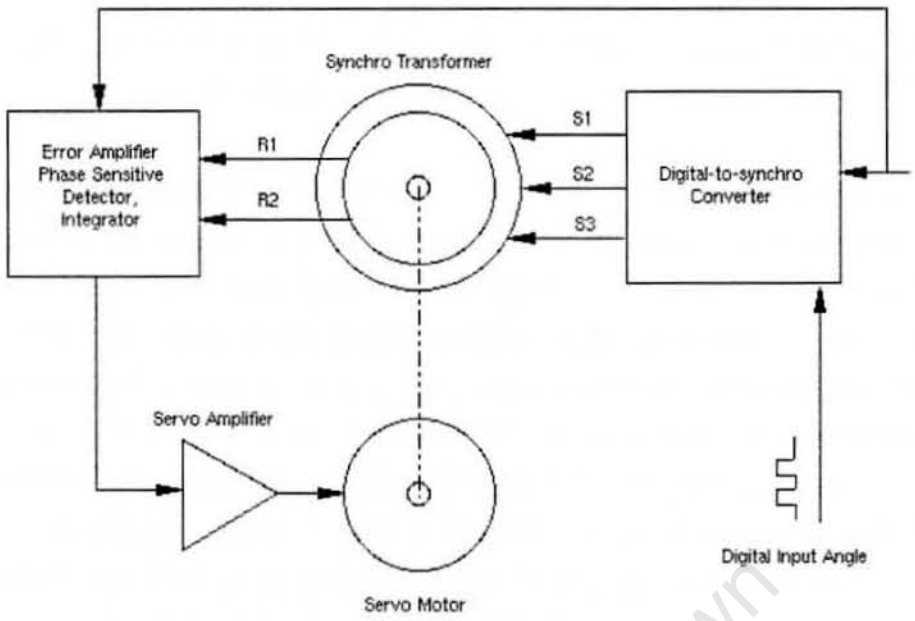


Figure 4.20: Concept drawing produced in project P1-2 (part of event chain 44).

An AOD was used in project P1-1 to express the directory structure of the software framework constructed by the engineers. The AOD built on the UML packet diagram [Arlow, 2005]. The AOD structure produced in P1-1 was extended and adapted for use in ESAOA workspaces to describe the following: 1) the starting point (or starting artefact) in an ESAOA workspace, 2) the organisation of directories and subdirectories, 3) important artefacts (e.g., executable program images), and 4) important interrelations between artefacts (e.g., datasheets used while writing certain code modules). Figure 4.21 demonstrates an AOD based on the Project P1-2 directory structure. The team workspace distribution contains an AOD that describes its directory structure.

University of Cape Town

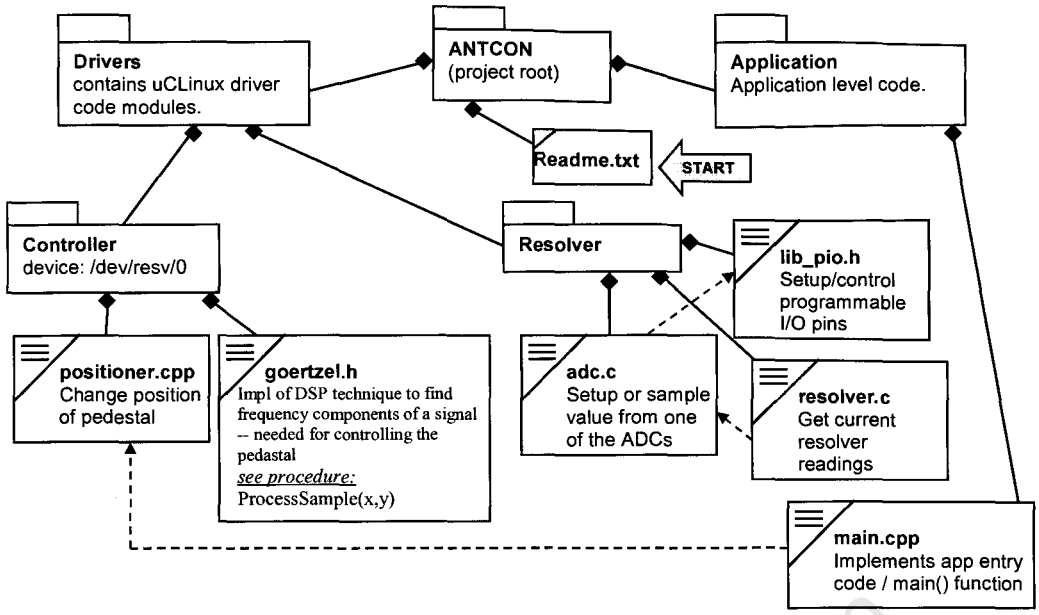


Figure 4.21: AOD for project P1-2 (see Section 3.11 for modelling language).

The Project P1-2 developers also constructed a simple embedded software simulation environment, essentially a 'virtual platform' [Wikipedia, 2009], which they used to test C code. This simulation environment composed a C code modules that were linked with the application code produced. This simulation environment was later overhauled by the research, integrated into the workspace of ESAOA KMS version 1, and was termed the *PC Box* platform. This PC Box platform was used in Experiment 2 to train participants on the use of ESAOA tools and workspaces.

4.6.6 ESAOA KM workflows and processes

This section is separated into subsections that describe the major processes performed by each role. Many of these processes relate to artefacts described in Section 4.6.5 and utilise ESAOA tools (provided in the ESAOA communal distribution and listed in Section 4.6.7). In some cases, different roles performed the same processes; these processes are described in the 'common processes' sections.

4.6.6.1 Processes of the chief knowledge officer (CKO)

The CKO is in charge of the ESAOA KMS. The CKO requires an understanding of KM, should be a skilled computer engineer, a good trainer, have experience in software engineering, and be excellent at programming ANSI C, C++ and Bash scripts. The CKO provides two fundamental influences: 1) guiding engineers in effective KM strategies within the specialised area of embedded system development, and 2) maintaining the overall structure of the ESAOA KMS, ensuring that the processes are applied consistently and that the support structures (i.e., the

University of Cape Town

CKS, ESAOA workspaces and associated computing infrastructure) are operating effectively. Figure 4.22 models the processes performed by the CKO and the roles related to these processes. The figure shows that the CKS can act as a CKO proxy, handling some of the requests submitted by team members. Generally, the CKS supports the CKO in maintaining ESAOA workspaces once team members have been trained.

The CKO should be involved in the code and design reviews performed by teams in order to provide expert advice relating to KM strategies. A principal duty of the CKO is to ensure that users of the ESAOA KMS understand the roles and associated processes they are designated to perform (an issue that can be checked effectively during code and design review meetings).

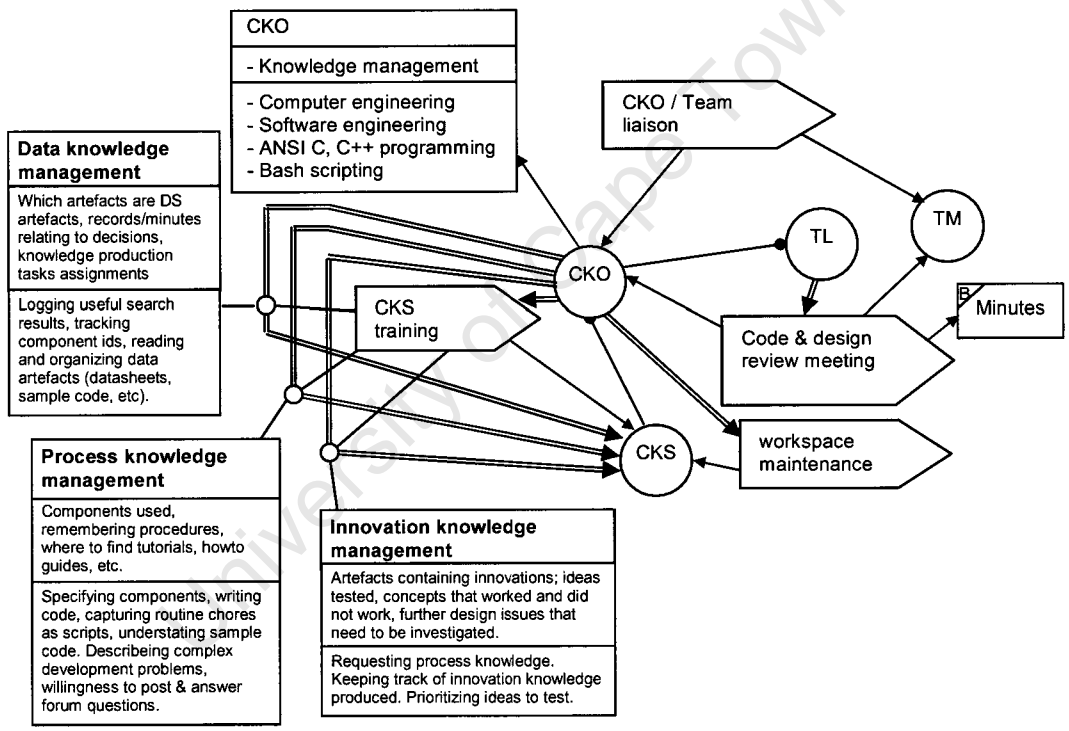


Figure 4.22: Processes performed by the CKO.

The CKO is responsible for training the CKS. This training ensures that the CKS is in turn able to train team members in performing the roles of DS, PE and IE. In Experiment 2, there was one CKS, and he was trained by observing how the CKO assisted the first couple of teams in learning their roles. Refinements to the duties of the CKO in version 2 of the KMS are explained in Section 6.6.1.

University of Cape Town

4.6.6.2 Processes of the communal knowledge steward (CKS)

The CKS acts as a proxy and assistant to the CKO. The CKS facilitates sharing of knowledge produced in one team with other teams, and is tasked with maintaining the communal workspace. The main responsibilities of the CKS are: 1) checking that team members are utilising the ESAOA KMS, and 2) adding or modifying communal artefacts. In some cases, and with the permission of the team concerned, the CKS will copy artefacts (and new functionality classification index entries) from a team workspace to the communal workspace, while maintaining the appropriate artefact classification structure. The CKS also suggests corrections to mistakes made in classifying artefacts. Figure 4.23 modelled the main processes performed by the CKS. Once the team members have been trained in their roles, the CKS focuses on supporting the CKO and team leaders in maintaining the communal workspace and team workspaces respectively.

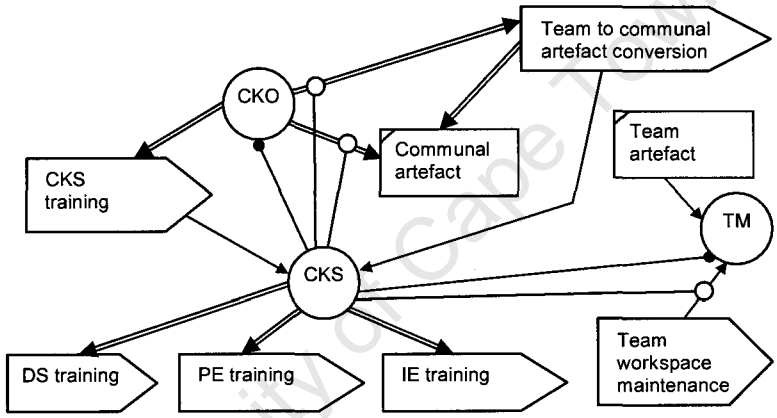


Figure 4.23: Major processes performed and maintained by the CKS.

4.6.6.3 Processes of the team leader (TL)

One of the members of each development team is assigned the role of team leader. This individual performs the team leader role in addition to roles of DS, PE, or IE as needed during the development. In ESAOA KMS version 1, only one process is defined for the team leader, specifically: the team leader reports to the CKO and CKS on the successes or problems encountered in using the ESAOA KMS (the CKO or CKS respond by providing support in terms of recommending KM strategies). Figure 4.24 models this process of the team leader. Other processes of the team leader are not defined in the ESAOA KMS; rather, the team leader decides on his or her own methods to manage the team, such as scheduling progress meetings, reminding team members of deadlines, and ensuring that each team member is doing their bit. The dashed lines in Figure 4.24 indicate that these team management processes are

University of Cape Town

part of the ESAOA KMS (i.e., they are external processes thoroughly covered by the literature, such as Thamhain [1990]).

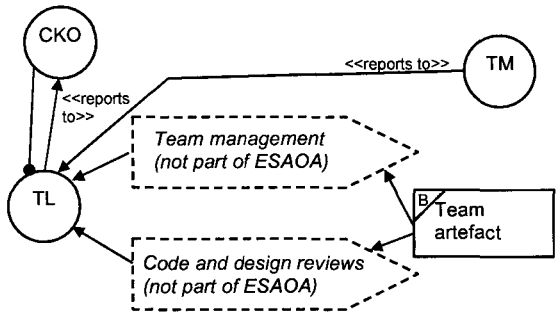


Figure 4.24: Major role interrelations and processes performed by the TL.

4.6.6.4 Processes of the data steward (DS)

The DS is mainly responsible for finding, reviewing and organising documentation and sample code used by the PE. The DS also maintains other files related to data knowledge produced by the team, such as test cases used for regression testing.

Skills required by the DS include effective search strategies (e.g., use of Google keywords) and knowledge of useful information sources (e.g., books and online forums) from which data knowledge can be gleaned rapidly. The DS should be competent in keeping records and taking notes at meetings so that important knowledge produced by the team is made explicit, and thus retained by the team.

As is the case for all the roles, the first process performed by the DS is training, specifically 'DS training' in which the CKS or CKO introduces team members to the production and management of data knowledge (Figure 4.25 (a)). Since the team members are all likely to perform each of the DS, PE, and IE roles at some stage during a project, all the teams members should receive training for all these roles.

The CKO maintains and documents the DS training process and teaches this process to the CKS. The CKS provides support for the DS on using the processes learned during DS training. Much of this training is tacit, with the CKS showing the DS effective strategies for web searches, methods of logging search text and URLs, and how and where to save datasheets and other information documents to the team workspace. The CKS also runs through the ESAOA tools (e.g., demonstrating use of the *fclass* and other tools), and explains the various artefacts, and their

University of Cape Town

classifications, which the DS is likely to find and use in a project (Section 4.6.7 details the scripts and programs of the ESAOA KMS version 1). Figure 4.25 (b) models the processes and artefacts used by the DS.

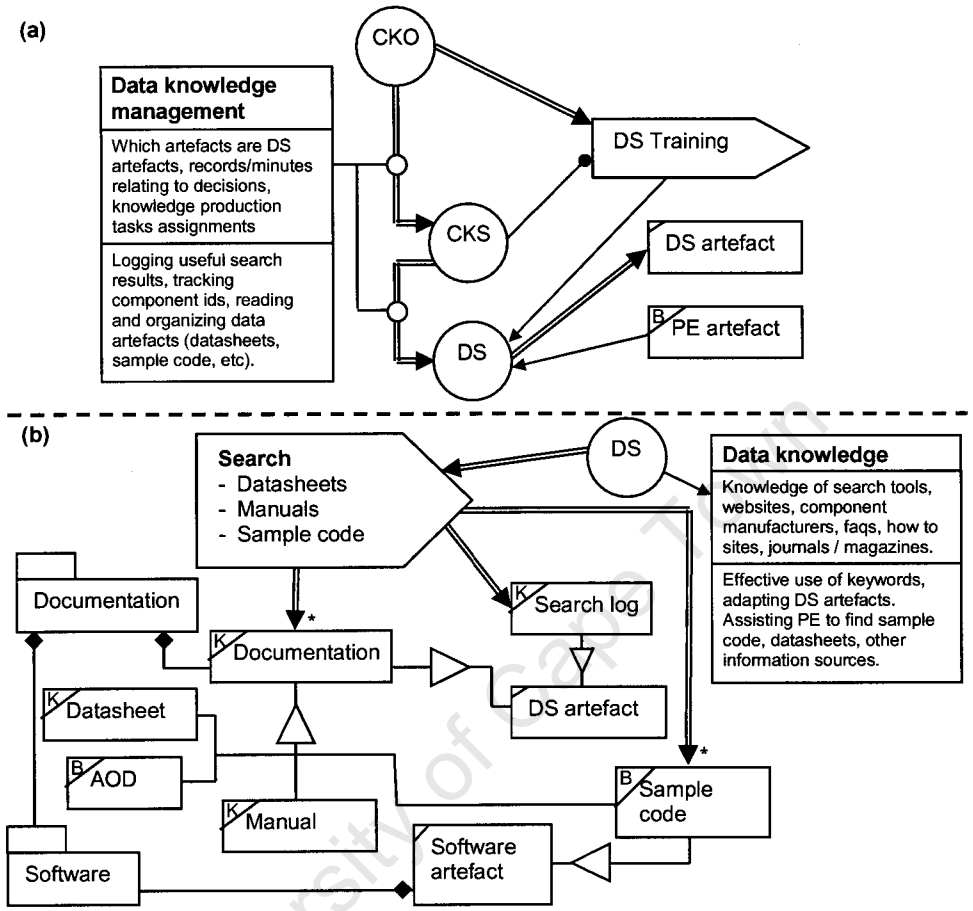


Figure 4.25: (a) DS training process; (b) DS search process.

4.6.6.5 Processes of the process engineer (PE)

The process engineer creates development procedures, such as figuring out the configuration settings for an operating system, or controlling a specific hardware component from software. The IE uses procedures created by the PE in order to test design concepts. Both the PE and IE are involved in experimentation. However, the difference between their experiments is that the PE attempts to determine methods of configuring and controlling components and tools that will be used in the system prototype, whereas the IE focuses on testing design ideas by using processes provided by the PE and thus exerts little additional effort in terms of developing methods to configure tools or interface components. The PE requires expertise in designing and testing development methods, as well as being able to keep memory

University of Cape Town

joggers and component lists that will help the PE or IE to repeat these processes at a future time. The PE may also add or customise ESAOA tools within the team workspace. Figure 4.26 models the processes and artefacts used by the PE.

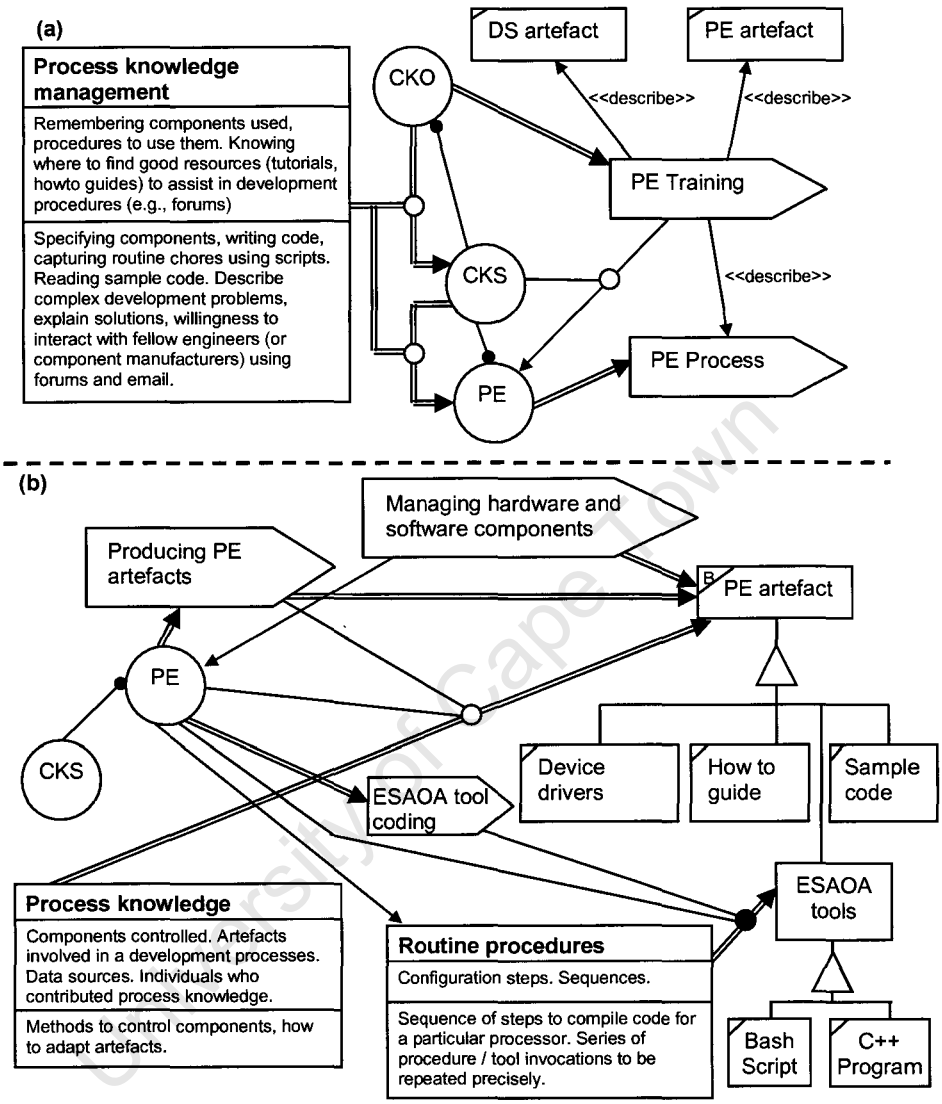


Figure 4.26: (a) Interaction between PE, CKO and CKS; (b) Processes performed by the PE that involve development process knowledge.

4.6.6.6 Processes of the innovation engineer (IE)

As mentioned in Section 4.6.4, the IE tests ideas to produce innovation in the project. Section 2.5.6 of the literature reviews theories related to innovation management; additionally, properties of individuals involved with innovation in development projects are investigated by authors such as Nonaka & Takeuchi [1995], Leonard [1999], McKeown [2008]. Consequently, the IE role is expected to exhibit qualities of: creativity, expression and experimentation. In addition, for effective use of the

University of Cape Town

ESAOA KMS and effective collaboration between the IE, PE and DS roles, the IE needs a willingness to test design ideas, coupled with the ability to explain to the PE the development procedures, related components and artefacts needed in order to carry out experimentation tasks from which innovation knowledge is built.

The PE and IE are likely to spend a significant amount of time working side-by-side, with the PE explaining to the IE how to carry out development methods, and the IE requesting from the PE additional development strategies or modifications to existing processes. Much of the interaction between the PE and IE is likely to be rapid and performed face-to-face, and thus difficult to capture and record. For this reason, the PE is involved in capturing solution strategies and memory joggers that will assist in recovering or remembering development strategies. The IE will generally learn the development strategies from the PE. However, the IE will generally not obtain the detailed process knowledge, which the PE learns, relating to how the development procedures are produced and maintained. The IE will simply learn which scripts and configuration files to use, and which tool commands (or keystroke sequences) to use, as explained by the PE. The processes and artefacts discussed here are modelled in Figure 4.27.

University of Cape Town

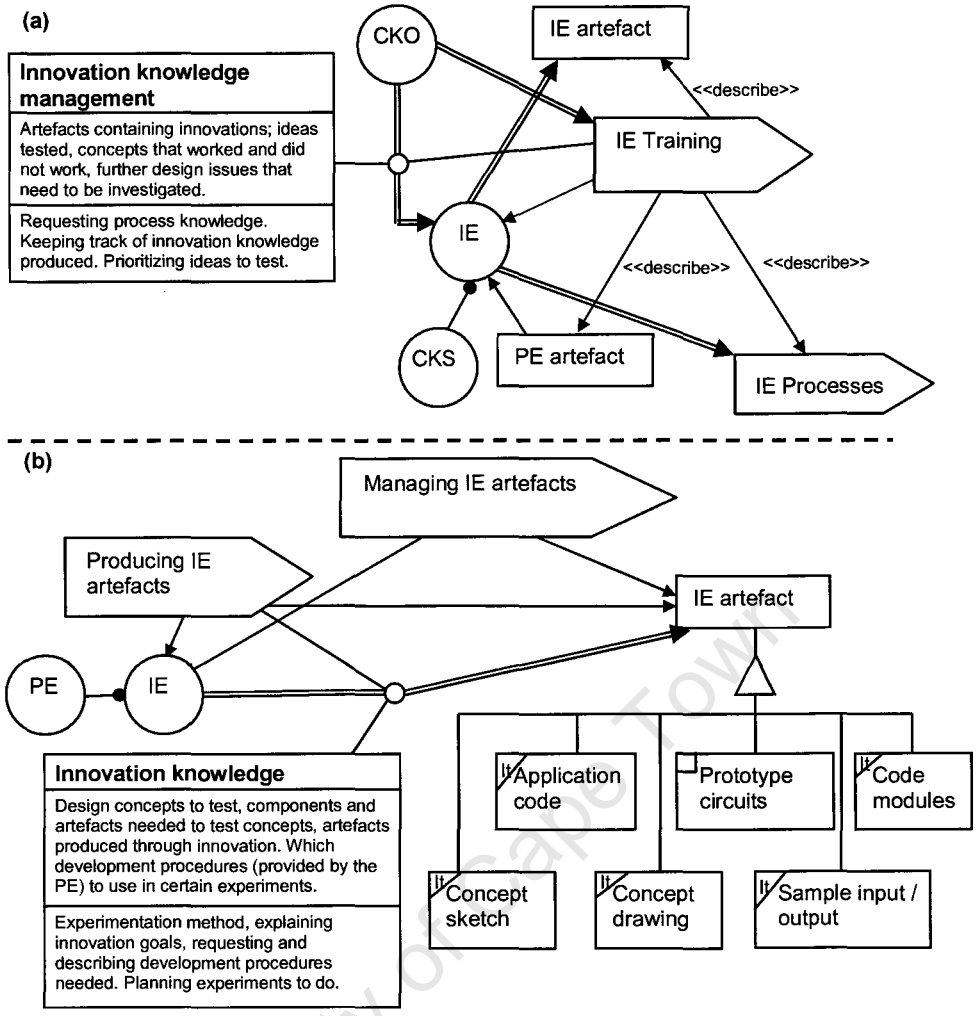


Figure 4.27: (a) Collaboration between CKO, CKS and IE; (b) Processes and artefacts used and managed by the IE.

University of Cape Town

4.6.7 Software design of ESAOA workspaces

The ESAOA workspaces comprise a directory structure of artefacts and KM support tools. The tools comprise Bash scripts (called *scripts*) and C++ executable programs (termed *programs*). The scripts and programs are listed in Section 4.6.7. The initial versions of the tools were all provided in the ESAOA communal distribution, and teams could modify them, or add additional tools, as needed for their own projects. The design of these workspaces became the main aspect of the framework construction phase described in research design (see Section 3.5.1). As mentioned in Section 3.5.1 the tools and other artefacts for ESAOA KMS version 1 were based on analysis of Experiment 1, and involved the adaptation of artefacts produced by the developers doing the experiment, or the addition of new artefacts developed by the researcher.

4.6.7.1 ESAOA scripts and tools

Many of the scripts simply call programs, and are often used as a means of calling the programs without specifying many, if any, command line arguments, which saves time. For example, the *aod* script changes to the directory containing AOD documents (i.e., directory 'Documentation/AOD' as per the functionality hierarchy in Section 4.6.5.2), and then displays the latest AOD drawing. ESAOA KMS version 1 has 40 initial scripts. Figure 4.28 shows the Bash code used to implement the *esaoa-snap* script. This code calls ESAOA programs (such as *esaoa-fm*) to perform its operation. The following scripts were of particular importance:

- **Build-esaoa:** compiling and linking a new ESAOA program.
- **Esaoa-cleanup:** removing various unimportant files, such as map files and object files, allowing backups to take less storage space.
- **Esaoa-create-script:** generating an ESAOA script starting point in the appropriate directory edit.
- **Esaoa-find:** using a combination of the GNU *find* utilities [GNU, 2008b] and the *fcs* program to access file metadata and find artefacts.
- **Esaoa-snap:** creating a zip file snapshot of a directory and its subdirectories.
- **Esaoa-tz:** creating a backup of a project directory (similar to esaoa-snap).
- **Godir:** similar to *esaoa-find*, it looks for a directory and changes the current working directory to the first directory found.

There are 34 initial programs in ESAOA KMS version 1. The most important programs are outlined below; these programs are often called from scripts (for example, the *esaoa-fm* and *fcs* programs were used in many scripts).

- **Esaoa-fm:** manipulating filenames (e.g., extracting the file extension).
- **Esaoa-menu:** used by scripts to present a menu to the user, in order to capture commonly performed actions.
- **Esaoa-mm:** traverses ESAOA directory structure to generate make files.
- **Fclass:** used for maintaining classification data of soft artefacts (i.e., assigning and viewing classifications applied to files in a workspace).
- **Fcs:** maintaining file metadata (e.g., artefact classification data) using CSV files. This program has a variety of command line arguments and is built into the *fclass* program.
- **Fim:** managing the functionality index classification file (i.e., used to add artefact functionality classifications).

University of Cape Town

Figure 4.29 provides the initial lines of the esaoa-fm.cpp code file that implements the *esaoa-fm* program. The code shows use of the KIT API, an API that was developed to facilitate construction of the ESAOA programs, such as *esaoa-fm*.

```
#!/bin/bash
# Script to create a zip snapshot of a directory, but does not
# add compiled objects files or executables.
# Part of ESAOA framework
pushd . >/dev/null   # Store the current path
DIR=$(esaoa-cwd)
# Determine relative path of the directory to 'snap'
APPDIR=$(esaoa-fm $DIR n)
cd ..
# Generate a new archive name
ARCH=$(esaoa-fm $APPDIR.zip u)
# Create the archive
echo "--------------------CREATING ARCHIVE--------------------"
# Display the ESAOA EXCLUDES environment variable
echo EX = $EXCLUDES
# Use zip command to generate the archive
zip -r $ARCH $APPDIR -x $EXCLUDES
echo "--------------------------------------------------------"
ls -lh $ARCH  # display size of the snap archive
popd . >/dev/nulls
```

Figure 4.28: Implementation of the esaoa-snap script.

```
/** @file esaoa-fm.cpp
 * Filename manipulation application.
 * Programmer: S. Winberg
 * Date      : 20 Jan 2005 */
/* Standard C includes */
#include <stdio.h>
#include <string.h>
#include <time.h>
/* KIT (Kit of IT tools) API includes */
#include "KitFile.h"
#include "KitStream.h"
#include "ESAOA.h"
#include "Kit_CONSOLE.h"

int verbose = 0; // set to 1 to enable debug messages
int is_console_app = 1; // if 1: use console; else GTK calls
/***************** FUNCTION IMPLEMENTATION ****************/
/** Function to provide help information */
void help () {
 fprintf(stderr,"ERROR: Please specify a file name\n");
 ...
```

Figure 4.29: Implementation of the esaoa-fm program.

University of Cape Town

4.6.7.2 The Kit for Information Technology (KIT)

A C++ application programming framework, called the Kit for Information Technology (or KIT), is included in the ESAOA communal distribution. KIT provides high-level functions for navigating ESAOA directory structures and for manipulating CSV files that contain meta-data for files. The KIT application programming interface (API) is used to connect a C++ program to the KIT application framework. The communal workspace includes a description of the KIT API with code examples. The UML class model for KIT, and a description of its classes, is provided in Appendix C.4 (the appendix includes a flowchart and code snippets of a C++ ESAOA program implemented using KIT).

4.6.7.3 The central server and the networking infrastructure

The communal workspace and team workspace (which were introduced in Section 4.6.2) are used to store and share artefacts. Artefacts of the communal workspace are shared between all teams, but artefacts of a team workspace are shared only between members of that team. The workspaces also constitute an environment that includes artefacts structured among directories and ESAOA tools to assist ESAOA tasks. A central server, running the Linux operating system, is used to meet these needs. The central server provided one common workspace, utilised by all teams, which was overlaid by a particular team member's team workspace, which was in turn overlaid by that team member's personal workspace. This method of overlaying workspaces was achieved by loading one Bash script environment configuration over another (with subsequent environment settings overwriting those of previous settings). Figure 4.30 describes this approach. This approach was mainly used in Experiment 2, in that ESAOA tools were accessed using remote login from lab computers. However, if a team member wanted to work 'off line' on his or her own computer (i.e., not logging in to the central server), a copy of the communal workspace and appropriate team workspace, in addition to the ESAOA tools and other development software, had to be installed on that computer.

University of Cape Town

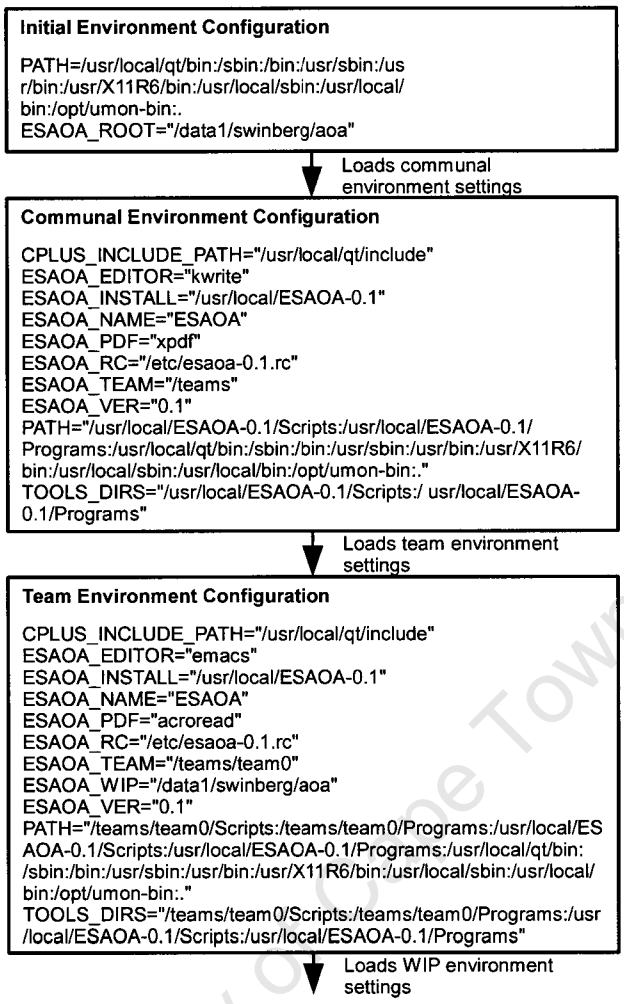


Figure 4.30: Consecutive loads of Bash environments for ESAOA workspaces.

4.6.8 Implementation and distribution of ESAOA workspaces

The ESAOA workspaces are put into place using ESAOA distributions. These distributions are in the form of archives that include soft artefacts organised into an ESAOA directory structure. The artefacts also include documents that describe roles, processes (captured in the form of guideline documents, how to manuals, scripts and programs) and other artefacts that collectively describe a baseline version of an ESAOA KMS. The distributions also provide template artefacts that are used by the roles described in Section 4.6.4. There are three types of ESAOA distributions, which are summarised in Table 4.33.

The ESAOA communal workspace distribution comprises 302 files in 26 directories. The team workspace distribution has 163 files in 27 directories. The workspace distribution consisted of 31 files in 9 directories (mostly a collection of installation

University of Cape Town

programs to be installed on the workspace computer if alternate applications have not already been installed – these software applications include free or open-source software programs, such as Putty.exe [Tatham, 2009]). Note that there is no personal workspace distribution; it is assumed that individual team members will start with a copy of their shared team workspace. Furthermore, only the team workspace distribution has an installation script (although it requires some manual intervention). The other workspaces and workstation distributions need to be manually installed.

Table 4.33: ESAOA KMS version 1 distributions.

ESAOA Distribution	Archive file	Contents
ESAOA communal workspace distribution	ESAOA-communal-1.0.tar.gz	Communal soft artefacts maintained by the CKO and CKS and used by all teams.
ESAOA team workspace distribution	ESAOA-team-1.0.tar.gz	A baseline collection of soft artefacts that provides a starting point for team projects.
ESAOA workstation distribution	ESAOA-workstation-1.0.tar.gz	Programs used by all roles to access the ESAOA communal and team workspaces.

4.6.8.1 Implementation of the ESAOA communal workspace

The ESAOA communal workstation distribution provides processes in the form of programs and scripts, together with template document and code files. These artefacts are not modified directly by team members, but are maintained by the CKO and CKS. The communal workspace has the directory structure shown in Figure 4.31; the annotations describe the directory contents. Further detail about these directories is provided below.

```
+-/usr/local/ESAOA-1.0/   (ESAOA communal artefacts)  ← project root
  +-Software/          ← root for software artefacts
  +-Tools/             ← Communal ESAOA tools (not toolchains for compilers, etc)
  | +-Scripts/         ← Communal ESAOA Bash scripts (e.g., dt script)
  | +-Programs/        ← Communal ESAOA C++ programs (e.g., mm program)
  +-Hardware/          ← root for artefacts most strongly related to hardware
  +-Copyright/         ← Copyright / licensing information for communal artefacts
  +-Documentation/     ← Other communal documentation
  +-Templates/         ← Templates for documents and code
```

Figure 4.31: Top-level directory structure of ESAOA communal workspace.

The ESAOA workspaces are accessed from the ESAOA shell environment. The ESAOA shell environment extends the Bash (Bourne Again Shell) [BASH, 2009] command line environment using additional scripts and C++ programs. The scripts and programs encode process knowledge and strategies for managing ESAOA

University of Cape Town

knowledge. Tools are provided in the communal distribution for finding, cross-referencing and relating files, amongst other tasks.

Both scripts and programs can be manually invoked from a terminal within the ESAOA shell environment. Text editors and integrated development environments (IDEs) are started from within the ESAOA environment and can be configured to invoke ESAOA scripts or programs automatically. The following subsections respectively discuss the ESAOA scripts, ESAOA programs, and ESAOA templates that are provided with the ESAOA communal distribution.

ESAOA communal scripts

ESAOA scripts are used by the PE as a means of capturing *automatable*[5] process knowledge that can be expressed effectively by a scripting language. Bash scripts were chosen for the ESAOA KMS, as these scripts can be coded swiftly because they follow a simple programming construct that can be learned quickly. A more advanced and powerful scripting language (e.g., Python [Lutz, 2006]) was not used, as these languages generally take longer to learn. The Bash scripting language was found to be an effective means of capturing routine process knowledge that occurred in Experiment 1 (e.g., the steps to netboot an embedded operating system that occurred in project P1-2 event chain 28). ESAOA scripts are divided into communal scripts (provided with the ESAOA communal distribution) and team scripts (which team members created during projects). In order to override a communal script, a team script is given the same name as the ESAOA communal script. The baseline team workspace provided by the team distribution does not provide any script. The scripts included in the ESAOA communal workspace are described in Table 4.34.

University of Cape Town

Table 4.34: ESAOA communal scripts.

Script Name	Description	Used by
aod	Display AOD diagram for project	All
build-esaoa	Build C++ ESAOA program that encodes a process, adds to Scripts directory in current project.	CKO / all
check-comments.sh	Checks number of comments in a code or script file. Can display all or a summary (first line) of comments in file.	TL
desc	Describe a file (adds to projects file description CSV file)	DS
dus	Show human-readable disk usage summary.	All
enter-aoa	Enter the ESAOA environment.	All
esaoa-addpath	Add the current (or specified) path to the current project environment.	PE
esaoa-artefact	Add meta-data to artefact (stored in CSV file in local directory).	DS

[5] The term 'automatable' is used here to refer to procedure knowledge that can be effectively represented as a computer program.

esaoa-burn	Script to 'burn' a bin file in to flash on a target platform.	PE
esaoa-checks	Perform checks to verify that the ESAOA environment is operating / installed correctly.	All
esaoa-clean	Remove unneeded / temporary / object files in current directory.	DS, PE
esaoa-cleanup	Goes through application directories for project and performs an m clean operation.	DS
esaoa-cp	Copy a file and its classification: calls the Linux cp command and ESAOA's fclass tool. (version 1 cannot copy classifications across different workspaces; this must be done manually)	All
esaoa-create-c	Create a new C file using a C file template stored in the /usr/local/ESAOA-1.0/Templates directory.	PE, IE
esaoa-create-cpp	Create a new C++ file using a C++ file template stored in the /usr/local/ESAOA-1.0/Templates directory.	PE, IE, CKO
esaoa-create-script	Create a new ESAOA Bash script using a file template stored in the /usr/local/ESAOA-1.0/Templates directory.	PE, CKO
esaoa-dt	Show the date and time in a standard, concise format (uses template DD-MMM-YYYY).	All
esaoa-edit	Open the default editor to edit a code file (teams can choose to override the default editor using their own esaoa-edit script).	All
esaoa-exeapp	Execute the application – if using PCBox executes on PC, otherwise instructs embedded platform to run the program.	PE, IE
esaoa-find	Find an artefact based on keywords. Find can search based on a filename or on text within the body of the file.	All
esaoa-help	Show help for a particular ESAOA program or script provided with the communal workspace.	All
esaoa-home	Go to root folder of team or personal workspace	All
esaoa-info	Display information about the active ESAOA communal workspace and ESAOA team workspace.	DS, CKS, CKO
esaoa-m	Make current application	PE, IE
esaoa-mkrc	Make ESAOA RC file (a Bash script file executed when the user changes into the directory containing that file).	PE, CKO, CKS
esaoa-mv	Rename/move a file and moves its fclass classifications. (Need to manually moved classifications between workspaces.)	All
esaoa-nop	Does nothing – used in testing the installation.	All
esaoa-platform	Change target platform or variant of platform.	PE, IE
esaoa-project	Manage the current project.	All
esaoa-rm	Removes a file and all its classifications using fclass.	All
esaoa-scratch	Manage temporary files.	All
esaoa-term	Open another terminal.	All
esaoa-test	Test a ANSI C or C++ file.	PE, IE
esaoa-tz	Compress current directory and sub-directories as tar.gz archive.	DS
exit-esaoa	Exit from the ESAOA environment.	All
godir	Go to a certain directory (e.g., go app tries to find a directory containing the text 'app').	All
log	Add an entry to the current project's log.	All
platform-info	Display information about the currently selected target platform.	PE, IE
title	Display or set the title of the directory.	DS
tree	Display the directory tree of the current directory.	All

ESAOA communal programs

ESAOA programs are C++ applications, which are used as a means of capturing routine process knowledge that can be expressed effectively using a Bash Script. A simple C++ application framework called KIT (see Section 4.6.7.2) is included in the ESAOA communal distribution. KIT provides high-level functions for navigating

ESAOA directory structures and for manipulating CSV files containing file meta-data. Table 4.34 describes the tools provided within the ESAOA communal workspace.

The PE was responsible for any customisation or addition of ESAOA tools in the team workspace in order to capture procedures and allow them to be repeated quickly (Section 6.6.5 indicates this as a PE process). Development of C++ programs (using KIT), as a means of capturing automatable procedures, were likely to take more time than writing Bash scripts for the same purpose. However, C++ programs may be better suited for some purposes than scripts. Furthermore, the developer may be skilled in C or C++ programming, but have little experience in Bash scripting. For these reasons, the PE could choose between Bash scripts or C++ programs.

An example of an ESAOA program produced in Experiment 1 is the *ascii-table* program[6] (see Table 4.34). This simple C++ program was written to provide a quick means to find ASCII codes of characters, thereby assisting in knowledge production; consequently, it was added to the set of communal ESAOA tools.

As was the case for ESAOA scripts, ESAOA programs are divided between communal programs (provided in the communal distribution) and team programs developed by members of a team. An ESAOA communal program could be overridden by giving a team program the same name as the communal program (the ESAOA environment configured the path to accommodate this facility).

One of the most important programs in the ESAOA communal program collection was the *esaoa-mm* (or *mm*) program. This program captured the researcher's knowledge of creating make files for portable embedded applications for which the code modules were organised according to the ESAOA team workspace structure.

Table 4.35: ESAOA communal programs.

Program Name	Description	Used by roles
ascii-table	Display ASCII table or part of it.	PE, IE
esaoa-apps	Display list of applications in team workspace or go to an application.	PE, IE
esaoa-askpass	Read a password.	All
esaoa-bar	Display a status bar.	PE, IE
esaoa-capture	Capture data from target (using an RS232C port linked to the embedded platform).	IE
esaoa-cwd	Display current working directory (following symbolic links.	All

[6] The P1-1 engineer developed the *ascii-table* program because he was tired of manually looking up ASCII codes needed in testing the SoSiG communications protocol.

University of Cape Town

esaoa-defs	List or add a definition to project defs.csv database.	DS
esaoa-err	Add a line of text or file to the ESAOA error log file for indicating user concerned.	PE, IE
esaoa-fm	ESAOA filename manipulation application.	All
esaoa-godir	Used by godir script to find a directory.	All
esaoa-log	Called by log script to manage logs for a project.	All
esaoa-menu	Display and control a menu used by ESAOA scripts.	PE
esaoa-mm	MakeMake application – creates a Makefile for an application.	PE, IE
esaoa-moddep	Determine dependencies for a C or C++ code module.	All
esaoa-neaten	Neaten a C or C++ file.	DS, TL
esaoa-pathtool	Add a directory to the path in the team workspace.	PE, IE
esaoa-ppc	Use in C or C++ pre-processing (e.g., finding comments, author names, etc).	PE, IE
esaoa-projroot	Determine path for the directory root of current project.	All
esaoa-prompt	Called by Bash when in the ESAOA environment to display a command prompt.	All
esaoa-readln	Prompt user for a line of text.	All
esaoa-setup	Configure the ESAOA environment.	PE
esaoa-sm	Send / log a message to fellow team member.	All
esaoa-snap	Make zip snapshot of current directory and subdirectories.	All
esaoa-status	Add a status entry or annotate an existing status entry.	All
esaoa-tally	Count files, occurrences of certain events logged.	TL
esaoa-upload	Upload a data file to the target (using an RS232C port linked to the embedded platform).	PE, IE
esaoa-version	Display current version of ESAOA communal workspace.	All
esaoa-zip	Zip archive manager (used to manage archives stored in Project directory).	DS
fim	Functionality classification index file manager.	All
fclass	Add functionality classification for a file.	All
Fcs	Used to maintain CSV files containing meta-data for files.	All
Istime	Check if a certain date/time has been reached.	All
Penv	Modify/display ESAOA environment parameter.	PE
Sdb	Simple database application (SDB). Used in manipulating CSV files using SQL.	All

ESAOA communal templates

The ESAOA templates provide convenient starting points for certain types of artefacts stored in an ESAOA workspace. The choice of templates is based on artefacts and development procedures observed in Experiment 1. Three artefact templates of interest are: 'definitions.defs', 'projects.mnu' and 'esaoa-project.rc'.

In project P1-2, the engineers maintained a list of term definitions. In the ESAOA KMS, the *esaoa-defs* program was used to assist in maintaining definitions across projects in multiple files called 'defs.csv' (one copy in each project root folder).

The 'menu.mnu' file is a menu file template used by the *esaoa-menu* program – the menu structure is simple, in that leaf menu[7] entries invoke a command (or shell script

[7] The term 'leaf menu' refers to menu items at the lowest level of a menu hierarchy, which do not have further submenus.

containing multiple commands) and then exit the menu program. This structure was based on scripts developed in Experiment 1 that could be made more efficient by using a menu of this design instead of asking a series of yes/no questions in order to run a particular command or sequence of commands.

In Project P1-1, the engineers found it necessary to modify their shell environment (e.g., adding paths and setting environment variables). Considering that both Projects P1-1 and P1-2 used notably different shell configurations, it was decided that each ESAOA project folder, when entered using the *cd* command, should configure the shell environment according to an 'esaoa-project.rc' file placed in the project root directory. This allows for more rapid change between projects, while having the important configuration settings kept in a known place.

Table 4.36: ESAOA communal templates.

Program Name	Description	Used by roles
aoa.tar.gz	ESAOA team workspace not containing files.	All
code.c	Template C file.	PE, IE
code.c++	Template C++ file.	PE, IE
code.cpp	Template C++ file.	PE, IE
dayplan.csv	Template dayplan (for team leader) in CSV format.	TL
dayplan.xls	Template dayplan (for team leader) in Microsoft Excel format.	TL
definitions.defs	Example list of definitions (managed by esaoa-defs).	DS
esaoa-project.rc	Example project RC file.	PE, CKO
esaoa-script	Template ESAOA Bash script.	PE, CKO
menu.mnu	Example menu.	PE, CKO

University of Cape Town

4.6.8.2 ESAOA team and personal workspace

The ESAOA team workstation comprises a Bash shell environment and a directory structure. The shell environment is entered using the *enter-esaoa* command. Figures 4.32 and 4.33 show screenshots of the ESAOA shell environment accessed using a Cygwin terminal [Cygwin, 2008]. The ESAOA shell environment is configured in two steps: first, the communal workspace configuration is loaded (file '/etc/esaoa-01.rc' for version 1) and then the 'esaoa.rc' in the personal ESAOA workspace is loaded.

The CKO maintains the communal 'esaoa.rc' file, and the team maintains the top-level 'esaoa.rc' file. Following this strategy, configuration changes can be applied effectively from the CKO to all teams or from one team to other teams.

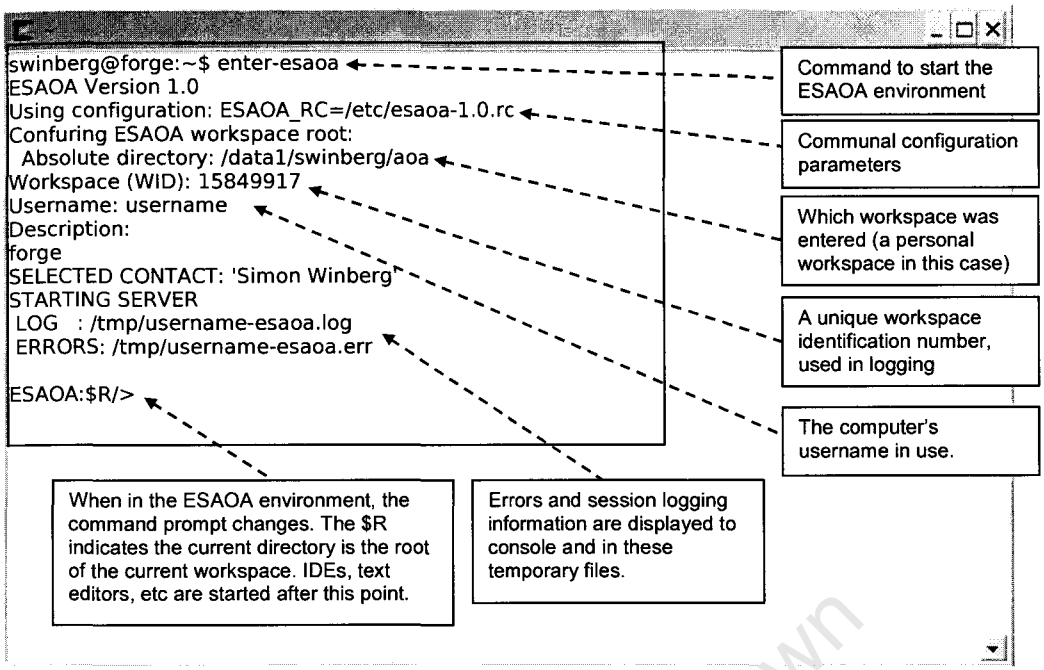


Figure 4.32: Using Cygwin to access a team workspace on the central server.

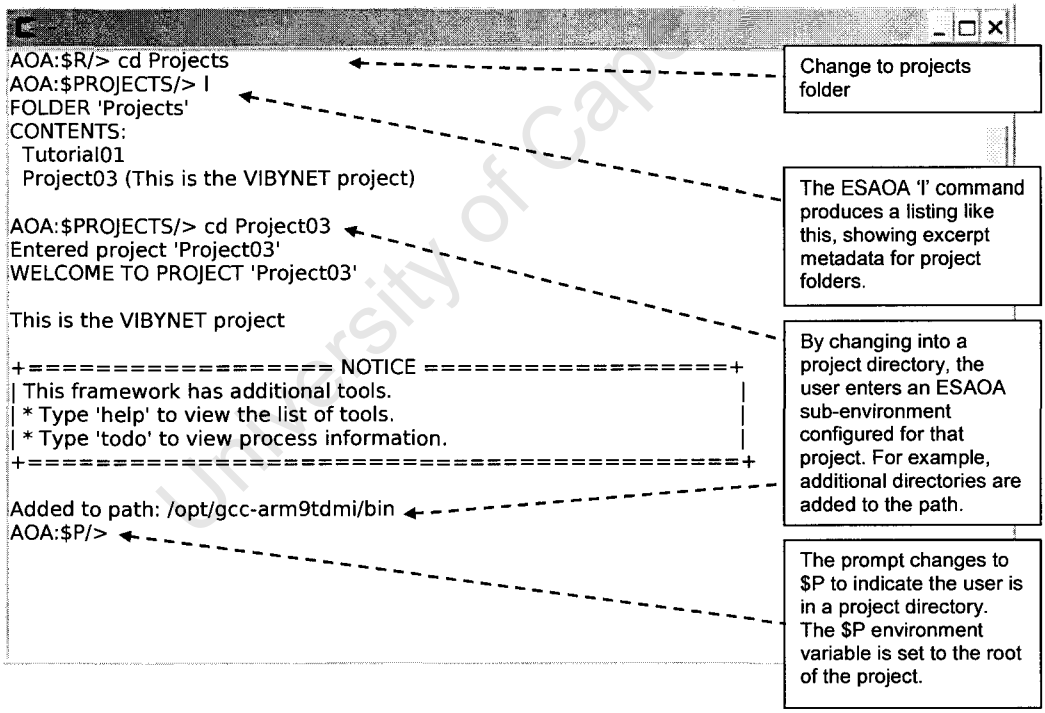


Figure 4.33: Using Cygwin to access a project in an ESAOA team workspace.

4.6.8.3 ESAOA workstation distribution

The ESAOA workstation distribution comprises a set of programs that were installed on team members' computer workstations. These programs include tools for accessing the embedded platform (e.g., uploading executable programs) and for

University of Cape Town

using the ESAOA workspaces (e.g., Cygwin and X-Windows clients). Most of these programs are open-source and freely available. Figure 4.34 shows the directory structure for this distribution.

```
ESAOA-workstation-1.0
   |- Software.txt
   |- SSH
   |- OpenOffice
   |- TFTPD
   |- Editors
   |- GoogleTools
   |- SIMULATOR
   \- UML
```

Figure 4.34: Directory structure of ESAOA workstation distribution.

4.6.8.4 Sample installation of ESAOA workspaces

The ESAOA workspaces can be configured in multiple ways. Figure 4.35 shows how ESAOA workspaces were prepared for Experiment 2. This structure was based on the way in which developers worked on Projects P1-1 and P1-2.

University of Cape Town

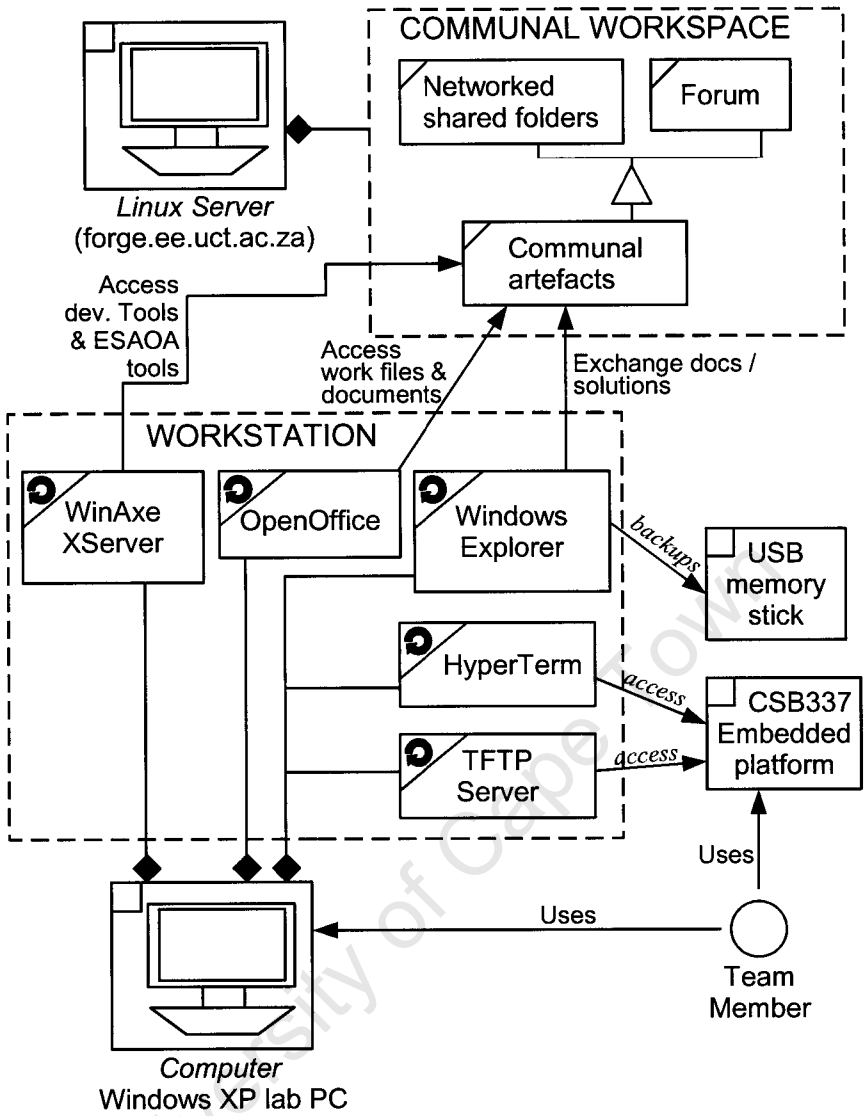


Figure 4.35: Sample ESAOA workspace installation.

4.7 Towards Experiment 2

The ESAOA KMS version 1, as described in Section 4.6, was implemented across 13 projects. These 13 projects formed part of Experiment 2. Data resulting from the activities of these development teams were captured and analysed. In the next chapter, the findings from Experiment 2, in which the first iteration of the ESAOA KMS was applied, are presented.

University of Cape Town

University of Cape Town

Chapter 5:

The Second Experiment

This chapter focuses on the findings of the second experiment. A brief overview of the thirteen projects that comprise this experiment is given in Section 5.1. This is followed, in Section 5.2, by a presentation of the detailed results obtained by means of the data analysis (described in Chapter 3). Section 5.3 summarises the detailed results presented in Section 5.2. The results of the code and design reviews and the review panel evaluations are given in Section 5.4. Section 5.5 performs comparisons between the findings, using correlations to investigate how the different variables identified in the results may influence one another. Section 5.6 reports on evaluations performed by Experiment 2 participants. Section 5.7 presents general conclusions concerning the first version of the ESAOA KMS. The final section, Section 5.8, identifies changes planned for the second version of the ESAOA KMS.

University of Cape Town

5.1 Overview of the second experiment

The second experiment comprised thirteen projects in which teams of novice ES engineers conceptualised, designed and developed ES prototypes over an eight-month period. These prototypes are simulations of what may be expected of ES engineers in commercial contexts. The selection criteria for these projects are detailed in Section 3.6.1; as mentioned in that section, the projects of Experiment 13 all used the same embedded platform (namely, the CSB337 embedded computer manufactured by Cogent Computers [2005]) and developers added additional, specialised hardware components as needed. The projects of Experiment 2 are described in Table 5.1.

Table 5.1: List of Experiment 2 Projects.

Project No.	Project Title	Project Description	Comments
P2-1	Location-aware Tourist Information System (TIS)	The TIS is a GPS-based device that allows a tourist to determine his/her current position and provides information about the current area.	Similar to the idea of GPS-type navigation systems in wide use.
P2-2	GPS Bus Tracker (GBT)	A GPS-based device that gives the closest bus station to the current position and the time of the next bus to a given destination. Includes function to show optimal path to a destination.	Idea to extend GPS navigation system for use with public transportation.
P2-3	Vibynet	A 'Vibynet piece' is a compact electronic device carried or worn by people. It has an identity code and allows storage of other identity codes used to recognise other Vibynets pieces in the vicinity. Features: shows recognised identity codes, direction, distance, call.	Product to be used by a wide range of private individuals, similar to a 'beeper' product.
P2-4	MyIP Phone Station (MPS)	A stand-alone (non-PC) VoIP (both voice and video) answering machine system that connects directly to Ethernet. Answers any VoIP calls not answered manually within a certain time limit.	Product similar to a commercial telephone answering machine.
P2-5	Home Automation System (HAS)	Comprises a central unit that connects to other electronic systems and household products around the house, incorporates input devices (keypads and sensors) and output devices (LCD screens and linear actuator).	Builds on the idea of 'intelligent home' products and service (see http://www.intelligenthome.com.au/)
P2-6	Automation Headlights Dimmer (AHD)	A motor car system that senses a bright on-coming light and automatically dims its own headlights until the light source has passed.	Similar to DIY upgrade kits advertised in *Popular Mechanics* magazines.
P2-7	Field Sensor for Maglev Trains (FSMT)	Based on the idea of using electromagnetic fields to levitate and drive high speed trains. Focus is a device for sensing electromagnetic fields and digitally notifying other systems.	A type of off-the-shelf component that a development company may buy.
P2-8	Cordless STereo (CST)	A stereo that has no speakers, but outputs digital audio for Blue Tooth headphones.	Similar to existing Blue Tooth products.
P2-9	Central Alarm Clock (CAC)	Main controller based in a common room in the home. Has buttons that correspond to rooms in the house. Remote control to help parents wake up their kids without moving a centimetre. Communicates over AC power lines.	Similar in concept to burglar alarm systems, which are commonplace products.
P2-10	Voice Activation System (VAS)	Voice recognition system to activate/deactivate/control electrical appliances (e.g. TV's, lights).	Similar to commercially available automatic dimmer switches.
P2-11	Supermarket Query Device (SQD)	Device communicates with supermarket server, which holds database of items on special. Lets user browse through current specials, locate items of interest.	Comparable to electronic information directory found in shopping centres.
P2-12	Personal Protection Device (PPD)	Has one button that can be used to set off an alarm to notify campus/site security people that a certain individual, at a certain location, is in danger so that the closest security officers can be sent to the rescue. Device is small enough to attach to ID swipe card or to wear around neck.	A product that could replace a standard panic alarm system.
P2-13	Vehicle Usage Tracker (VUT)	Affordable vehicle usage monitoring kit (for DIY installation). Displays real-time car statistics, such as average speed and fuel consumption. Colour display and audio warning system.	Could be sold as a DIY kit, like others currently advertised in *Popular Mechanics* magazines.

University of Newcastle

Data were collected from the projects of Experiment 2 using the strategy described in Section 3.8. Once the experiment was completed, the data sources were examined by following the same procedure as the one used in Experiment 1, in which the data sources were all printed, sorted chronologically and then annotated (see Section 4.3). The data systematising procedure was then applied (as described in Section 3.9.2), which involved producing a knowledge register for each project (Appendix B shows the complete knowledge register of the first project, Project P2-1).

5.2 Results of the second experiment

The ES laboratory was prepared before the thirteen project teams began their work. The ESAOA workstation version 1 tools (listed in Section 4.6.7.1) were installed on the laboratory computers. The ESAOA environment and baseline ESAOA workspaces were installed on a central server. The SnapGear uCLinux operating system [SnapGear, 2007] was used on the embedded platforms, with Micromonitor as a "boot platform" [Sutter, 2002, pg. 62].

Before starting on the projects, each team member completed a three-hour training session[1] on using the ESAOA KMS version 1. The set of participants were divided into two sessions, with 18 individuals attending the first session and 21 attending the second session. During training, the participants worked in an individual capacity, not as a team. Each individual was trained in the procedures of the DS, PE, and IE roles (Section 4.6.4 defines these roles). The training sessions introduced the participants to basic aspects of KM and to terminology related to the ESAOA KMS (specifically the *ESAOA knowledge ontology*) and made use of ESAOA models to assist in explaining the KMS. In addition, the training took the participants through a simple tutorial [Winberg, 2005b] on the use of the ESAOA environment, ESAOA workspaces, and ESAOA workstations (which included the procedures relevant to the DS, PE and IE roles). During training, participants were exposed to refined versions of some of the artefacts produced by the Experiment 1 teams (see Section 4.6.5.5); this was done mostly as case study discussions during the training. In each training session, the participants developed a simple C program that ran on the *PC box* virtual platform (a simplified ES simulated platform that executed on the central server – see Section 4.6.5.5).

[1] This training period and the two weeks learning the ESAOA KMS were performed before the start of the eight-month period; during this time teams also worked on product requirements.

University of Cape Town

After training, the participants were required to work more closely together in their teams. The teams each spent a further two weeks familiarising themselves with the ESAOA KMS, during which time they each worked on a mini-project in which they used aspects of the ESAOA communal workspace to implement an installation of uCLinux [SnapGear, 2007] on the CSB337 platform, and to develop a 'Hello World' embedded software application [Winberg, 2005b]. The team members were encouraged to consult with the research (filling the CKO role) and knowledge steward to improve their understanding of the ESAOA KMS while working on this mini-project. During this time, the members also decided on their primary ESAOA role (i.e., DS, PE or IE) for their project (this is termed the 'primary role' because, as mentioned in Chapter 3, an individual can change roles as needed during a project).

Once the initial two week training period was over, the teams started to work on their main projects, and the process of capturing data related to ESAOA KM began. During the projects, the teams supplemented and adapted tools and artefacts of the ESAOA workspaces. These modifications tended to be changes to the digital artefacts within ESAOA workspaces (such as changes to scripts, code files and directory structures), as well as modifications to the roles and procedures of ESAOA KMS version 1.

The thirteen subsections that follow respectively provide results for each one of the thirteen projects of Experiment 2. Each subsection begins with an overview of the general progression of development work done on the project. For each project, a figure showing a *boundary artefact,* which was extracted from the data related to the project, is provided as a means to assist the project descriptions. Four *knowledge occurrence graphs* (explained in Section 3.9.7.2) are provided for each project; these respectively show (a) data knowledge occurrences over event chains, (b) process knowledge occurrences over event chains, (c) innovation knowledge occurrences over event chains, and (d) total knowledge occurrences over event chains. Each graph tracks both the accumulated productive knowledge occurrences (the black trend line) and the accumulated non-productive knowledge occurrences (the red dotted trend line). *Knowledge occurrence tables* (see Section 4.5.3.1) are also provided for each project. Each subsection ends with a short synopsis that reflects on the probable relations between development trends, together with a knowledge occurrence table that summarises the statistics. The process knowledge shown in the knowledge occurrence tables is divided into subcategories of role knowledge, logistics knowledge, and engineering method knowledge – this was done (as

University of Cape Town

explained in Section 5.7.2) because these subcategories were found to be largely distinct from one another and were therefore separated to provide better views into process knowledge production.

The numbering of figures, graphs and tables is based on subsection numbers: the caption numbers of the tables in Section 5.2.1 start with 'Table 5.2.1'; the caption numbers of the tables in Section 5.2.2 start with 'Table 5.2.2', and so on. The same method is used for the captures of figures and graphs.

5.2.1 P2-1 Location-aware Tourist Information System (TIS)

The first project (Project P2-1) was the creation of a 'Tourist Information System' (or TIS for short). The objective of this product is to allow a tourist (the user of the system) to determine his/her current position on a detailed street map. In addition, it provides information about the surrounding area that is relevant to tourists (e.g., shops, hotels and historic sites). The product was intended to be mounted in a rented car, and to broadcast information and advertising on the car's radio. The device is programmed to provide information based on the current longitude and latitude position obtained from a GPS unit. Information and advertising stored on the unit are updated by connecting to a central server using the general packet radio service (GPRS). Figure 5.2.1 illustrates the TIS; Figure 5.2.1 (a) provides a system block diagram, and Figure 5.2.1 (b) shows an enclosure drawing. Both images shown in Figure 5.2.1 were included in the team ESAOA workspace for the project and were classified by the team as boundary artefacts. The first diagram was assigned the additional functionality classification 'Documentation/concept model', whereas the second was assigned 'Documentation/concept sketch'. These artefacts were used repeatedly during meetings as boundary artefacts.

University of Cape Town

The TIS prototype made use of the following components: a Motorola on-core M12M GPS receiver module (GPS starter kit and RS232 evaluation board)[2], an 8-column 7-Segment LED display, LXT971A on-board Ethernet controller, and the AT91RM9200 on-chip timers.

[2] http://www.avnet.co.za/Semiconductors/Motorola/FS_Oncore/MotGPS-FS_Oncore.htm

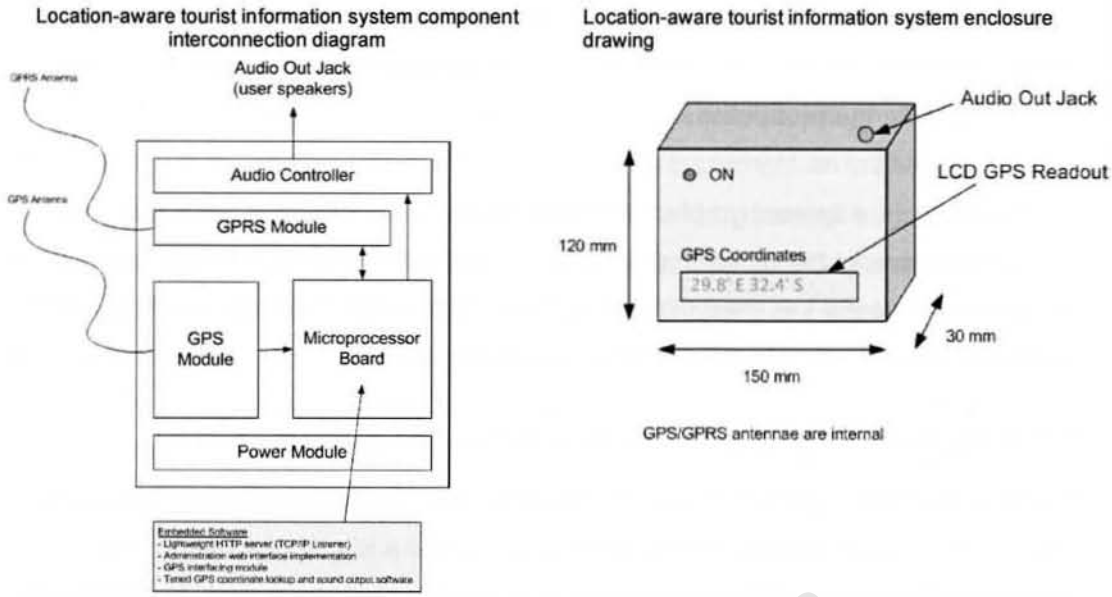


Figure 5.2.1 (a): Component interconnection diagram of Tourist Information System (Project P2-1); (b): Enclosure drawing.

Data knowledge (see Graph 5.2.1 (a)) was produced during the first half of the project in order to decide on components to use and to build process knowledge for the components that were chosen. After event chain 37, no further data knowledge was required by the team.

The data knowledge graph clearly indicates that, as the project developed, a portion of data became non-productive. This occurred between event chains 12 to 16, in which the group spent a significant portion of time reading up on GPS module options, with the intent of finding one that was simple to control via an RS232 serial connection, as well as exploring GPRS module options during this stage. This body of knowledge was classified as non-productive because only one of the GPS modules was ultimately used, and because the group decided not to incorporate a GPRS modem for uploading advertising and location information, but to use a direct TCP/IP Ethernet for this task instead. In terms of productive data, event chain 28 required a considerable effort in producing productive data knowledge during which time the group researched the National Marine Electronics Association (NMEA) ASCII communications protocol used to communicate with the GPS receiver module.

Process knowledge production, shown by Graph 5.2.1 (b), tended to level out when developers produced significant data and innovation knowledge (e.g., between event chains 9 – 19 and 27 – 41). The developers first had to understand how to develop

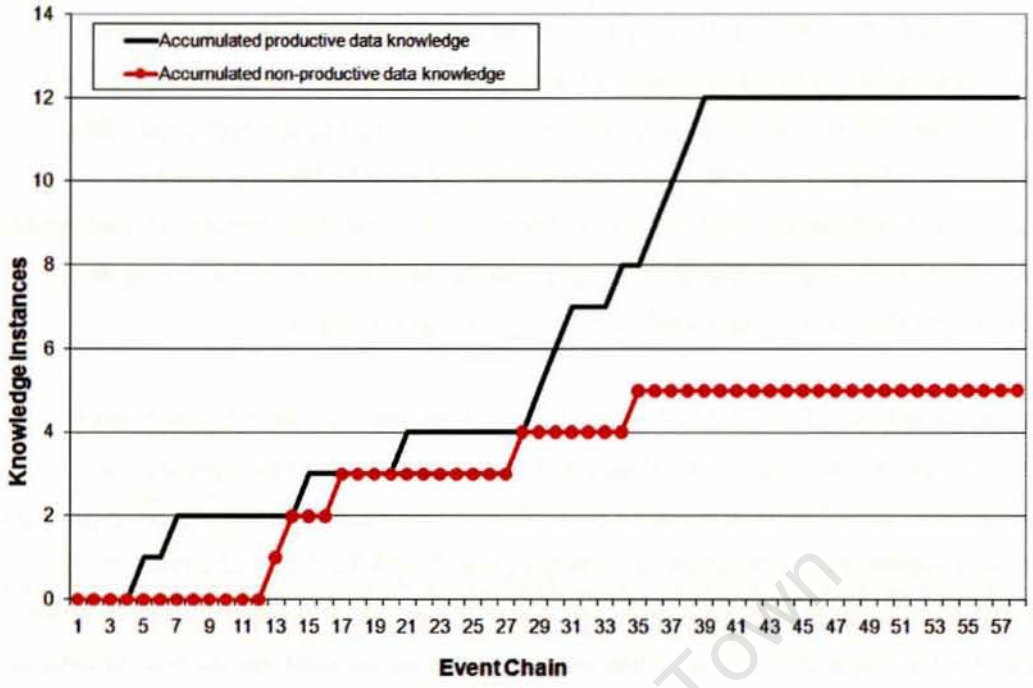
University of Cape Town

embedded software for the uCLinux operating system (event chain 3 – 10), before they could create methods of using TCP/IP sockets and the Ethernet controller (event chains 18 – 23). Only towards the end of the project (event chains 41 – 49) did the team succeed in accessing the GPS module. Most of the non-productive process knowledge occurred during event chains 21 – 27 when the team experienced problems in ISO-mounting the uCLinux RAM disk (which was necessary to add their executable application program to the boot image) and in using tftp (in order to test their application without placing it in the RAM disk).

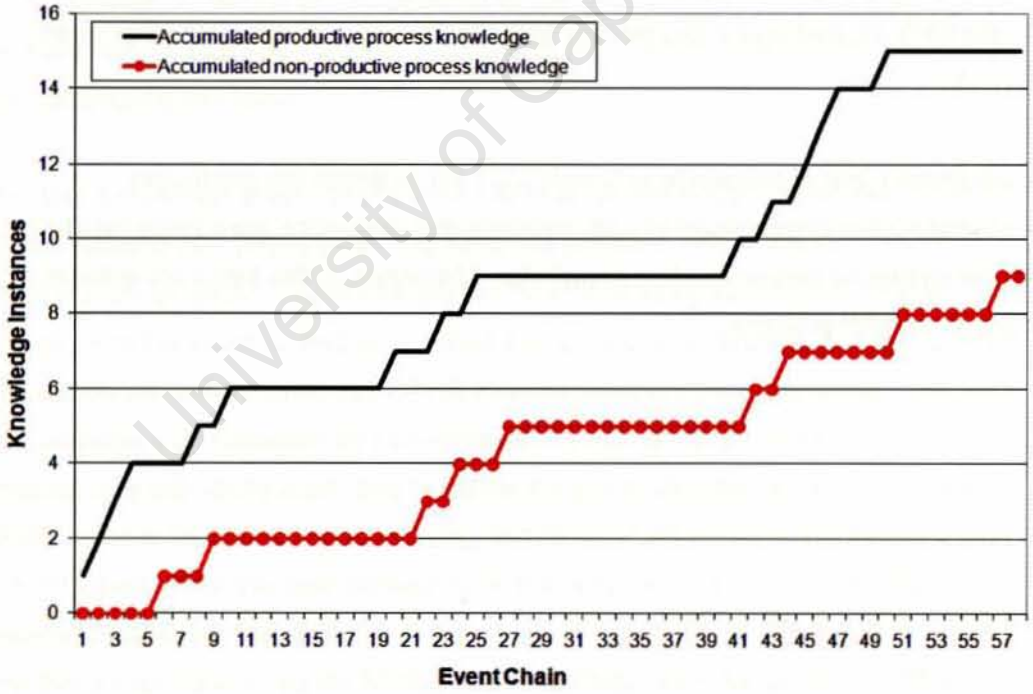
Innovation started at event chain 10 and, apart from the first few non-productive occurrences of innovation, almost all other event chains generated productive innovation knowledge (with the sole exception of event 52). In comparison to process and data knowledge, innovation knowledge (see Graph 5.2.1 (c)) occurred more consistently, as seen in the frequent upward steps in the innovation graph. At event chain 19, the team did have an initial version of the embedded application running on the embedded platform. However, this version was using a GPS stub module because the important process knowledge related to methods used to communicate with the GPS module was acquired only towards the end of the project (after event chain 41).

A divergence trend is noticeable in Graph 5.2.1 (d), in which the productive knowledge for the combination of data, process and innovation grew more rapidly than the combined non-productive knowledge. Variations of this trend are evident across all thirteen projects.

University of Cape Town

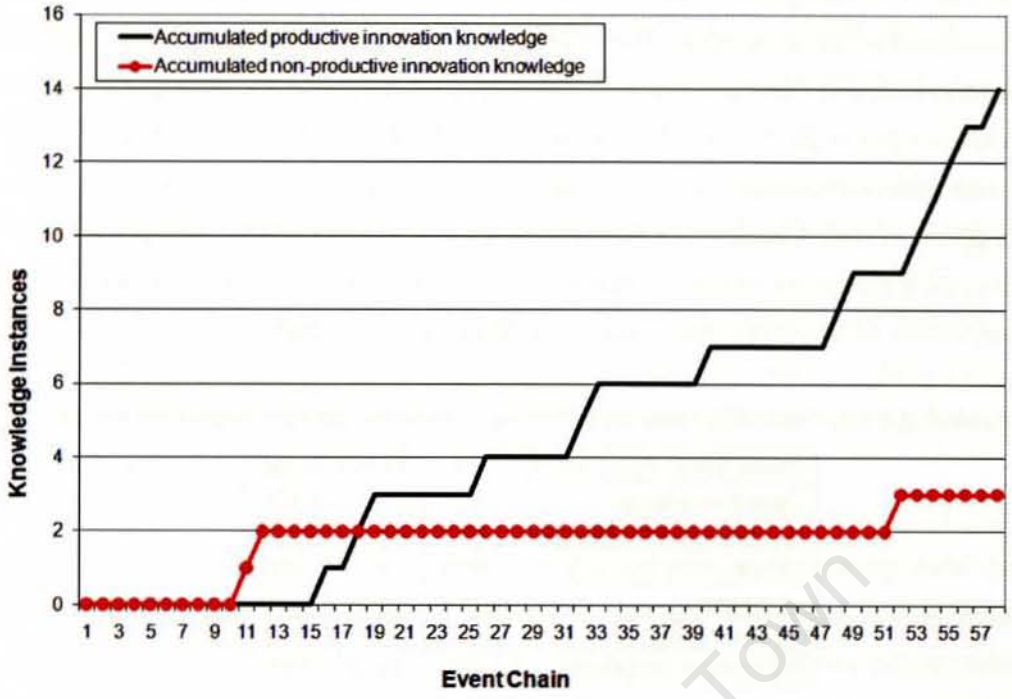


Graph 5.2.1 (a): Data knowledge in P2-1.

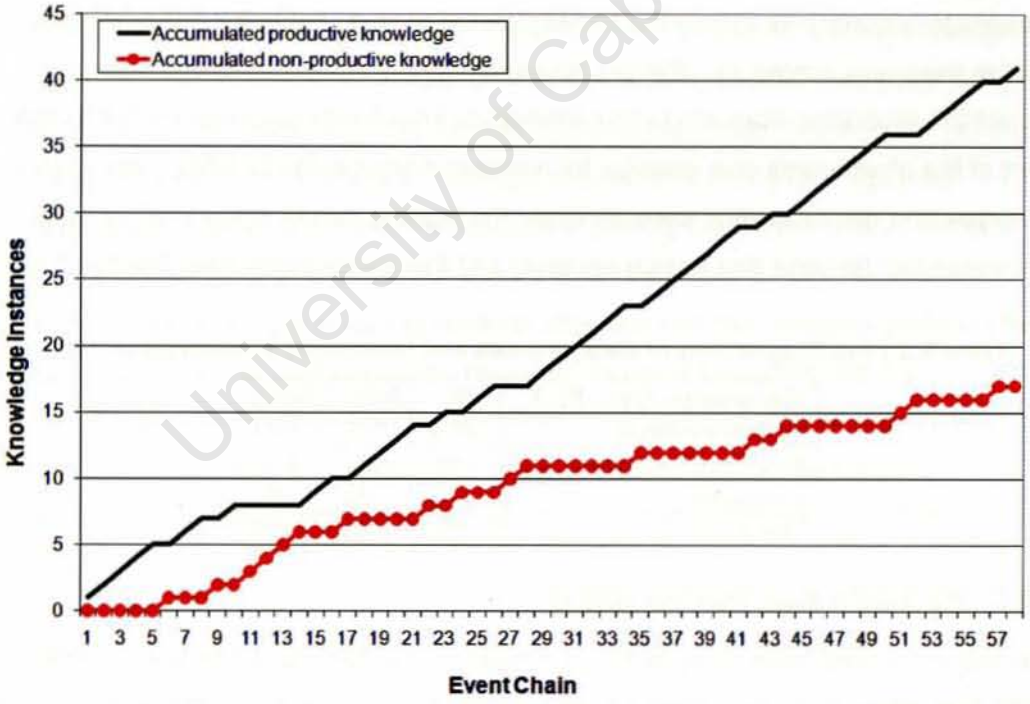


Graph 5.2.1 (b): Process knowledge in P2-1.

University of Cape Town



Graph 5.2.1 (c): Innovation knowledge in P2-1.



Graph 5.2.1 (d): Productive and non-productive knowledge in P2-1.

Knowledge occurrences were summarised, and are tabulated in Tables 5.2.1 (a) and (b). The first table (Table 5.2.1 (a)) shows the percentages of productive and non-productive knowledge within each of the three knowledge categories. Table 5.2.1 (a)

University of Cape Town

shows that in all categories of knowledge, there was more productive knowledge than non-productive knowledge. The ratio of productive knowledge to non-productive knowledge was even for both logistics knowledge (a ratio of 1:1) and engineering methods (a ratio of 29:25 ≈ 1.1:1). The most out-of-balance ratio for productive to non-productive knowledge was role knowledge (a ratio of 25:4, equivalent to 6.25:1). The majority of data knowledge occurrences were productive (71%), which was found to be a similar trend across all projects. Process knowledge was the most common form of non-productive knowledge found in this project.

Table 5.2.1 (a): Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-1	PK	NPK	Total
Data Knowledge	71%	29%	100%
All Process Knowledge	63%	38%	100%
Role	25%	4%	29%
Logistics	8%	8%	17%
Engineering methods	29%	25%	54%
Innovation Knowledge	82%	18%	100%

Table 5.2.1 (b) summarises the percentages of productive and non-productive knowledge occurrences across the knowledge categories. Table 5.2.1 (b) indicates a relative evenness across the different knowledge types in the productive category. Project 2-1 produced more productive innovation knowledge occurrences (24%) than most of the other teams (the average for innovation knowledge is 19%). This team's prototype and demonstration were awarded the highest overall score (100%) by both the researcher (in code and design reviews) and the review panel (see Section 5.4).

Table 5.2.1 (b): Proportions of data, process and innovation knowledge in P2-1.

Knowledge Type: P2-1	PK	NPK	Total
Data Knowledge	21%	9%	29%
Process Knowledge	26%	16%	41%
Innovation	24%	5%	29%
TOTALS	71%	29%	100%

5.2.2 P2-2 GPS Bus Tracker (GBT)

The second project team (Project P2-2) worked on the 'GPS Bus Tracker' project, which was intended to help users find the closest bus stop and the optimal route from one bus stop to another (Figure 5.2.2 illustrates the concept). The team developed a portable device that was intended to interface with a cellular telephone. The PC/phone software was supposed to retrieve bus data (time tables, routes, locations, etc) from a server. Bus data would then be transferred to the Bus Tracker via a Bluetooth connection. The user would call up on the cellular telephone (connected to

University of Cape Town

the Bus Tracker) to choose a destination and view the best bus stop. The components used included: the M12M Motorolla GPS receiver module (included in the GPS RS232 starter kit), the Dallas DS1307 real time clock, pushbuttons, LEDs, and the LXT971A Ethernet controller.

The team had to do a considerable amount of research on the components they had chosen, particularly the GPS modules, which is evident in the occurrences of productive data knowledge produced by the team (see Graph 5.2.2 (a)). A likely cause for the large amount of non-productive data knowledge was the team's inability to find sample code for using this particular GPS module with the uCLinux operating system (which all the project teams were using).

In this project, the first occurrences of non-productive process knowledge (see Graph 5.2.2 (b)) were caused by the team's difficulties in defining their roles and performing within their (disputed) roles. A later difficulty was the use of a GPS module for which they developed complex implementation procedures in order to be able to use the device in the product. Knowledge occurrences began as the group installed ESAOA tools on their own PCs. The team was working on two problems: getting interrupts to function and getting the GPS module to work. Once the team obtained good sample code (around event chain 28 – 49), they were able to communicate with the GPS module. The productive process knowledge occurrences from event chain 69 onwards are indicative of the team's success in getting these interrupts to function.

At event 81, the team attempted to develop alternate and more elegant methods for using the components and interrupts. However, much of this effort was not successful, and it would have meant significant changes to the work already done by the innovation engineer. Consequently, most of the non-productive process knowledge occurrences after event 80 can be attributed to this unused effort.

University of Cape Town

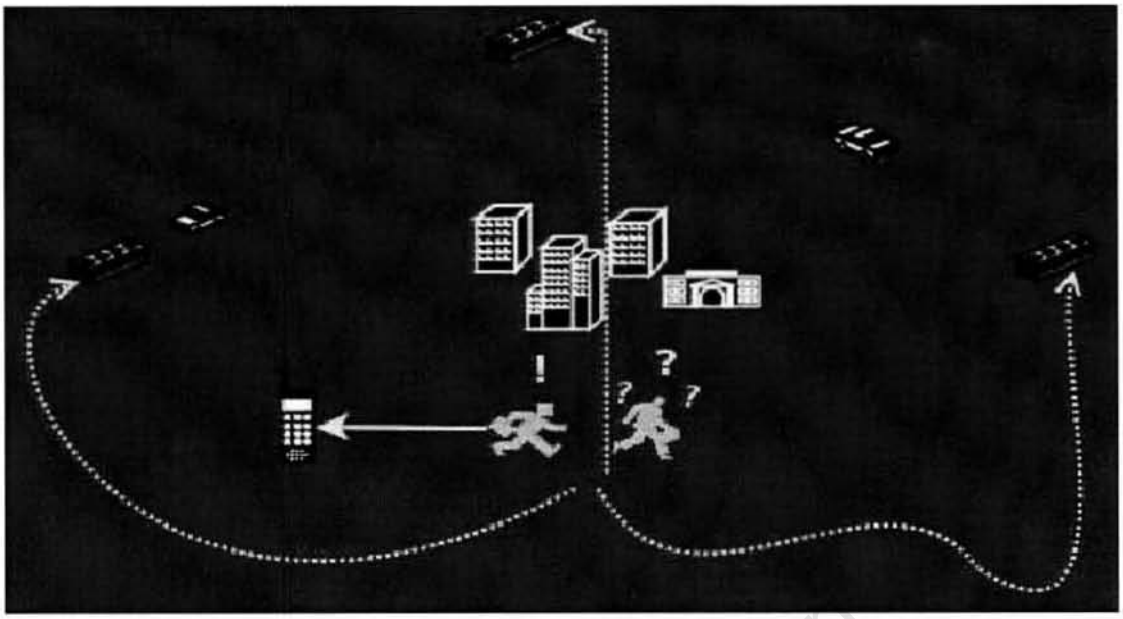
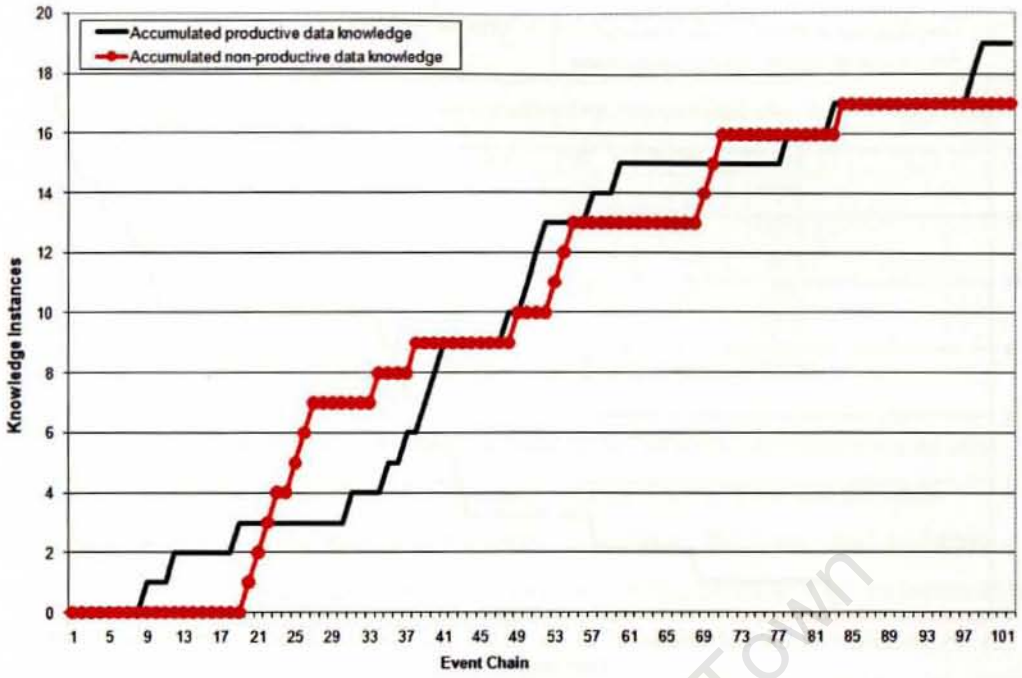


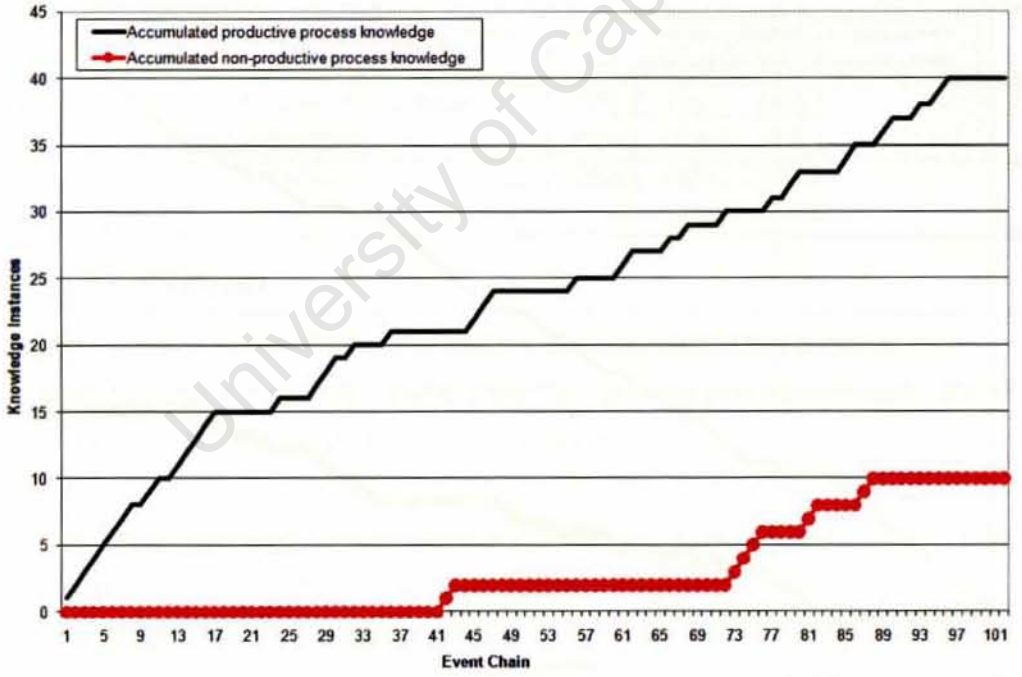
Figure 5.2.2: Concept scenario for the GPS Bus Tracker (Project P2-2).

Innovation knowledge occurrences began (see Graph 5.2.2 (c)) at event chain 18, when the team developed the first version of their software, which could be executed on the embedded hardware. At event chain 33, the group successfully installed the menu. At event chain 39, there was further development in computing the bus routes based on GPS positioning. Subsequent indications of innovation are indicative of further integration and refinement. A divergence trend (see Graph 5.2.2 (d)) is noted in P2-2's increase in productive over non-productive knowledge occurrences.

University of Cape Town

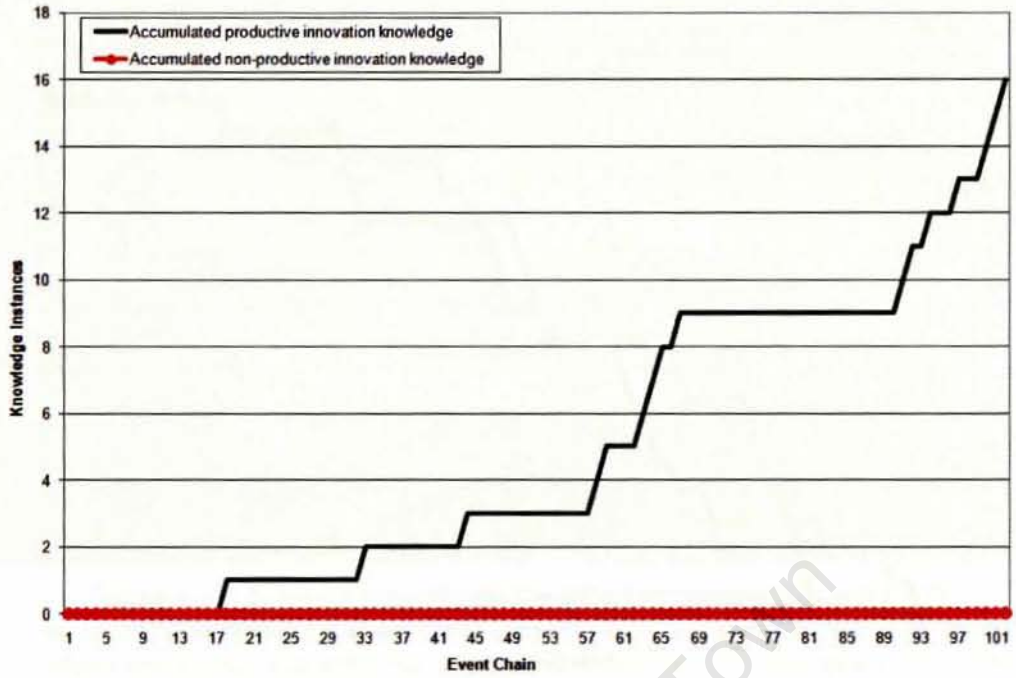


Graph 5.2.2 (a): Data knowledge in P2-2.

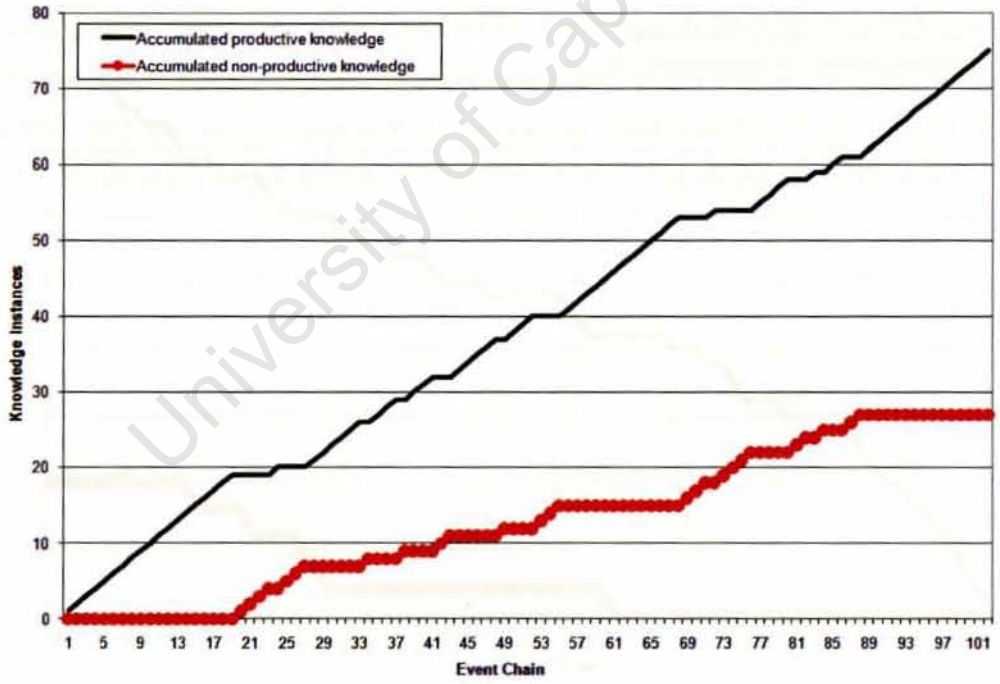


Graph 5.2.2 (b): Process knowledge in P2-2.

University of Cape Town



Graph 5.2.2 (c): Innovation knowledge in P2-2.



Graph 5.2.2 (d): Productive and non-productive knowledge in P2-2.

Summaries of knowledge occurrences in the productive and non-productive categories are tabulated in Tables 5.2.2 (a) and (b). Table 5.2.2 (a) indicates a relatively high occurrence of non-productive data knowledge compared to the other

University of Cape Town

projects. As previously explained, the cause of this was the difficulty of the project, and challenges in finding useful hardware/software interfacing information.

Table 5.2.2 (a): Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-2	PK	NPK	Total
Data Knowledge	53%	47%	100%
All Process Knowledge	80%	20%	100%
Role	16%	4%	20%
Logistics	16%	0%	16%
Engineering methods	48%	16%	64%
Innovation Knowledge	100%	0%	100%

Table 5.2.2 (b) shows that process knowledge occurrences accounted for almost double those of data or innovation. This was slightly higher than the process knowledge occurrences of the other projects. In addition, this team had a slightly lower than average percentage of innovation knowledge occurrences. In terms of the assessment of artefacts by the researcher, and the evaluation of the prototype demonstrated to the review panel, this team achieved the lowest overall score (65%).

Table 5.2.2 (b): Proportions of data, process and innovation knowledge in P2-2.

Knowledge Type: P2-2	PK	NPK	Total
Data Knowledge	19%	17%	35%
Process Knowledge	39%	10%	49%
Innovation	16%	0%	16%
TOTALS	74%	26%	100%

University of Cape Town

5.2.3 P2-3 Vibynet

The third project (Project P2-3) involved the development of the 'Vibynet' product (illustrated in Figure 5.2.3). A 'Vibynet piece' is a product that a user wears, which alerts the user to other Vibynet users in the vicinity.

The Vibynet project team built productive innovation knowledge quite early in the project, viz. from event chain 4, as seen in Graph 5.2.3 (c). The product was designed around an S/PDIF-based [Finger, 1992] wireless transceiver module (using the nRF24Z1 evaluation kit), to be configured to operate in the 49Mhz unlicensed band. The group chose this audio-streaming module because they originally intended for the Vibynet to have two modes: 'alert' mode (sensing neighbouring devices) and 'hail' mode (operating as a digital two-way radio). There was a delay in obtaining the wireless evaluation kit, which meant that the group only started to read about the specific module quite late in the project (event chain 29, see data knowledge graph,

Graph 5.2.3 (a)). Since the delivery of the module was delayed, the group began by simulating broadcast wireless communications, using an *ad hoc* 1-wire protocol (OWP). Although the team had planned to use a wireless module capable of providing at least 65Kbps, they ultimately resorted to using a slower 'bit-bang' software driver for their OWP, which could only achieve 8Kbps. A significant amount of the non-productive innovation knowledge (see Graph 5.2.3 (c)) occurred when the team attempted to determine how to increase the speed of the bit-bang routine.

When the actual wireless module arrived (around event chain 35), the team had to learn how to configure and use it. During this stage, process knowledge (see Graph 5.2.3 (b)) was produced (much of which was non-productive due to problems with the interface). At around event chain 41, the team began to generate productive innovation knowledge concerning the wireless module (all other innovative knowledge was built using the OWP simulator). Graph 5.2.3 (d) indicates a trend, also noted in the previous projects, in terms of the divergence of productive and non-productive knowledge occurrences.

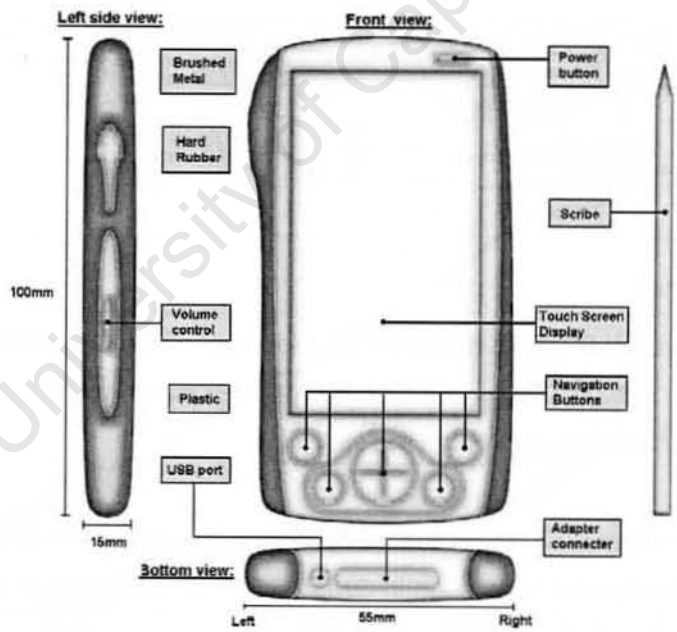
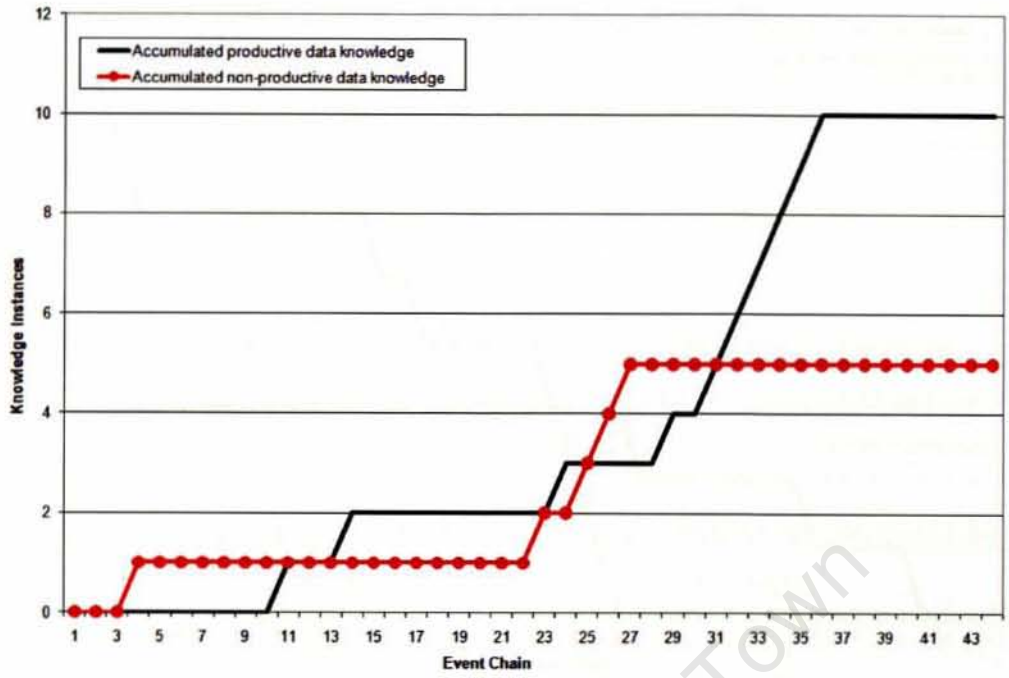
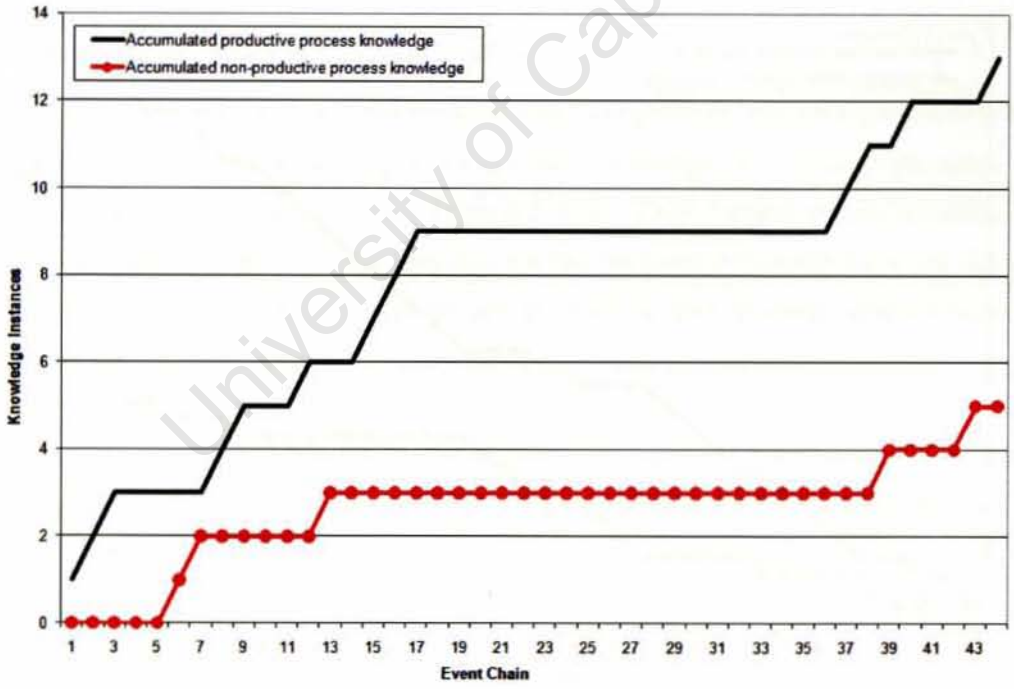


Figure 5.2.3: Concept drawing for Vibynet (Project P2-3).

University of Cape Town

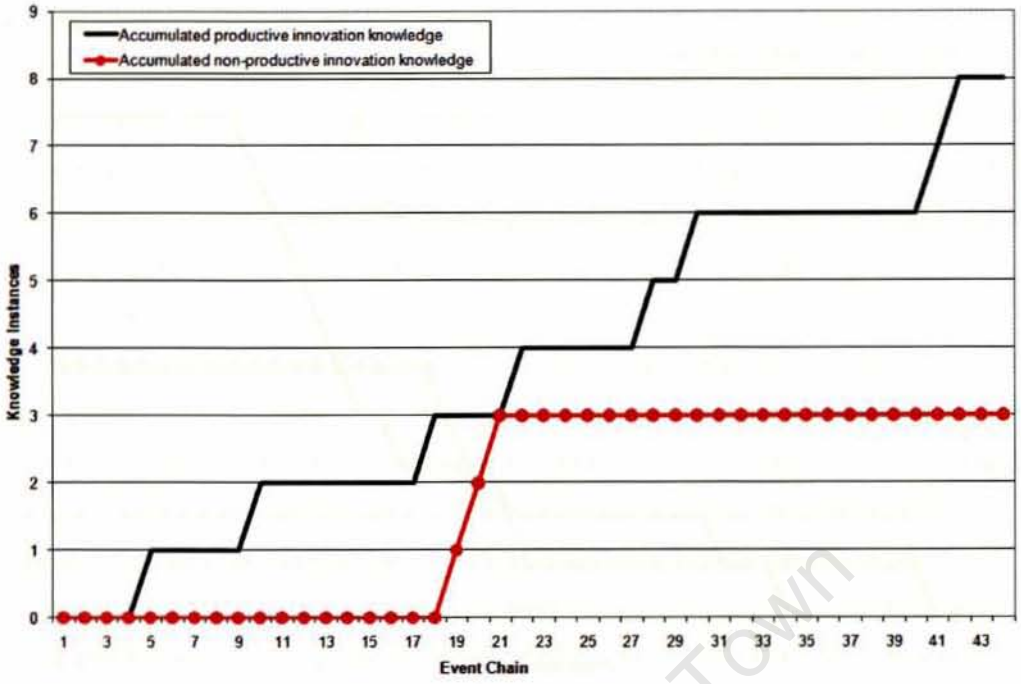


Graph 5.2.3 (a): Data knowledge in P2-3.

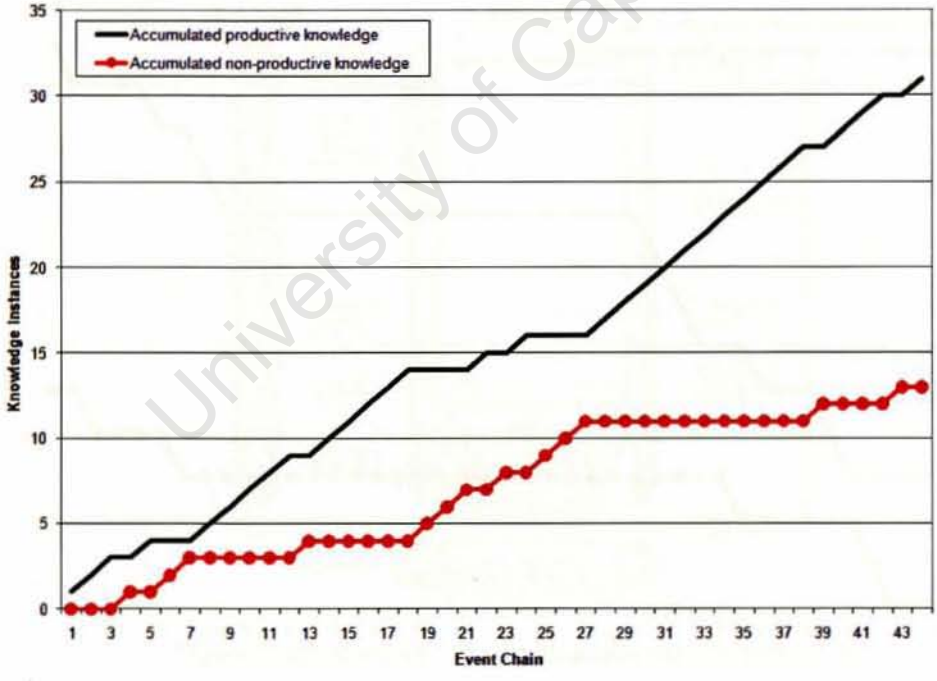


Graph 5.2.3 (b): Process knowledge in P2-3.

University of Cape Town



Graph 5.2.3 (c): Innovation knowledge in P2-3.



Graph 5.2.3 (d): Productive and non-productive knowledge in P2-2.

Table 5.2.3 (a) shows that, despite the difficulties previously described, the Vibynet team were able to produce a relatively high level of productive knowledge.

University of Cape Town

Table 5.2.3 (a): Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-3	PK	NPK	Total
Data Knowledge	67%	33%	100%
All Process Knowledge	72%	28%	100%
Role	17%	0%	17%
Logistics	28%	11%	39%
Engineering methods	28%	17%	44%
Innovation Knowledge	73%	27%	100%

The relatively high proportion of process knowledge (shown in Table 5.2.3 (b)) was linked to the team simulating RF communication. They then had to adapt the simulation code so that it would work when their wireless transceiver kit arrived. This team achieved a combined score of 77.5% for their prototype and demonstration.

Table 5.2.3 (b): Proportions of data, process and innovation knowledge in P2-3.

Knowledge Type: P2-3	PK	NPK	Total
Data Knowledge	23%	11%	34%
Process Knowledge	30%	11%	41%
Innovation	18%	7%	25%
TOTALS	70%	30%	100%

University of Cape Town

5.2.4 P2-4 MyIP Phone Station (MPS)

The MyIP, or Voice over Internet Protocol (VoIP) answering machine project (P2-4), produced the highest percentage of productive knowledge (91% of all knowledge occurrences was productive – see Table 5.2.4 (b)). Only 4 event chains culminated in dead ends (see Graph 5.2.4 (a) and (b)). As can be seen in Graphs 5.2.4 (a), (b) and (c), the MyIP team consistently developed productive data, process, and innovation knowledge.

Three main hardware components were used in this project: the Phillips PCF8591 (a single chip containing multiplexed ADC and DAC), ADC and DAC analogue/digital converters, the LXT971A on-chip Ethernet controller (built into the microcontroller), and the S1D13706QVGA LCD controller. The reason for using the PCF8591 was to record and alternatively replay digital voice messages (alternating between the ADC and DAC function of the PCF8591, without having to learn a second interfacing protocol). The fact that they needed to learn only one protocol may have contributed to the high level of productivity in this team.

The team had partial success with the S1D13706QVGA LCD controller; the team wanted to display only the number of messages and times on the LCD. They were

able to write to the display, but they were not able to set the mode or use colour bitmap graphics – this accounted for event chain 39 being classified as producing non-productive innovation knowledge (see Graph 5.2.4 (c)). There was only one occurrence of non-productive data knowledge at event chain 36 (see Graph 5.2.4 (a)), which was similarly related to difficulties with the display.

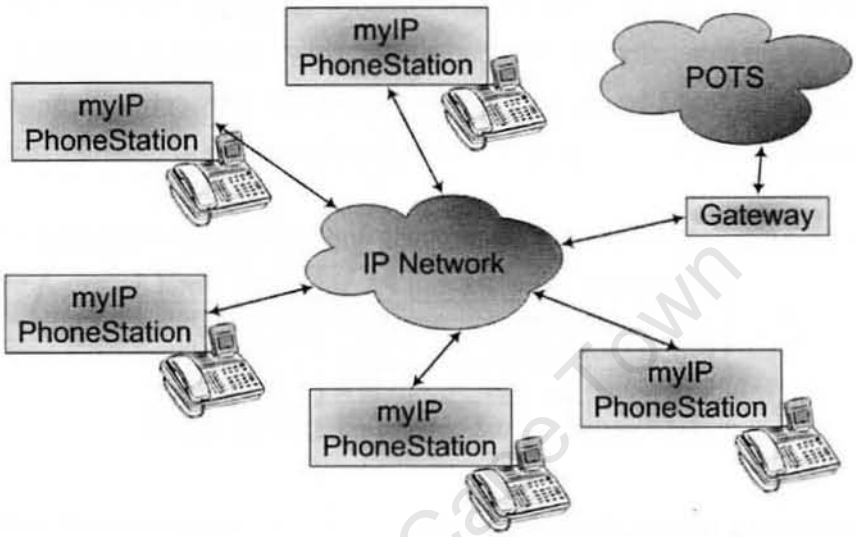
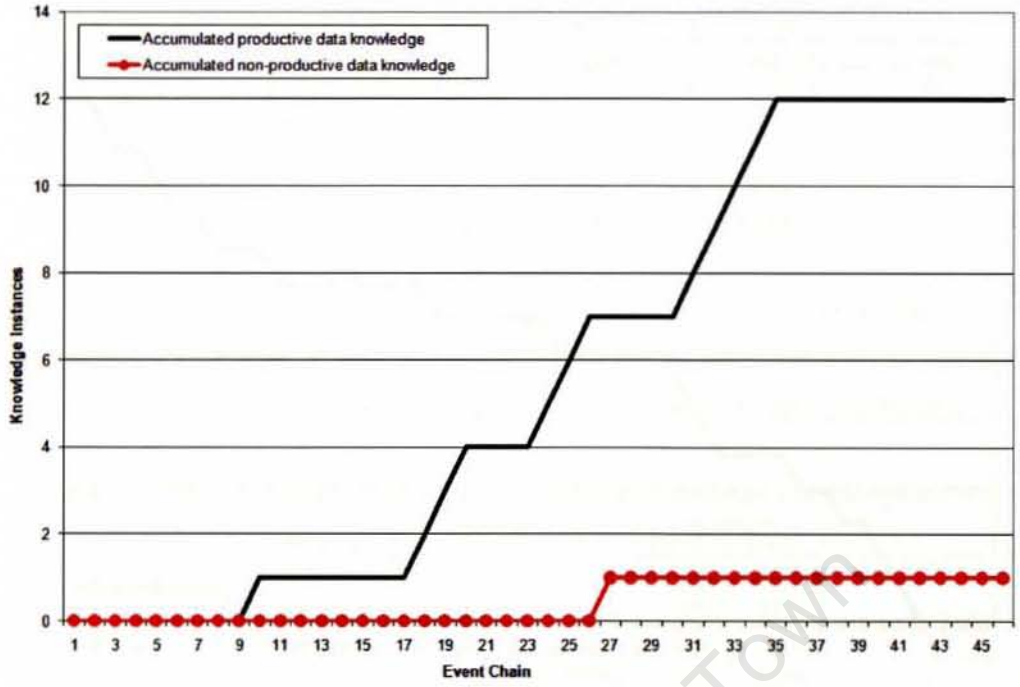


Figure 5.2.4: Topology of myIP Phone Station (Project P2-4).

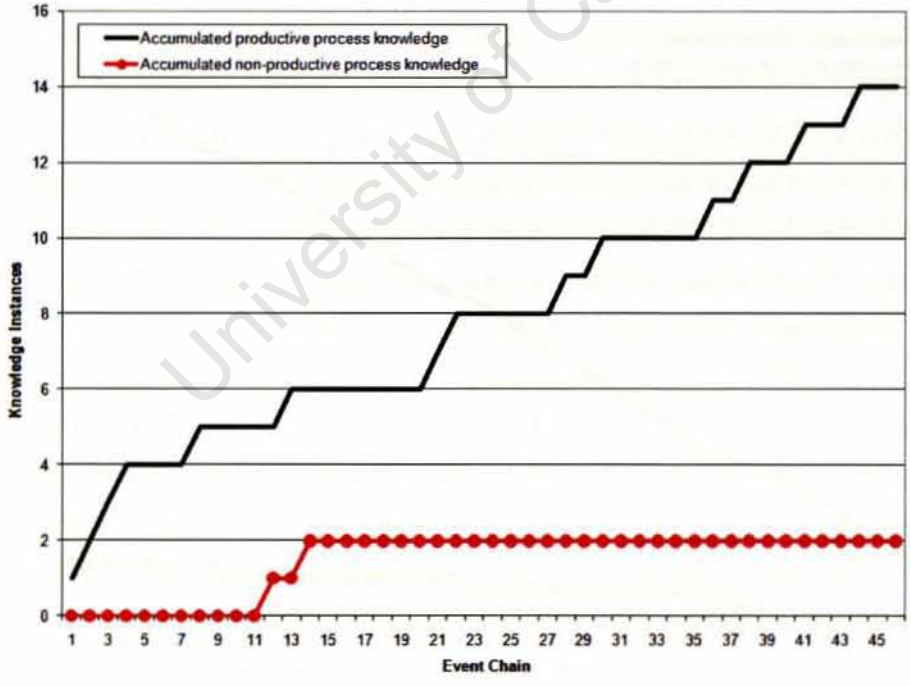
For the MyIP functionality, the team utilised a simple UDP-based communications protocol. They did not implement the official VOIP protocol for the prototype, but instead received raw 8-bit voice data sampled at 4Khz. The main problem they experienced was testing the socket routines, by sending UDP packets between a PC and the CSB337. This was solved relatively quickly after a non-productive knowledge occurrence in event chains 12 – 13 (see Graph 5.2.4 (b)).The MyIP team also used a pre-built web server (namely, the BOA web server [Doolittle & Nelson, 2006]) compatible with Linux. They used a web server instead of implementing a front panel (the users had to connect to the answering machine via a PC).

Graph 5.2.4 (d) and Table 5.2.4 (a) indicate P2-4's high levels of productive knowledge, which was the highest of all the projects.

University of Cape Town

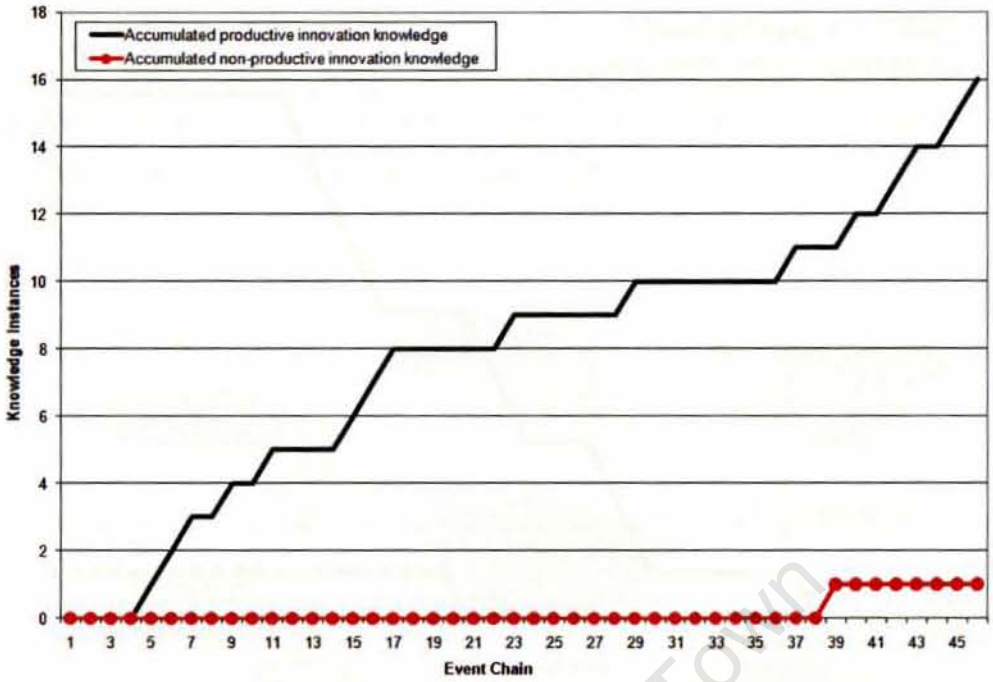


Graph 5.2.4 (a): Data knowledge in P2-4.

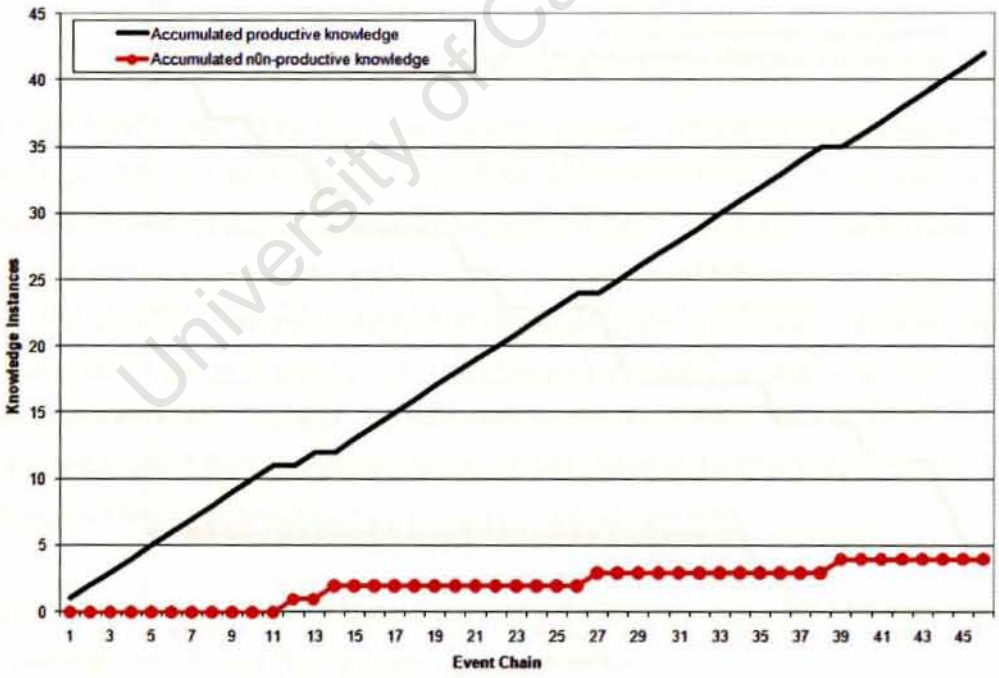


Graph 5.2.4 (b): Process knowledge in P2-4.

University of Cape Town



Graph 5.2.4 (c): Innovation knowledge in P2-4.



Graph 5.2.4 (d): Productive and non-productive knowledge in P2-4.

University of Cape Town

Table 5.2.4 (a): Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-4	PK	NPK	Total
Data Knowledge	92%	8%	100%
All Process Knowledge	88%	13%	100%
Role	25%	0%	25%
Logistics	6%	6%	13%
Engineering methods	56%	6%	63%
Innovation Knowledge	94%	6%	100%

Table 5.2.4 (b) clearly shows the team's high level of innovation knowledge occurrences – indicative of their highly innovative work. Team Vibynet came a close second to P2-1, with a combined artefacts and demonstration score of 99.25%.

Table 5.2.4 (b): Proportions of data, process and innovation knowledge in P2-4.

Knowledge Type: P2-4	PK	NPK	Total
Data Knowledge	26%	2%	28%
Process Knowledge	30%	4%	35%
Innovation	35%	2%	37%
TOTALS	91%	9%	100%

5.2.5 P2-5 Home Automation System (HAS)

The purpose of the HAS was to control electronic consumer products; it did this by controlling relays to switch the main current of appliances on/off (see Figure 5.2.5). HAS instruments comprised an actuator network – with each multiple actuator node controlled by a central computer. Each actuator node was a CSB337 embedded platform, and so was the central computer (the team started off by implementing an initial version of the central computer program, called 'PCControl', which ran on a workstation PC, and they later ported it to the CSB337 platform). The actuator nodes and central control computer were connected using a mains modem that communicates over 200 VAC house mains circuits (they used the LM2893 by National Semiconductors configured to produce 128KHz noise on the mains line for a 64KBps data rate). Each actuator node contained 3 outputs: a relay (capable of switching 2 amps at 220VAC), a DC motor (to open/close curtains, widows, operate fans, etc.), and a LED 7-segment LCD display, controlled by a parallel interface. There was one input: a reed relay, simulated as a Pushbutton in the prototype. The user control interface was complex, requiring the user to program each actuator node. The team originally planned to use the MC3479 step motor controller, but chose not to do so due to time constraints.

University of Cape Town

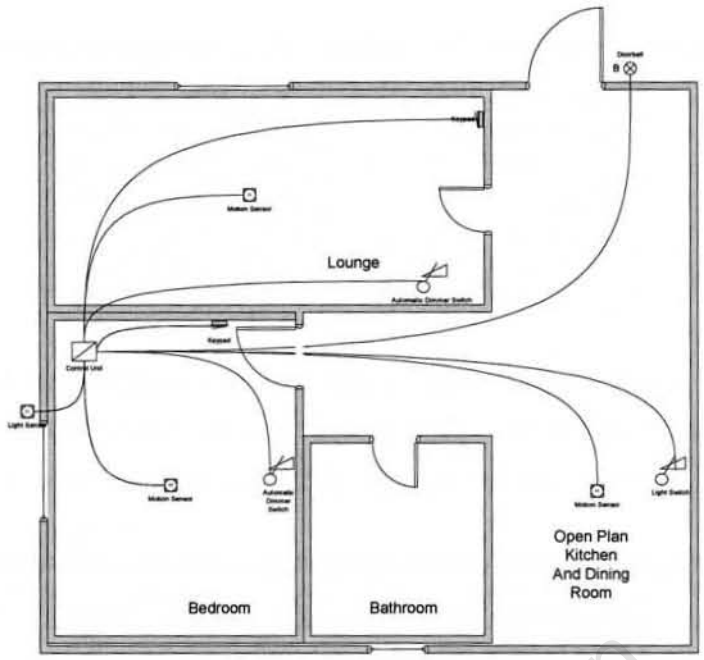


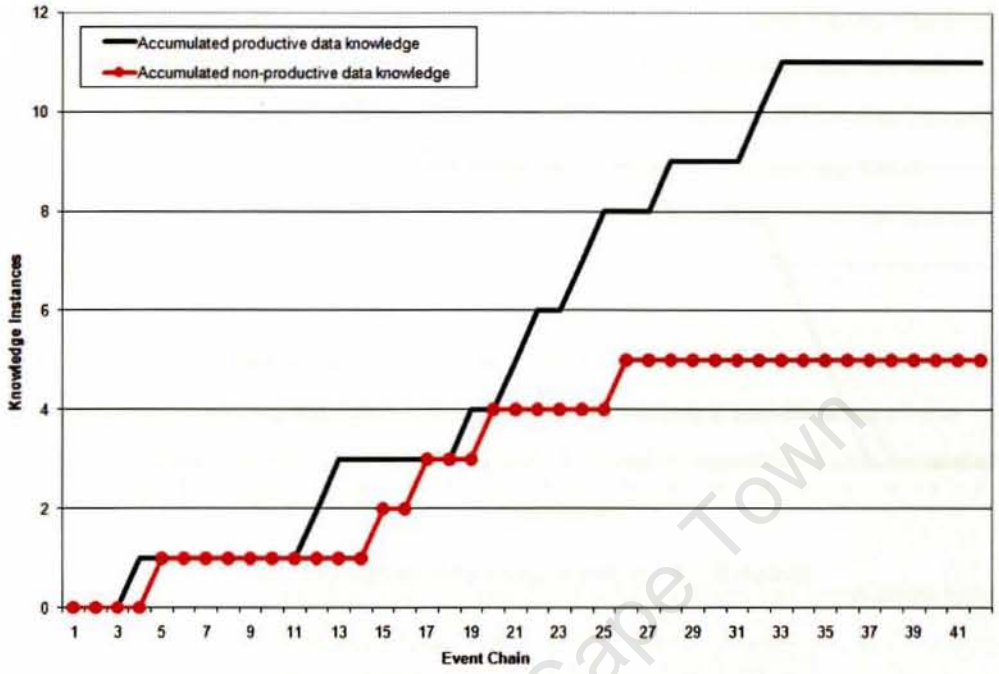
Figure 5.2.5: Installation diagram of Home Automation System (Project P2-5).

Difficulties experienced by the HAS team include: the main modem could only communicate with actuator nodes in the same circuit. However, most homes have multiple circuits for safety reasons. The team had access to only two mains modems (due to costs). However, the mains modem was simple to use, providing a standard AT command interface via an RS232 serial interface. The team experienced some power problems because the modem does not draw from the AC mains, and so a 9V radio battery was supplied. The CSB337 does not provide a 9V power output, so the mains drawing current battery did not last long. The mains modem was not a low power device, and thus the battery had to be replaced frequently. While this did slow the team down, the activities spent finding and replacing batteries did not count as knowledge occurrences.

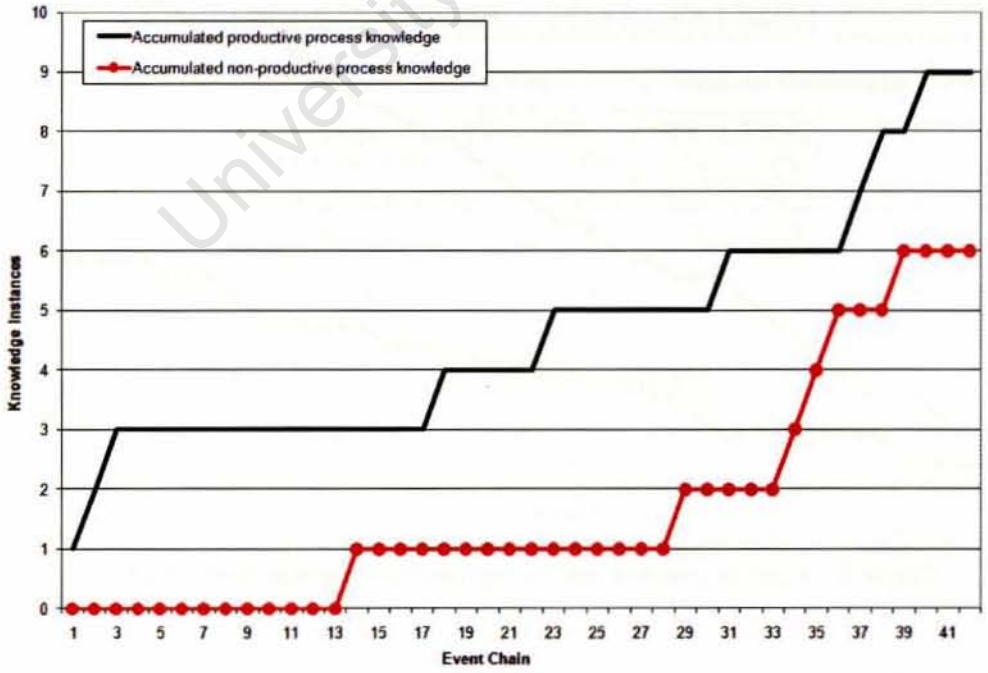
Data knowledge was produced (see Graph 5.2.5 (a)) as the team researched the various components. Finding relevant sample code is indicated in the 'jumps' in event chains 14 – 18 (see Graph 5.2.5 (b)). The team had a number of occurrences of non-productive data knowledge as they attempted to activate and handle interrupts from the pushbuttons. This involved reading about how interrupts are enabled on the AT91RM9200 microcontroller, around event chain 26. The team read sample code related to sockets in event chains 15 and 18. They also had difficulty understanding the RS232UART controller built into the AT91RM9200 microcontroller. The device

University of Cape Town

had two comports, and the team experienced problems configuring the second comport, which was used to connect to the mains modem.

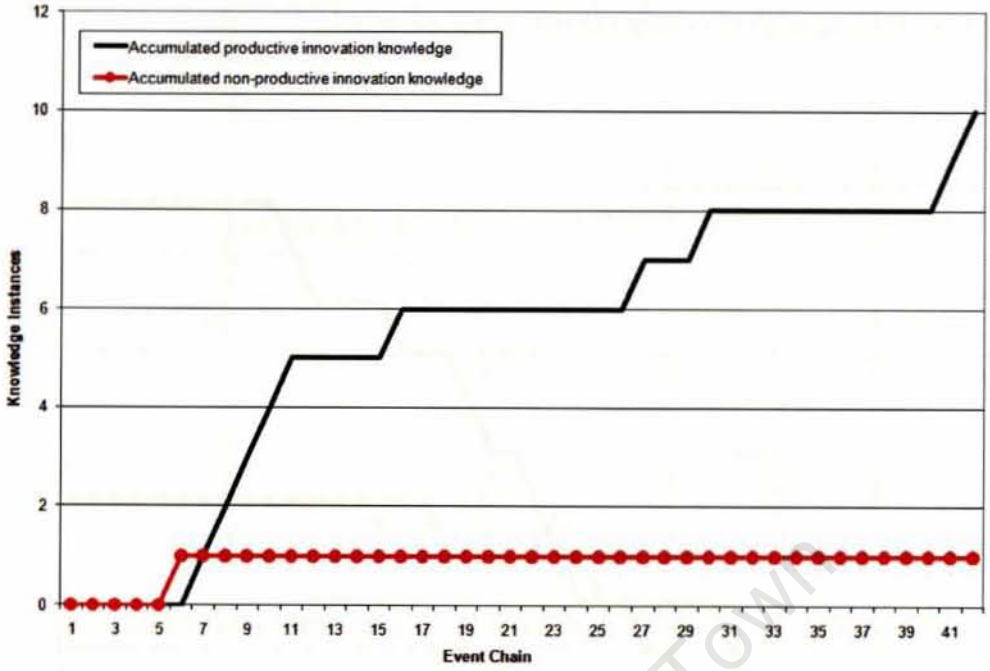


Graph 5.2.5 (a): Data knowledge in P2-5.

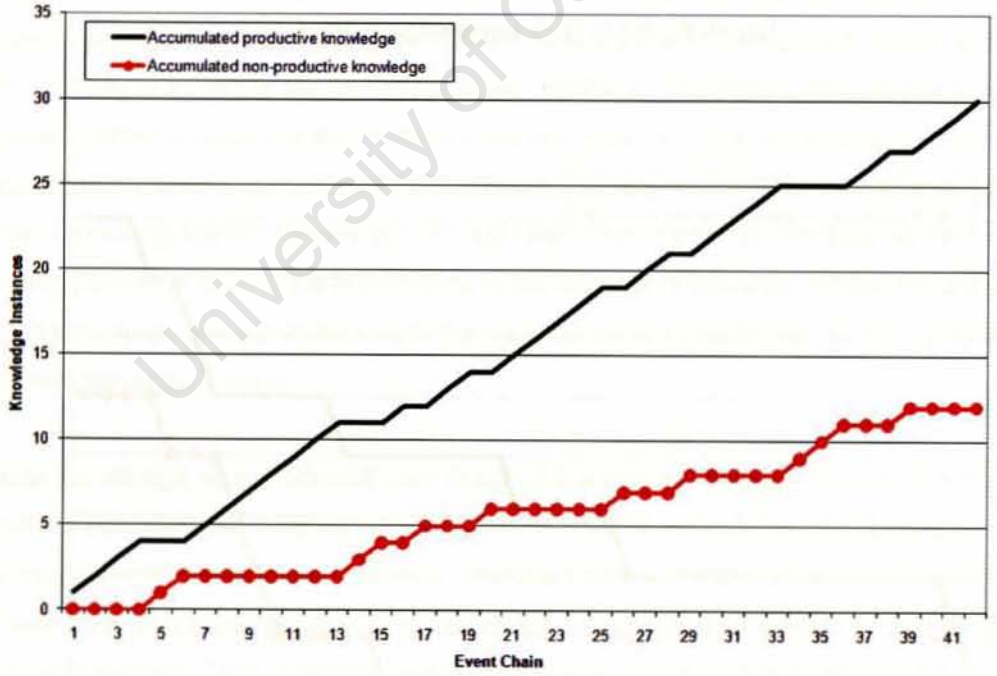


Graph 5.2.5 (b): Process knowledge in P2-5.

University of Cape Town



Graph 5.2.5 (c): Innovation knowledge in P2-5.



Graph 5.2.5 (d): Productive and non-productive knowledge in P2-5.

University of Cape Town

Event chain 10 (see Graph 5.2.5 (c)) indicates the occurrence of innovation knowledge when the team successfully configured the second comport and managed to control the mains modem from the embedded software. Innovation occurred in steps, usually when the team got together and worked productively for long hours over the PC. A TCP/IP communication packet structure was used as an interface to the control computer, whereas the LED and pushbuttons were utilised for the actuator nodes. These were not used as a front panel, but rather to verify actuation events and to test inputs. Graph 5.2.5 (d) shows a similar divergence trend, indicating the team's steady production of more productive than non-productive knowledge.

Table 5.2.5 (a) shows the team's high levels of productive knowledge within each of the knowledge categories. Table 5.2.5 (b) indicates an even distribution of the different knowledge types across the project. This team was awarded a combined score of 93.5% for their prototype and demonstration.

Table 5.2.5 (a): Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-5	PK	NPK	Total
Data Knowledge	69%	31%	100%
All Process Knowledge	60%	40%	100%
Role	20%	0%	20%
Logistics	13%	7%	20%
Engineering methods	27%	33%	60%
Innovation Knowledge	91%	9%	100%

Table 5.2.5 (b): Proportions of data, process and innovation knowledge in P2-5.

Knowledge Type: P2-5	PK	NPK	Total
Data Knowledge	26%	12%	38%
Process Knowledge	21%	14%	36%
Innovation	24%	2%	26%
TOTALS	71%	29%	100%

University of Cape Town

5.2.6 P2-6 Automation Headlights Dimmer (AHD)

This project turned out to be more straightforward than the other projects. The project focused on the design of a DIY headlight dimmer for a standard vehicle. If such a system is to be implemented commercially it would be subject to many safety regulations. In their creation of the prototype, however, the engineers decided to ignore the large volume of literature on safety issues. This was a practically oriented team, which was reluctant to engage in research tasks. The group used the following main components: the Phillips PCF8591, the CSB on-board flash memory (the

28F640J3A flash memory chip), the LEDs and pushbuttons, the Infineon SAK82C900 (CAN controller), a digital pot, and a photo-transistor. A feature these had in common was that many of these have simple interfaces, with the exception of the 28F640J3A flash memory chip and the PCF8591. The team planned to adjust the strength of the light using a dimmer device instead of turning the headlights on/off.

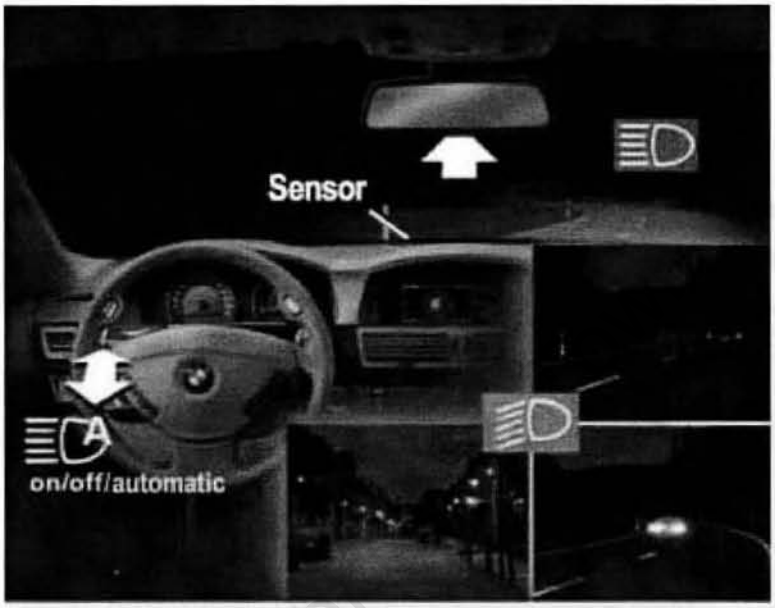
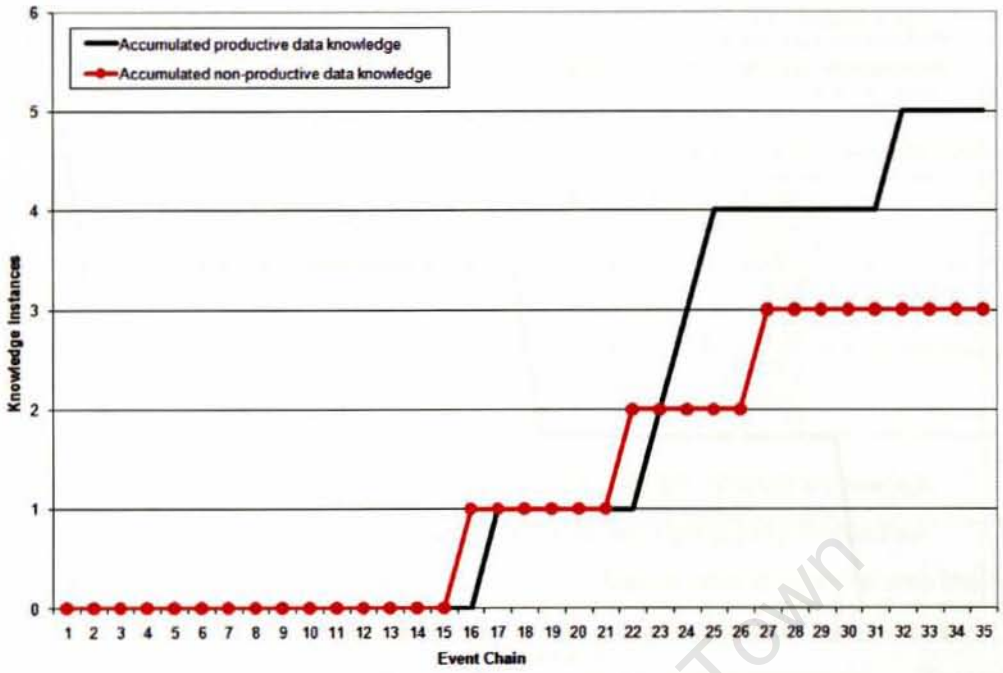
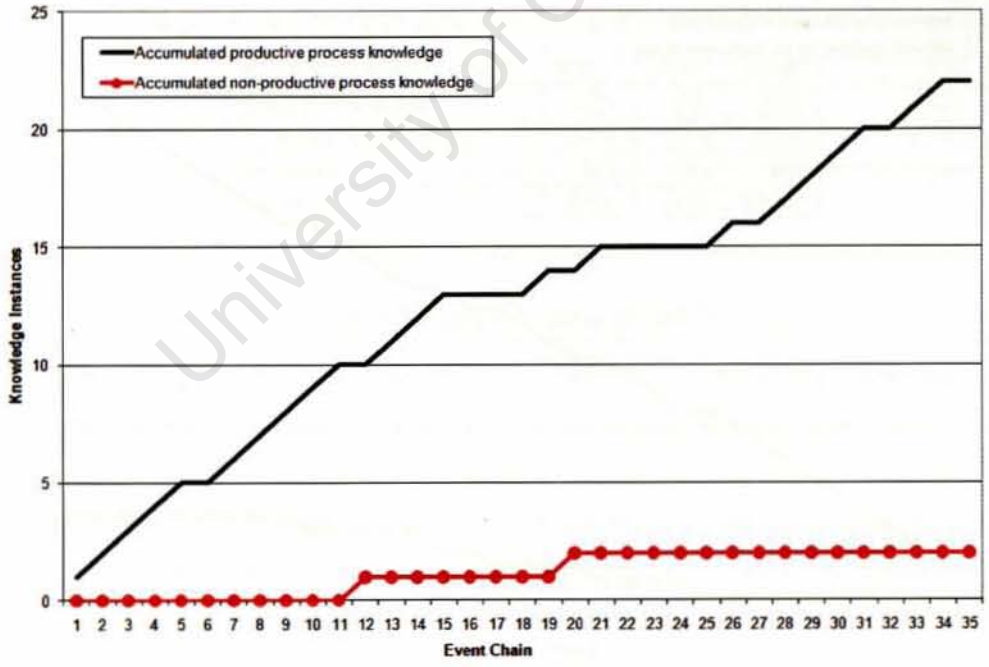


Figure 5.3.6: Installation of the Automation Headlight Dimmer (Project P2-6).

Unlike most of the other projects, which tended to begin with data knowledge, this team chose not to research the components, and this is evident in the lack of initial data knowledge (see Graph 5.2.6 (a)). Instead, productive process knowledge (see Graph 5.2.6 (b)) occurred from the start because the team began writing software right away. Furthermore, this team used Micromonitor directly instead of using the uCLinux operating system, which most of the other teams used. Apart from minor difficulties with using Micromonitor, around event chains 12 and 20, everything went fairly well. The team managed to speedily produce a simple menu interface at event chain 6 without any difficulties (see Graph 5.2.6 (c)). At event chain 17, the team got the PCF8591 operating in ADC mode. At event chain 31, the Automation Headlights Dimmer product was functional. Project P2-6 showed a divergence trend between productive and non-productive knowledge occurrences, similar to ones seen in the other projects.

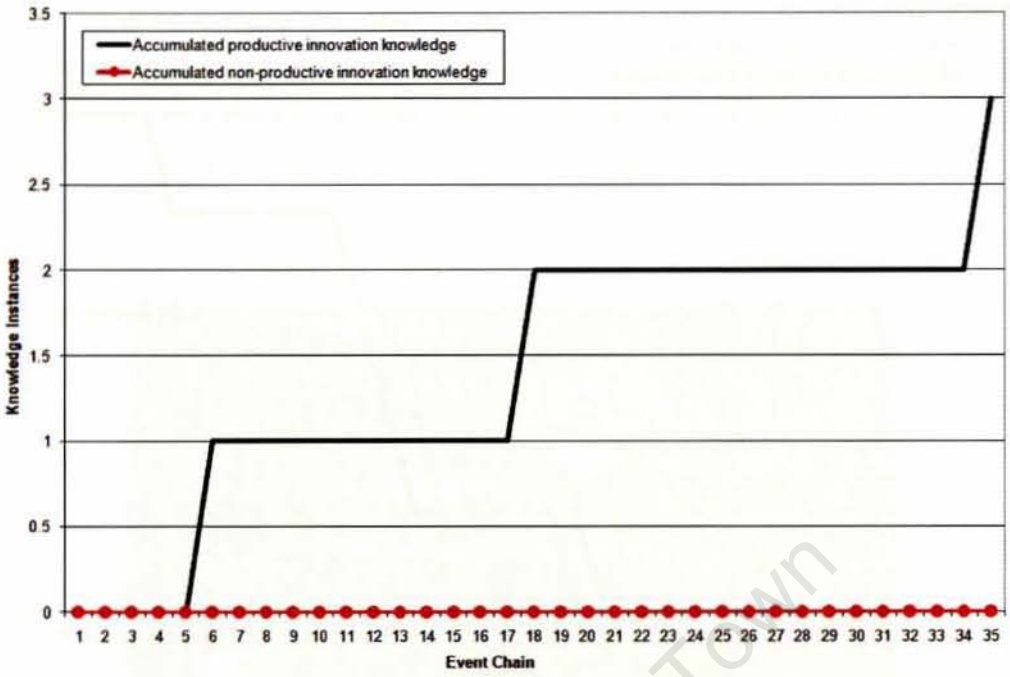


Graph 5.2.6 (a): Data knowledge in P2-6.

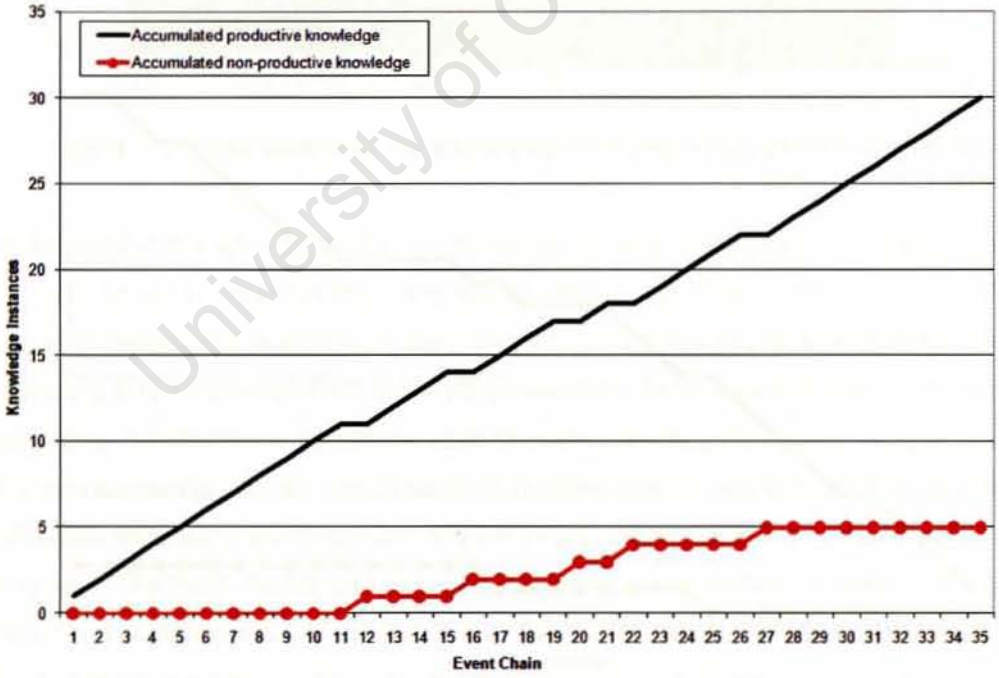


Graph 5.2.6 (b): Process knowledge in P2-6.

University of Cape Town



Graph 5.2.6 (c): Innovation knowledge in P2-6.



Graph 5.2.6 (d): Productive and non-productive knowledge in P2-6.

University of Cape Town

Table 5.2.6 (a) shows moderate levels of non-productive data knowledge, low levels of non-productive process and no non-productive innovation knowledge occurrences. 100% of the innovation knowledge was productive.

Table 5.2.6 (a): Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-6	PK	NPK	Total
Data Knowledge	63%	38%	100%
All Process Knowledge	92%	8%	100%
Role	25%	8%	33%
Logistics	21%	0%	21%
Engineering methods	46%	0%	46%
Innovation Knowledge	100%	0%	100%

Table 5.2.6 (b) shows that innovation comprised only 9% of total knowledge occurrences, thus indicating that this project was not particularly innovative. Nevertheless the team did produce a working prototype, and they were also the first team that managed to get together an initial operational system. The team was awarded a combined score of 94% for their prototype and demonstration. The high score was awarded on the basis of their competent demo presentation and the functionality of their prototype.

Table 5.2.6 (b): Proportions of data, process and innovation knowledge in P2-6.

Knowledge Type: P2-6	PK	NPK	Total
Data Knowledge	14%	9%	23%
Process Knowledge	63%	6%	69%
Innovation	9%	0%	9%
TOTALS	86%	14%	100%

University of Cape Town

5.2.7 P2-7 Field Sensor for Maglev Trains (FSMT)

This team designed a small magnetic field sensor, one of the subsystems used in a magnetic levitation (Maglev) train. In order to sense the magnetic fields, they designed a custom circuit, which contained an inductor coil and an ADC among other commonly used circuit components. These sensors were designed to be placed along the undercarriage of a (hypothetical) Maglev train. Components used by Project P2-7 included: the PCF8691 ADC, the LF198 (a sample and hold IC) and the KMZ10A magnetic field sensor. The team planned to use the SAK82C900 Infineon CAN controller via QSPI, but were not able to accomplish this.

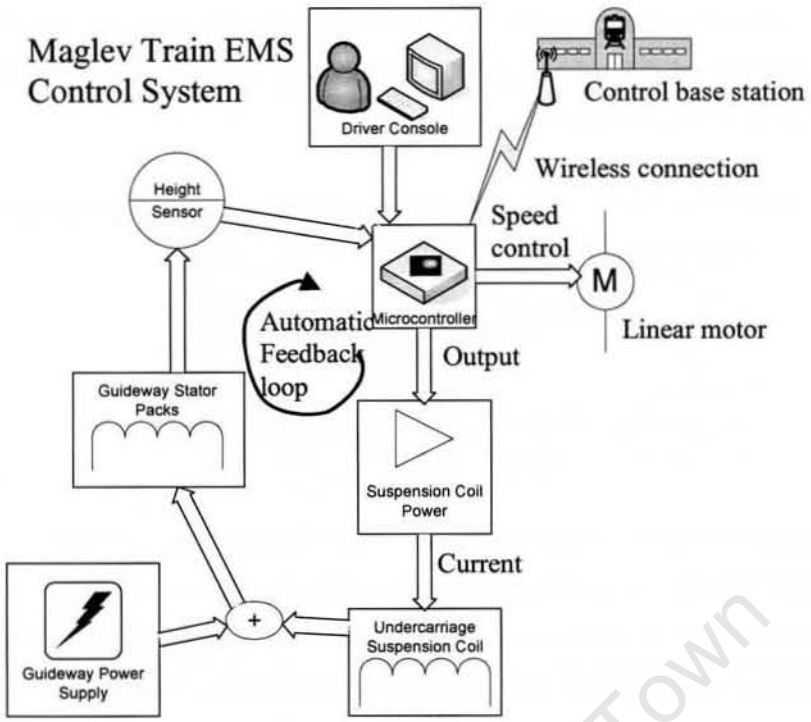
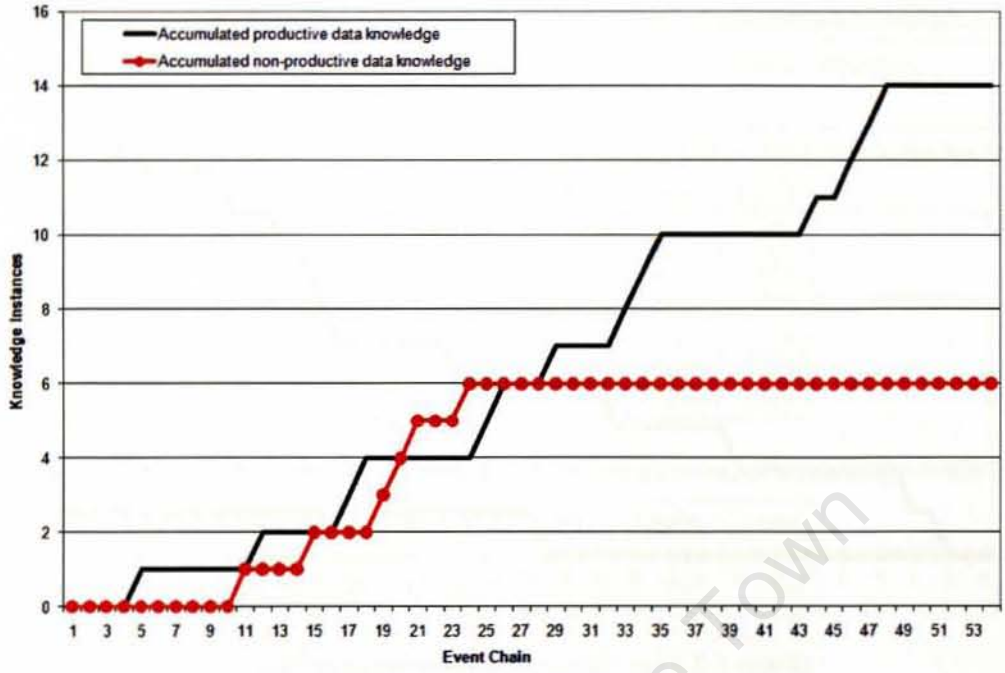


Figure 5.2.7: Component interconnection diagram of Field Sensor for Maglev Trains (Project P2-7).

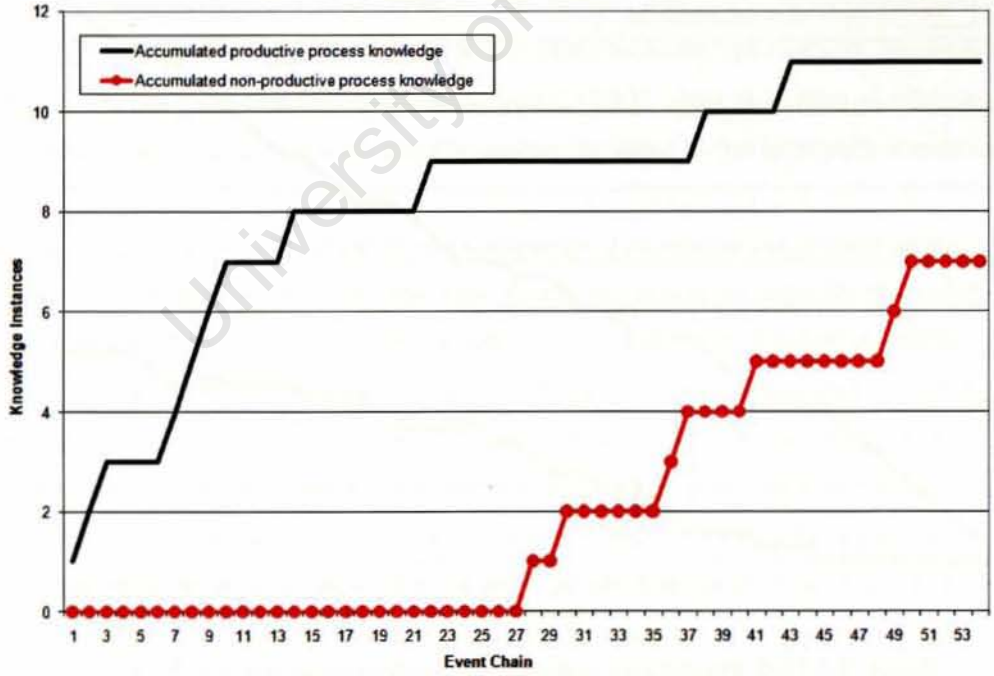
The group corrupted the flash memory device, and therefore needed to read up about the MacRagior Raven and the flash memory device in order to clear and reformat the flash memory and to reinstall the boot loader. This corruption occurred while they were trying to install the uCLinux RAM disk and kernel in flash. This incident produced non-productive data knowledge between event chains 15 – 24 (see Graph 5.2.7 (a)). The earlier non-productive data knowledge occurrence (at event chain 11) had to do with the connection between the ADC and microcontroller.

The project team erroneously followed methods described in a web-based manual concerning use of a BDI2000 JTAG device; except that they applied the methods to a different device, the MacRagior Raven JTAG device. This account for non-productive process knowledge between event chains 27 – 49 (see Graph 5.2.7 (b)). Once the team realised that the BDI2000 and Raven were not totally compatible, they came up with their own approach that worked. Despite these difficulties encountered, the team began to produce innovation knowledge from quite early in the project (event chain 3) and had only one instance (event chain 26) of non-productive innovation knowledge (see Graph 5.2.7 (c)). This occurrence was due to the team's changeover from Micromonitor to uCLinux. This project also shows a clear divergence trend in the productive vs. non-productive graph (Graph 5.2.7 (d)).

University of Cape Town

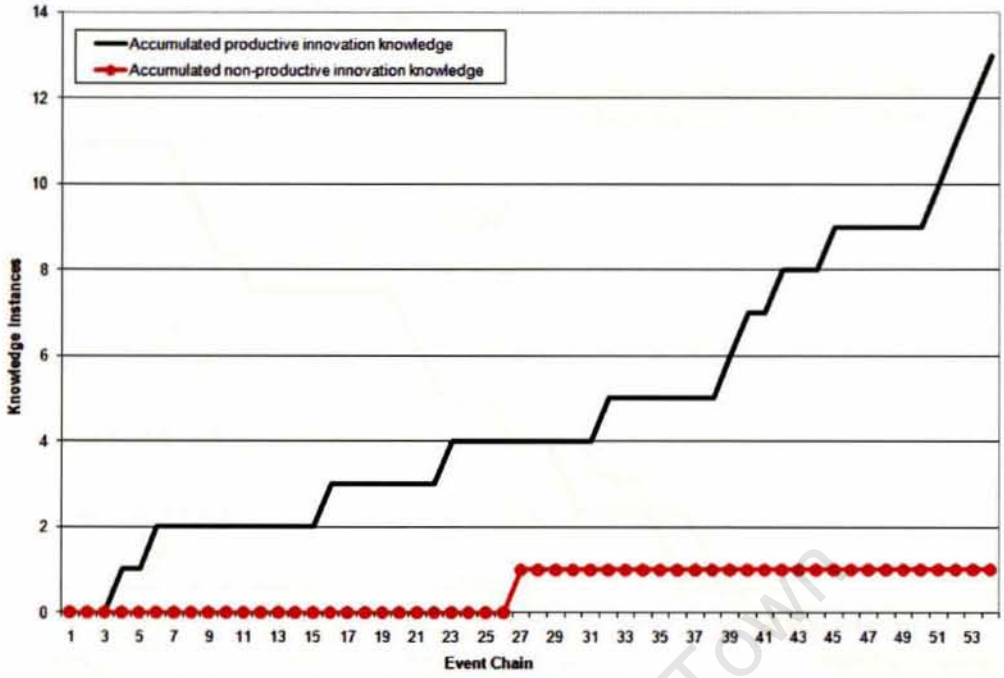


Graph 5.2.7 (a): Data knowledge in P2-7.

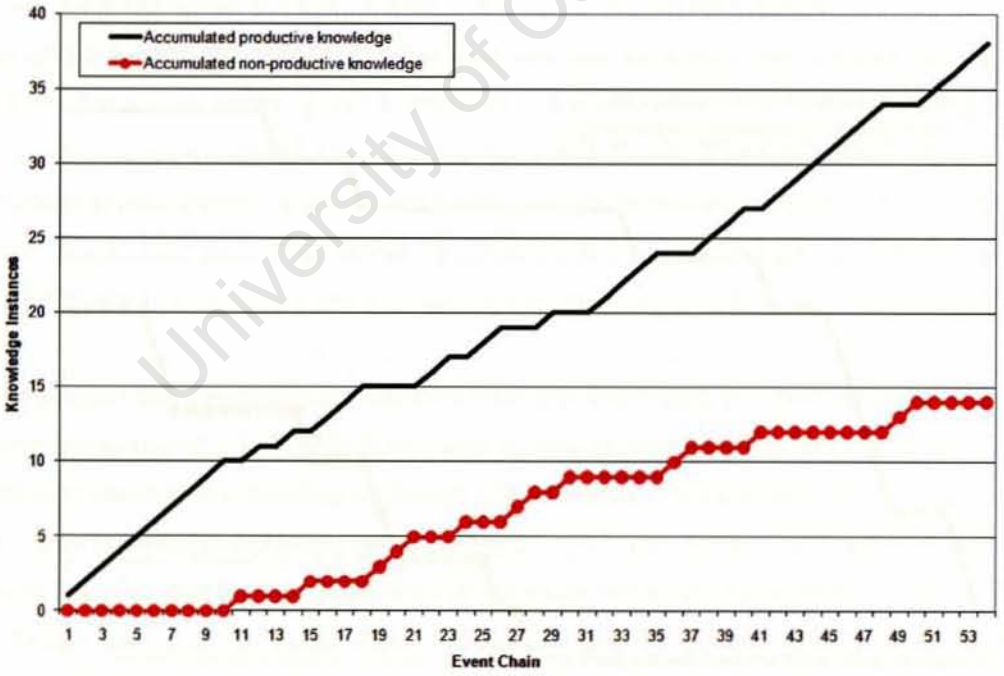


Graph 5.2.7 (b): Process knowledge in P2-7.

University of Cape Town



Graph 5.2.7 (c): Innovation knowledge in P2-7.



Graph 5.2.7 (d): Productive and non-productive knowledge in P2-7.

Table 5.2.7 (a) shows high levels of productive knowledge occurrences in this project. The lack of non-productive role and logistics knowledge occurrences is

University of Cape Town

indicative of good project leadership and team work. Table 5.2.7 (b) shows an even distribution of productive knowledge across knowledge types and an average distribution of productive and non-productive knowledge occurrences. The team was awarded an overall score of 89.75% for their prototype and demonstration.

Table 5.2.7 (a): Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-7	PK	NPK	Total
Data Knowledge	70%	30%	100%
All Process Knowledge	61%	39%	100%
Role	17%	0%	17%
Logistics	11%	0%	11%
Engineering methods	33%	39%	72%
Innovation Knowledge	93%	7%	100%

Table 5.2.7 (b): Proportions of data, process and innovation knowledge in P2-7.

Knowledge Type: P2-7	PK	NPK	Total
Data Knowledge	27%	12%	38%
Process Knowledge	21%	13%	35%
Innovation	25%	2%	27%
TOTALS	73%	27%	100%

5.2.8 P2-8 Cordless Stereo (CST)

The Cordless Stereo project involved developing an FM radio repeater that could be tuned to rebroadcast an FM channel as a Bluetooth digital sound transmission. The components used by the team were: an RF radio receiver, a Bluetooth transceiver module (in the form of a DIP chip), and two PCF8591 units operating in ADC mode. The team intended to use a Bluetooth headset to listen to the Bluetooth transmission.

The team consistently produced data knowledge throughout the project as they researched Bluetooth and FM radio (see Graph 5.2.8 (a)). In order for the product to be constructed as planned, it was necessary for the team to determine how to control the Bluetooth module. However, they did not fully accomplish this task – and their attempt to solve this problem accounts for the majority of occurrences of non-productive process knowledge in event chains 15 – 41 (see Graph 5.2.8 (b)). The team divided the work into three major parts: one team member focused on understanding Bluetooth, the second team member focused on connecting the radio receiver and sampling the stereo 8-bit sound data, while the third team member focused on coding the embedded software, which integrated a menu, the sound sampling routines, and the BlueTooth transmission routines. This third group member was responsible for most of the productive process knowledge, while the other two members produced most of the data knowledge.

University of Cape Town

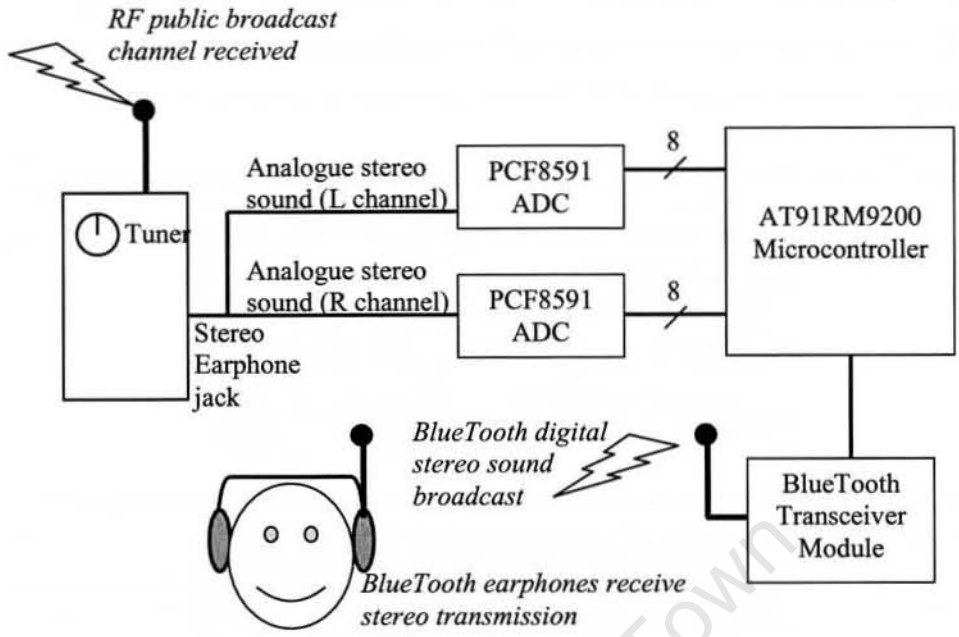
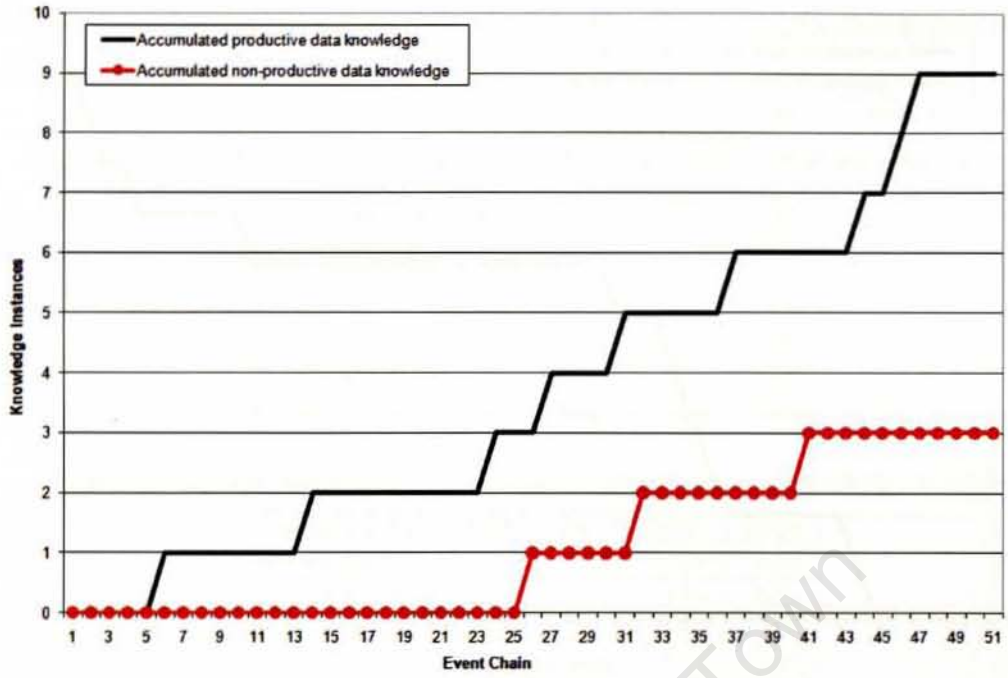


Figure 5.2.8: Concept drawing for the Cordless Stereo (Project P2-8).

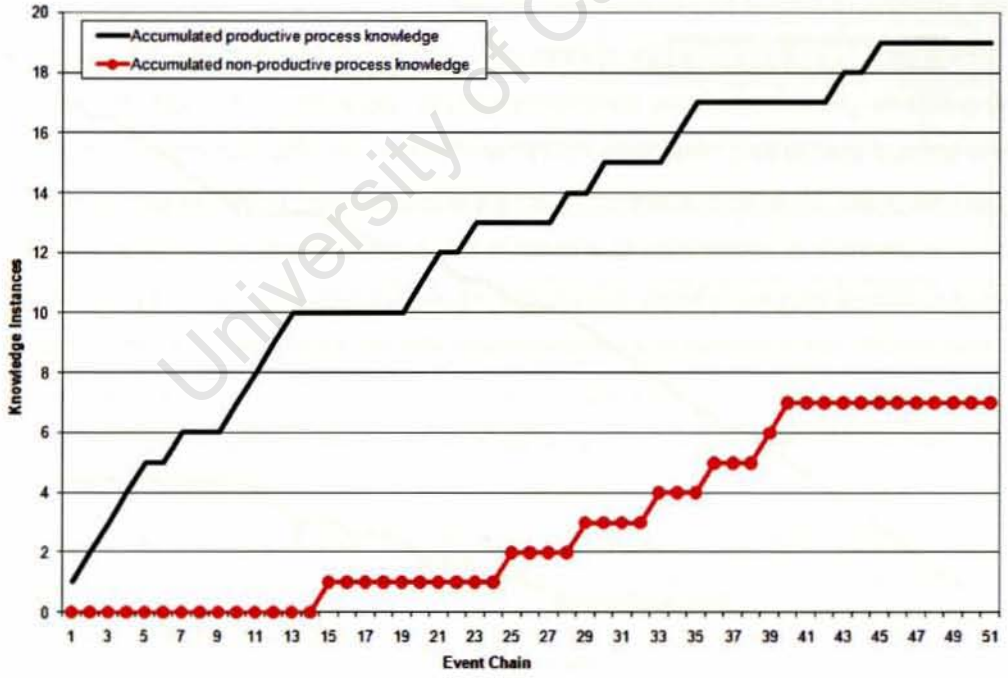
There was no occurrence of non-productive innovation knowledge. The third team member was responsible for the initial innovation (event chains 7 – 21) in which the embedded software was designed. The team then came together at the point where the innovation graph levels out (event chains 21- 40; see Graph 5.2.8 (c)); at this point they were producing productive process knowledge for connecting the RF receiver to the Microprocessor, but were producing non-productive knowledge with regard to connecting the Microprocessor to the Bluetooth module. The rise in innovation knowledge at the end of the project involved directing the sampled RF digitised sound to the comport, since the team ran out of time in their attempt to use the Bluetooth module.

The productive vs. non-productive knowledge occurrence graph (Graph 5.2.8 (d)) indicates that, overall, there was a divergence tendency, with the occurrences of productive knowledge exceeding those of non-productive knowledge.

University of Cape Town

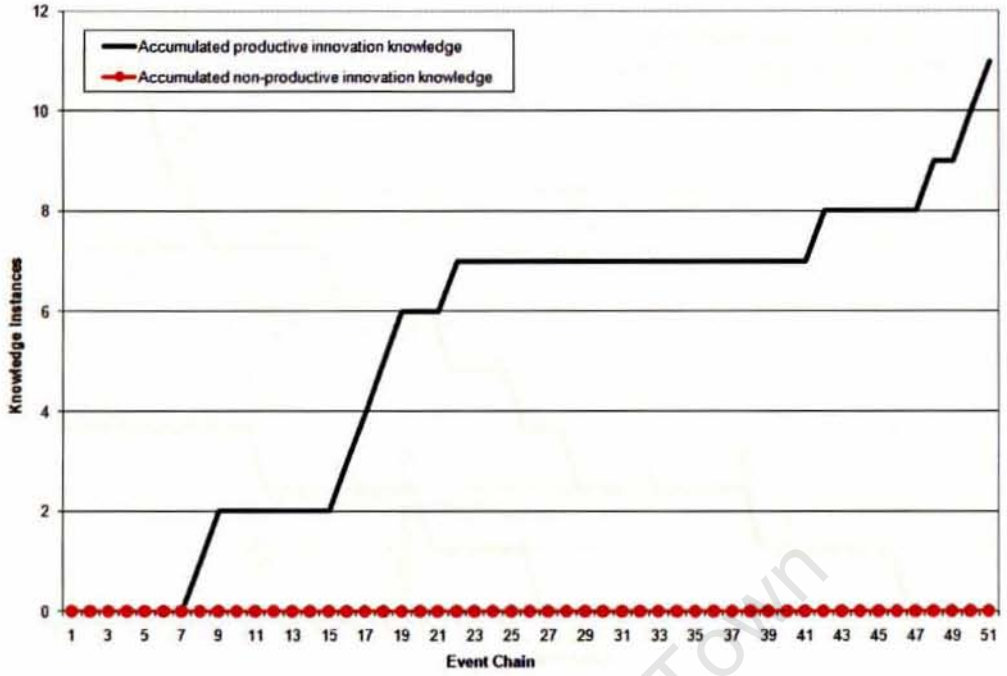


Graph 5.2.8 (a): Data knowledge in P2-8.

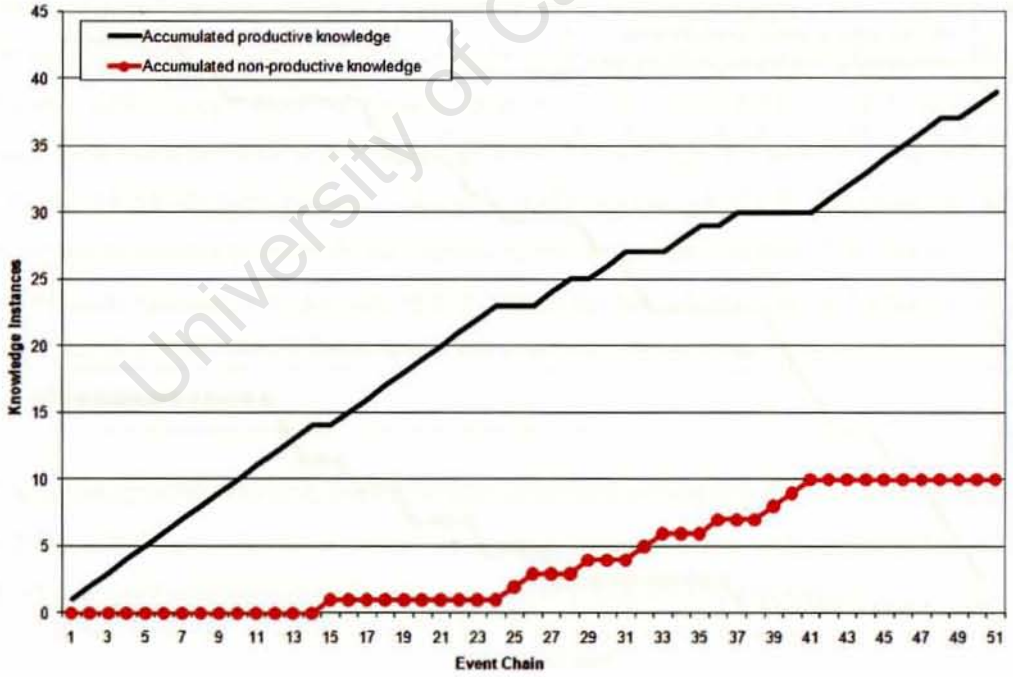


Graph 5.2.8 (b): Process knowledge in P2-8.

University of Cape Town



Graph 5.2.8 (c): Innovation knowledge in P2-8.



Graph 5.2.8 (d): Productive and non-productive knowledge in P2-8.

Table 5.2.8 (a) indicates moderate levels of productive and non-productive knowledge occurrences within each of the categories. Table 5.2.8 (b) indicates that there was a higher occurrence of process knowledge than of other knowledge types

University of Cape Town

in this project. This was largely due to the team's attempts to communicate with the Bluetooth module. The team achieved a combined score of 80% for their prototype and demonstration.

Table 5.2.8 (a): Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-8	PK	NPK	Total
Data Knowledge	75%	25%	100%
All Process Knowledge	73%	27%	100%
Role	15%	0%	15%
Logistics	8%	0%	8%
Engineering methods	50%	27%	77%
Innovation Knowledge	100%	0%	100%

Table 5.2.8 (b): Proportions of data, process and innovation knowledge in P2-8.

Knowledge Type: P2-8	PK	NPK	Total
Data Knowledge	18%	6%	24%
Process Knowledge	39%	14%	53%
Innovation	22%	0%	22%
TOTALS	80%	20%	100%

5.2.9 P2-9 Central Alarm Clock (CAC)

The Central Alarm Clock system consists of a central control unit based in a common room in the home (e.g., the lounge). The central control unit is able to activate alarm clocks remotely throughout the house. The system allowed the times of all alarm clocks in the home to be changed and synchronized centrally, thereby enabling the user to set alarms for different rooms from the central unit. The central control unit provided a menu-driven user interface that was connected via a PC using an RS232 ASCII terminal. Each alarm clock in the system was connected to the central control unit via an *ad hoc* RF network (although the project initially planned to utilise a mains modem). The main electronic components used by this project were: the Unigen UGWR2US RF Transceiver, the NE566 (in square wave generation mode), the DS1307 Dallas Real-time clock (RTC) module, and a pseudo-electric beeper.

The group successfully developed the initial outline of the embedded software application for the alarm clocks and central controller. Non-productive process knowledge occurred at event chains 13 – 14; what the team was doing was attempting to drive an 8 ohm speaker from a single programmable output bit on the microcontroller and to develop different tones. They decided to simplify things instead by using a pseudo-electric beeper (event chain 25 onwards). Process knowledge developed towards the end of the project did not affect the innovation knowledge

University of Cape Town

produced previously (see Graph 5.2.9 (c)), because they simply replaced one component with another, and changed the way in which it was accessed.

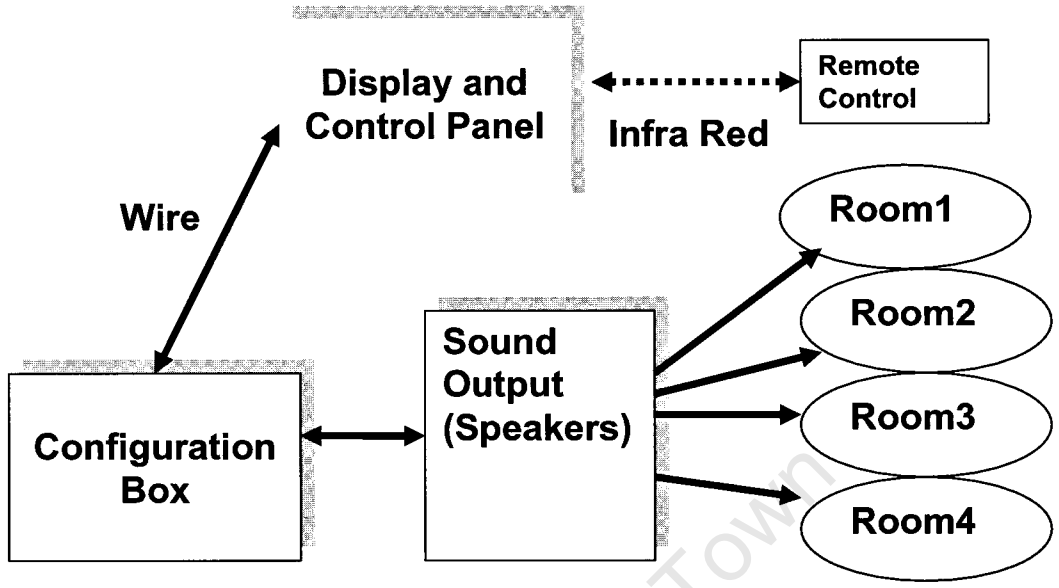


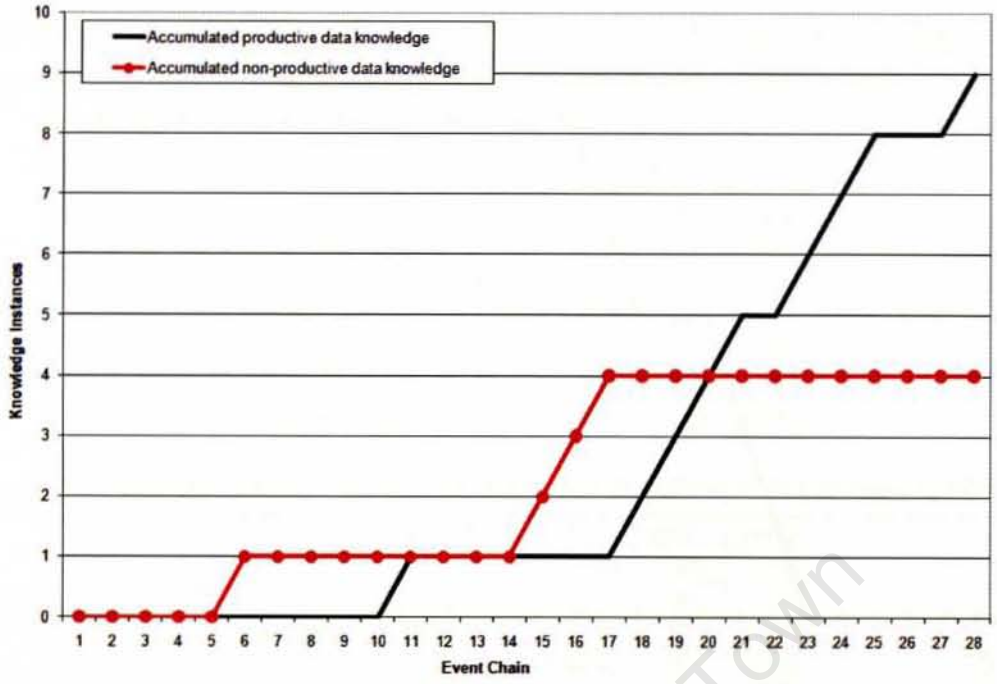
Figure 5.2.9: Installation diagram for Central Alarm Clock (Project P2-9).

There were no non-productive innovation knowledge occurrences. The innovation appeared early in the project (event chain 6), as the team produced the software and became able to communicate between the central controller and a remote alarm clock. They did not draw much on data knowledge for innovation knowledge occurrences.

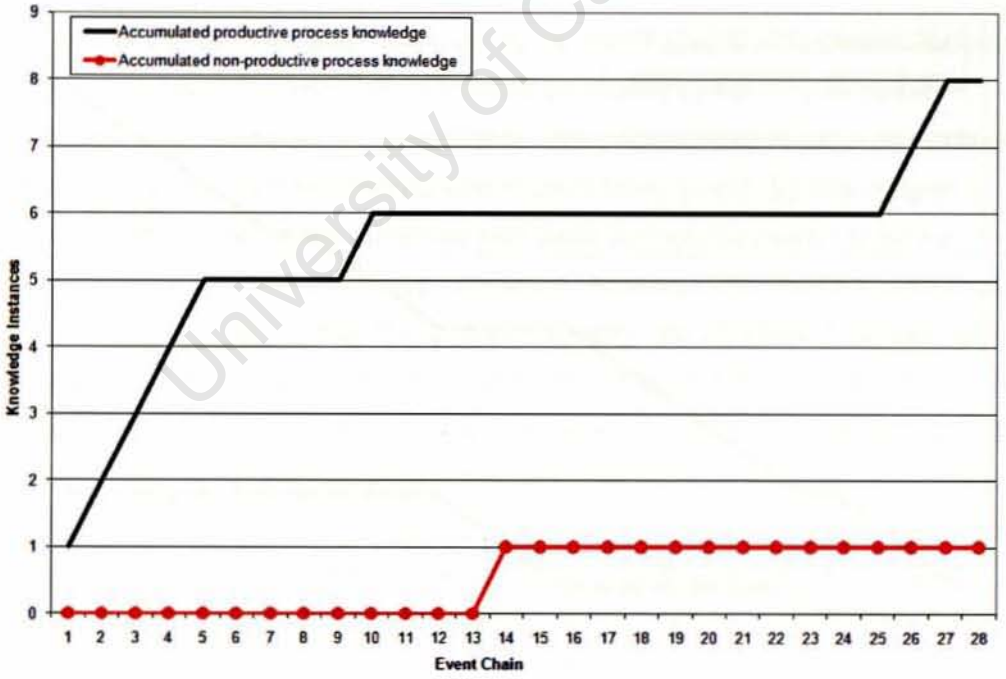
Graph 5.2.9 (d) shows occurrences of both productive and non-productive data knowledge; at event chain 17, the non-productive occurrences cease and the productive knowledge occurrences continue, showing the usual divergence trend.

Table 5.2.9 (a) indicates that there were moderate to low levels of non-productive knowledge. The team experienced no problems with regard to roles or logistics.

University of Cape Town

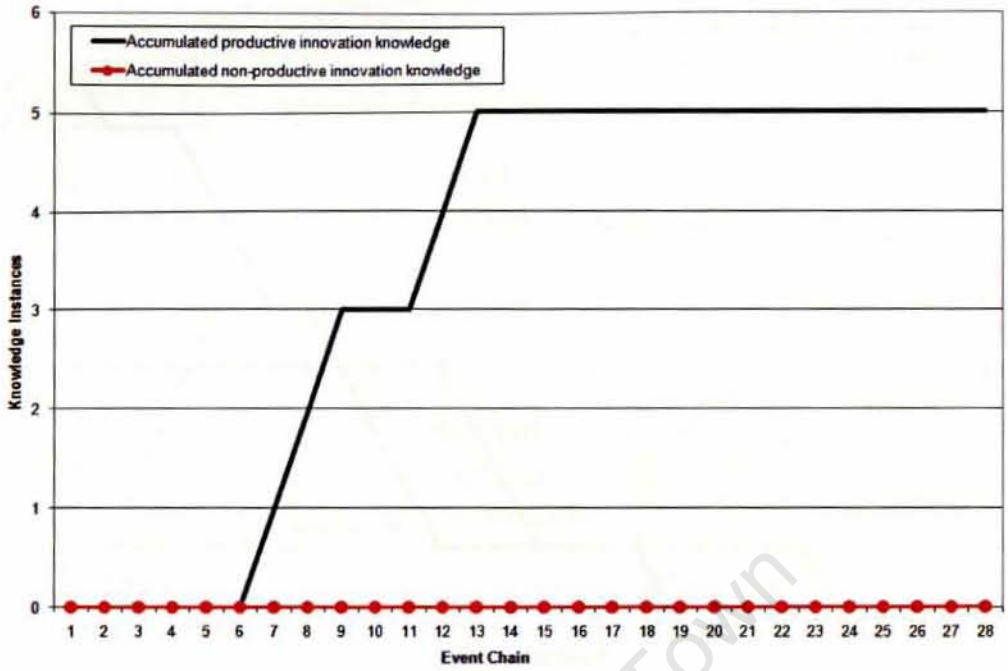


Graph 5.2.9 (a): Data knowledge in P2-9.

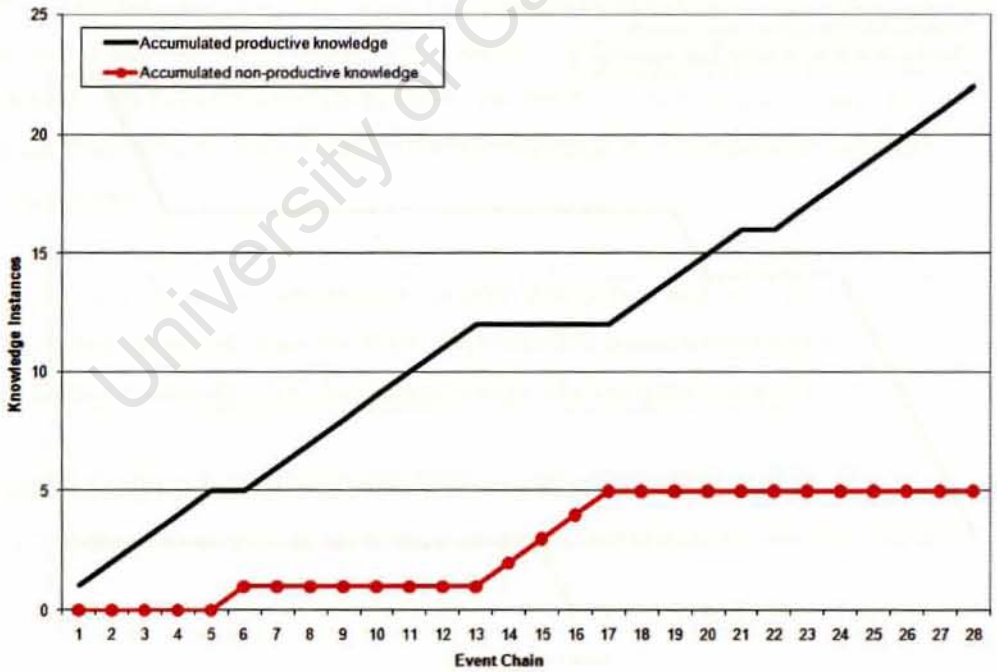


Graph 5.2.9 (b): Process knowledge in P2-9.

University of Cape Town



Graph 5.2.9 (c): Innovation knowledge in P2-9.



Graph 5.2.9 (d): Productive and non-productive knowledge in P2-9.

Table 5.2.9 (b) indicates that data knowledge occurrences were higher than average, and process knowledge occurrences slightly lower than average, while innovation

University of Cape Town

knowledge occurrences were about average. There was a high overall level of productive knowledge, but only an average occurrence of innovation knowledge. The team scored 86.5% for their prototype and demonstration.

Table 5.2.9 (a): Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-9	PK	NPK	Total
Data Knowledge	69%	31%	100%
All Process Knowledge	89%	11%	100%
Role	33%	0%	33%
Logistics	22%	0%	22%
Engineering methods	33%	11%	44%
Innovation Knowledge	100%	0%	100%

Table 5.2.9 (b): Proportions of data, process and innovation knowledge in P2-9.

Knowledge Type: P2-9	PK	NPK	Total
Data Knowledge	33%	15%	48%
Process Knowledge	30%	4%	33%
Innovation	19%	0%	19%
TOTALS	81%	19%	100%

University of Cape Town

5.2.10 P2-10 Voice Activation System (VAS)

The objective of the VAS was to turn on/off the mains supply of electronic appliances by voice command. Examples of appliances controlled by the VAS included: televisions, music systems, PCs, and lights. The initial concept design was rather ambitious, and involved one microphone in each room, which digitally relayed a recorded voice to the central computer that could turn appliances on or off by sending commands over a wireless connection. However, the resultant system was simpler and more elegant than the original concept – the prototype that was built comprised only two subsystems: a collection of 'Voice-activated Plugs' (or VAPs) and a central control computer (a PC). This team developed drawings that they called 'concept cartoons' to describe the operation of their system. Although most of their concept cartoons were done (by the innovation engineer) using pen on paper, they did reproduce a selection of the drawings as more professional digital images. Figures 5.2.10 (a) and 5.2.10 (b) graphically represent the operation of the system using a concept cartoon.

The team originally planned that the VAPs would be controlled (turned on or off) and programmed wirelessly by the control computer via a wireless link (the programming task involving two voice commands being recorded and digitized, viz. one command

to turn the appliance on, the other to turn it off), which would make it possible to upload the recordings to a VAP. The team decided to substitute the wireless connection with a wired Ethernet LAN to reduce development time for the prototype.

The team used the following components: LXT971A (an on-board Ethernet controller), a PCF8591 in ADC mode, the CSB337 Epson graphics controller, a CM741 microphone, as well as on-chip timers and on-chip counters. A PC central controller was connected to multiple linked switches via a LAN.

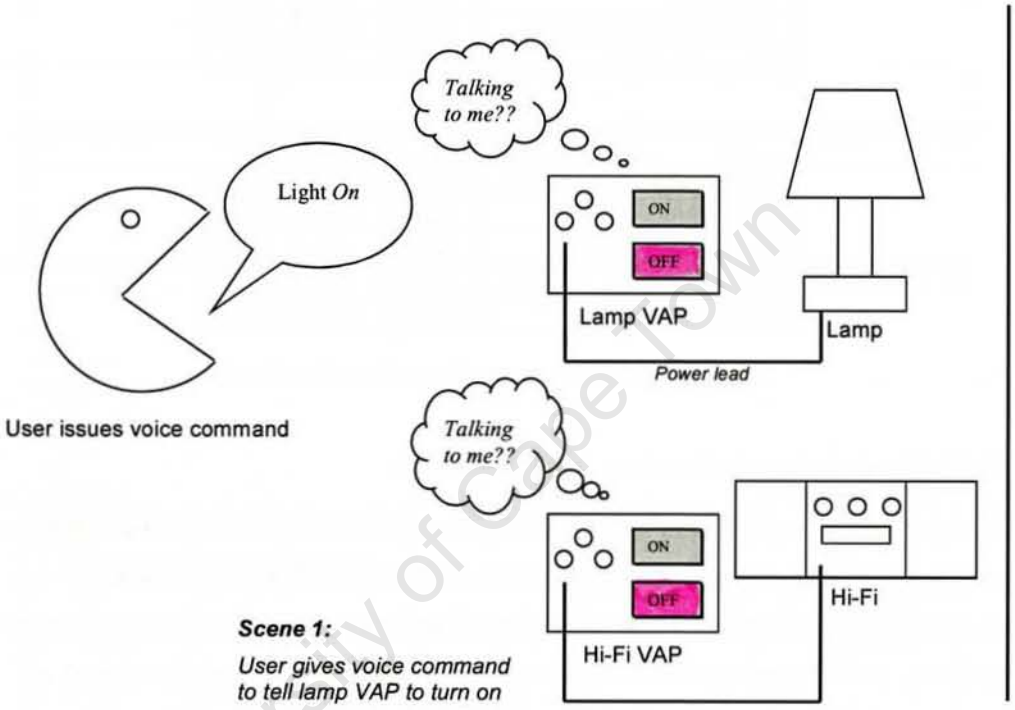


Figure 5.2.10 (a): First scene in the concept cartoon showing how the voice-activated plugs (VAPs) operate in the Voice Activation System (Project P2-10).

The first occurrences (event chains 6 – 7) of non-productive data knowledge had to do with problems encountered in understanding a module (which the team downloaded from the Internet) for parsing command strings. The non-productive occurrence between event chains 14 – 24 related to problems in understanding how to control the timers on a VAP. Instead of timers, they used on-board counters clocked by down-sampling the 133MHz master clock for the processor. The resultant data knowledge produced for using the timers was classified as non-productive. Further non-productive data knowledge occurred at event chain 32 – 33; this involved controlling pushbuttons (intended as a manual override for switches that did not respond to voice commands) and linking an interrupt to the pushbuttons.

University of Cape Town

The team started to build productive process knowledge at the start of the project; this was related to the GCC compiler and developing a TCP/IP debugging menu interface to control the voice activated switches. This menu interface was expected to respond to commands sent via the network (e.g. "LIGHT ON"). The menu was always active to allow for easy debugging.

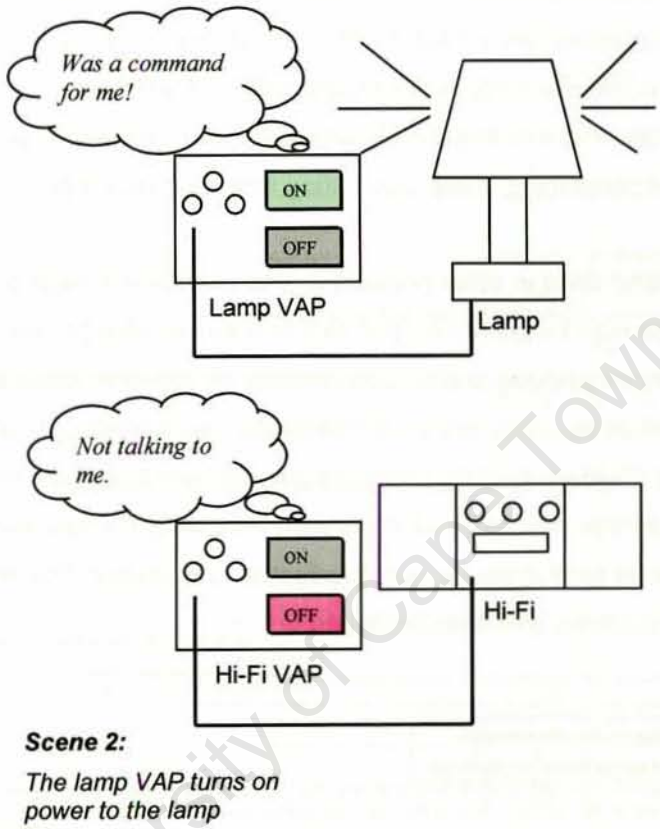


Figure 5.2.10 (b): Second scene of concept cartoon for operation of VAPs.

There were two instances of non-productive process knowledge (event chains 15 and 31), which had to do with using the PCF8591 ADC to sample voice commands. They encountered numerous difficulties in the task of sampling voice commands, such as matching voice sample rates on the VAPs and PC. Voice commands and microphone functionality were achieved only close to the end of the project.

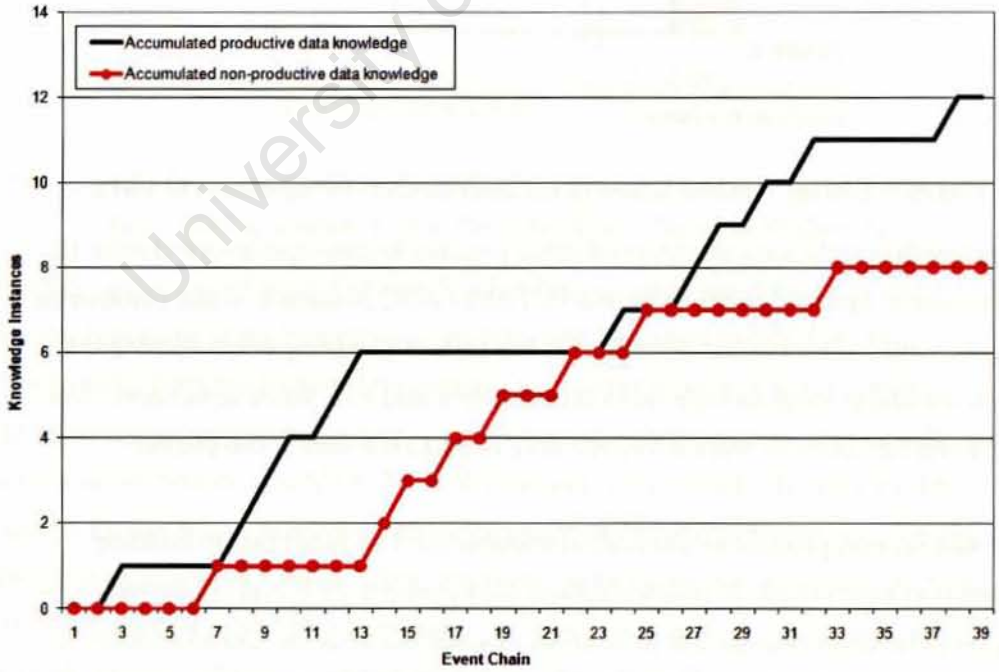
There was no non-productive innovation knowledge. The team began building innovation at event chain 3, which involved configuring a VAS and implementing query commands to change the responses to voice commands. Considerable innovation occurred between event chains 19 – 23 as the team successfully integrated solutions to turn an appliance on or off based on a time duration.

University of Cape Town

At event chain 34, the team put in place voice command overrides (to compensate for cases where the remote switch was unable to understand the command). On/off manual overrides were put in place prior to completing voice recognition.

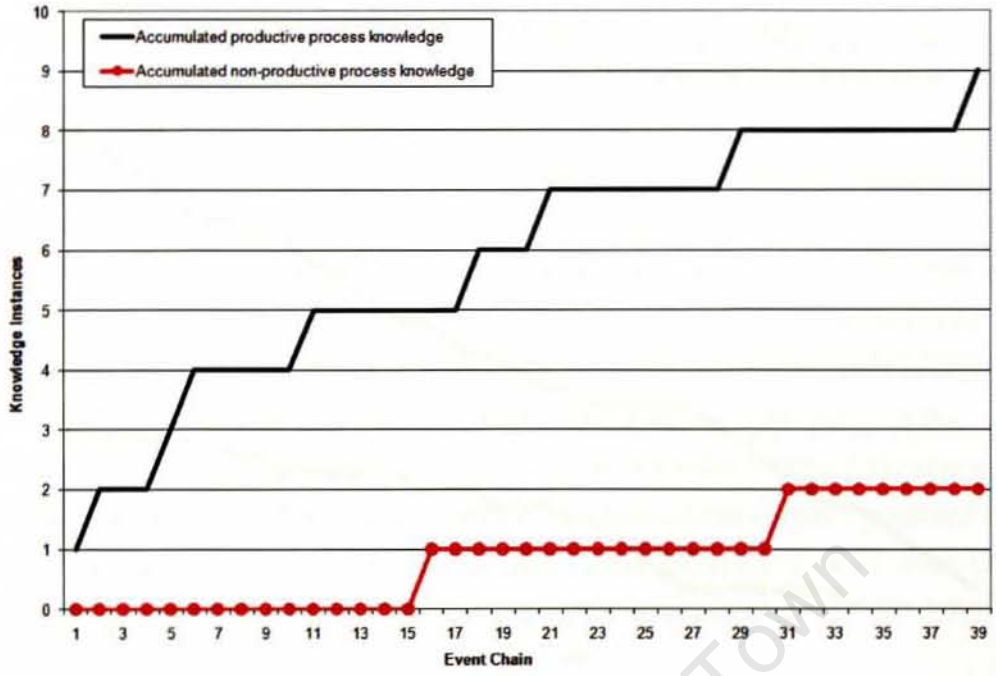
At the end of the project (event chains 35 – 37), partial success was made in making VAPs respond to voice commands (only short commands worked, such as 'ON', as long as they were pronounced similar to the original recording). Considering that voice recognition is an inherently difficult topic, that VAPs did respond to voices, and that the implemented voice recognition functionality was kept in the system, the innovative effort in performing voice recognition was considered productive.

The divergence trend seen in other projects is also evident in Graph 5.2.10 (d). Table 5.2.10 (a) indicates high levels of non-productive data knowledge, low levels of non-productive process knowledge and no occurrences of non-productive innovation knowledge, as well as no occurrences of non-productive knowledge with regard to roles and logistics. Table 5.2.10 (b) indicates average levels of data knowledge, but high levels of productive process and innovation knowledge. There was an average overall percentage of productive vs. non-productive knowledge. The team scored 80.75% for their prototype and demonstration.

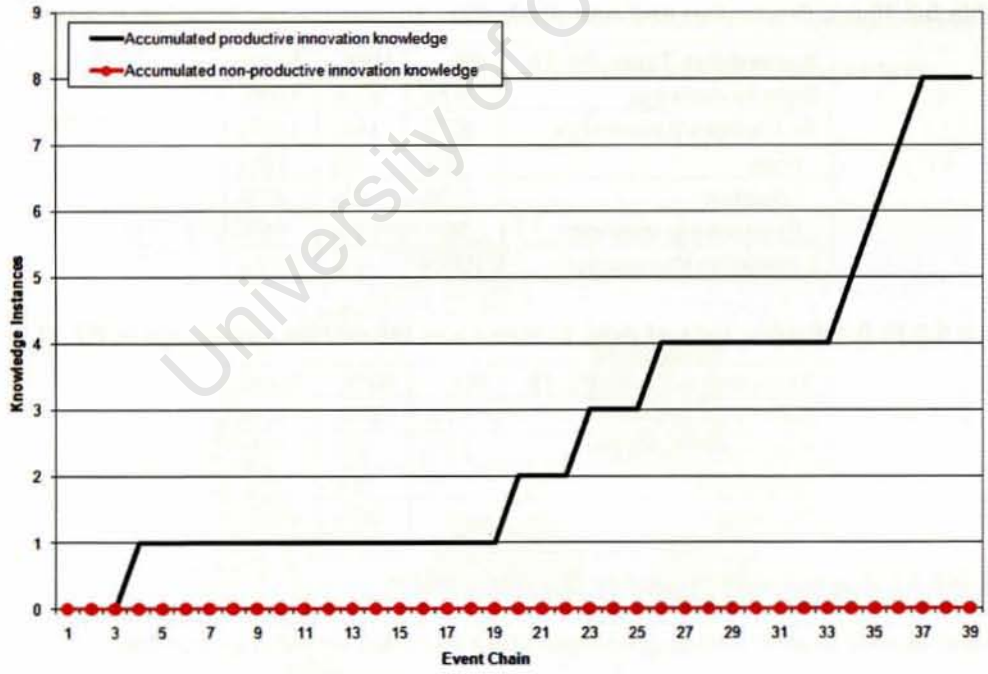


Graph 5.2.10 (a): Data knowledge in P2-10.

University of Cape Town

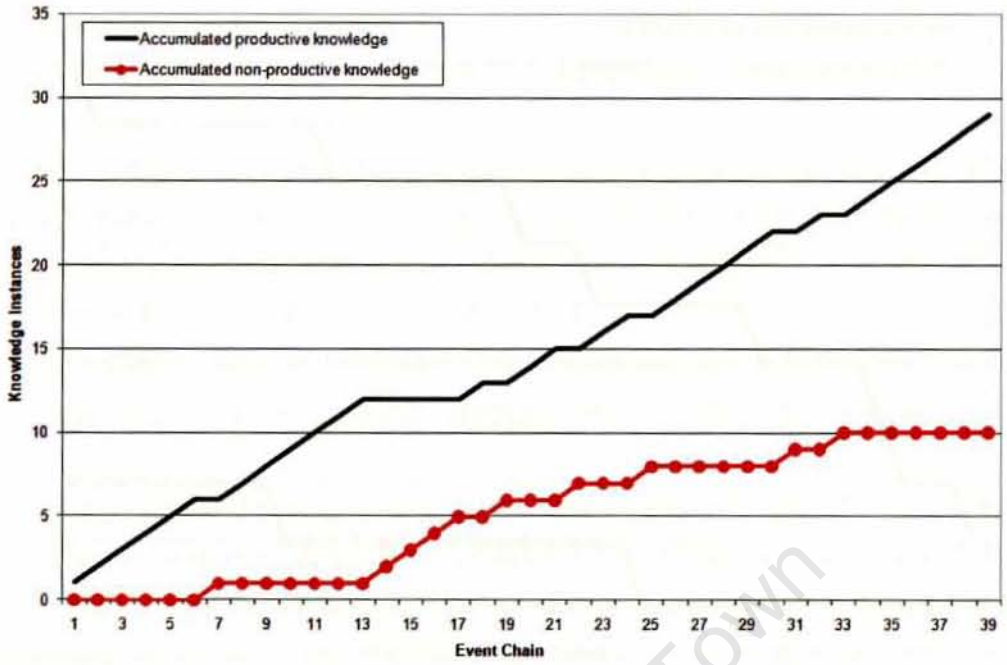


Graph 5.2.10 (b): Process knowledge in P2-10.



Graph 5.2.10 (c): Innovation knowledge in P2-10.

University of Cape Town



Graph 5.2.10 (d): Productive and non-productive knowledge in P2-10.

Table 5.2.10 (a): Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-10	PK	NPK	Total
Data Knowledge	60%	40%	100%
All Process Knowledge	82%	18%	100%
Role	18%	0%	18%
Logistics	27%	0%	27%
Engineering methods	36%	18%	55%
Innovation Knowledge	100%	0%	100%

Table 5.2.10 (b): Proportions of data, process and innovation knowledge in P2-10.

Knowledge Type: P2-10	PK	NPK	Total
Data Knowledge	31%	21%	51%
Process Knowledge	23%	5%	28%
Innovation	21%	0%	21%
TOTALS	74%	26%	100%

5.2.11 P2-11 Supermarket Query Device (SQD)

The Supermarket Query Device (illustrated by Figure 5.2.11) comprised the installation, in a supermarket, of a server to which SMS messages could be sent. The server would then reply to the caller with an SMS, telling the customer about 'specials' available in the store as well as where items were located. There would be two types of query: 1) queries with regard to 'specials' (special + keyword) and 2) queries regarding location (locate + keyword). Components used by the team

University of Cape Town

included: the Erickson T10 cellphone, the Motorola JK evaluation board with LCD panel, and a Falcom Twist MC35 GPRS modem (RS232 version).

The team needed to do Internet-based research on the Erickson T10 in order to gather data on how to enable the T10 to act as a device to send and receive SMS messages to/from the CSB337, which was acting as the supermarket query server. This team produced more non-productive than productive data knowledge (see Graph 5.2.11 (a)) due to the intensive research they needed to do on the Erickson T10. They were unable to get the SMS system to work, and this resulted in much of the data becoming unproductive. They had to find a way of understanding the communication protocol to control the T10 without using the Microsoft Windows software provided with the Erickson cellular telephone. They tried to implement their own driver for the T10, but ultimately abandoned the T10 (around event chain 19) and instead obtained a Falcom Twist MC35 GPRS modem.

Process knowledge occurred early on due to simulation and the sending of text messages. The team were able to send query messages directly to the CSB337, but not via a cell phone text message. At event chain 13 (see Graph 5.2.11 (b)), they found the solution to the query. Much of the process knowledge between event chains 20 – 35 involved getting the GPRS modem to send and receive text messages.

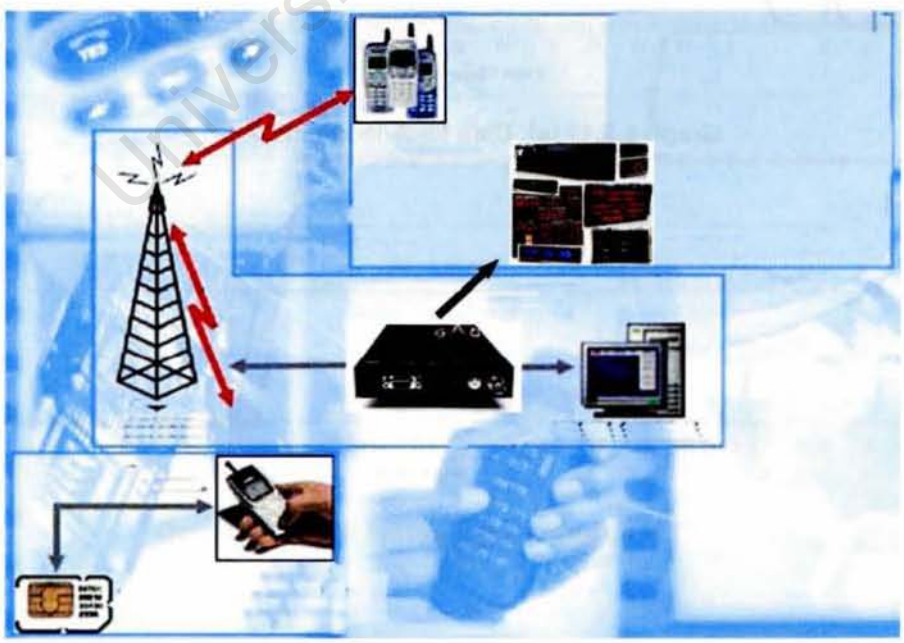
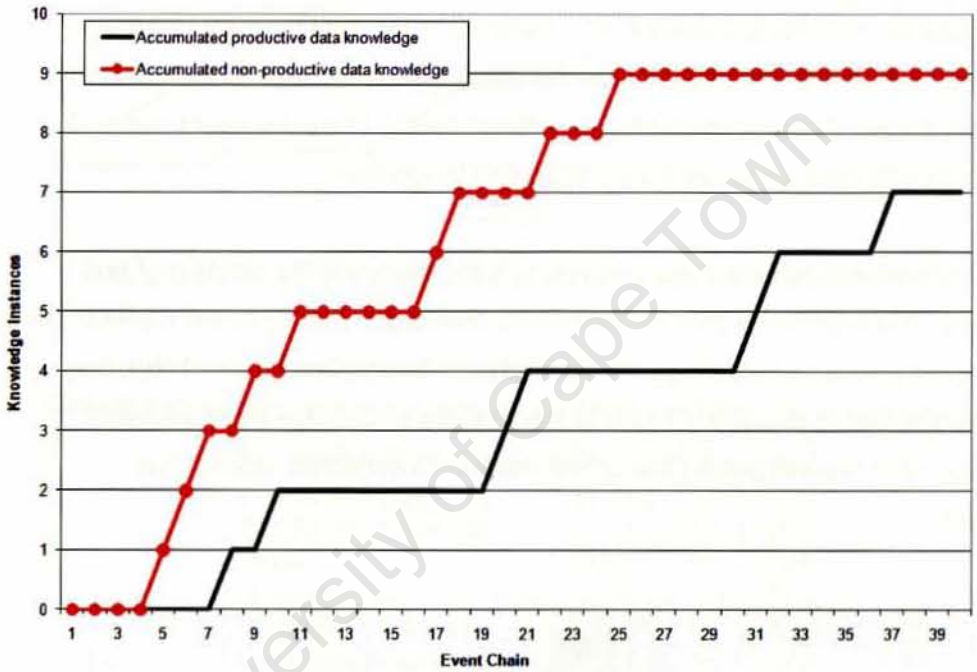


Figure 5.2.11: Concept diagram of Supermarket Query Device (Project P2-11).

University of Cape Town

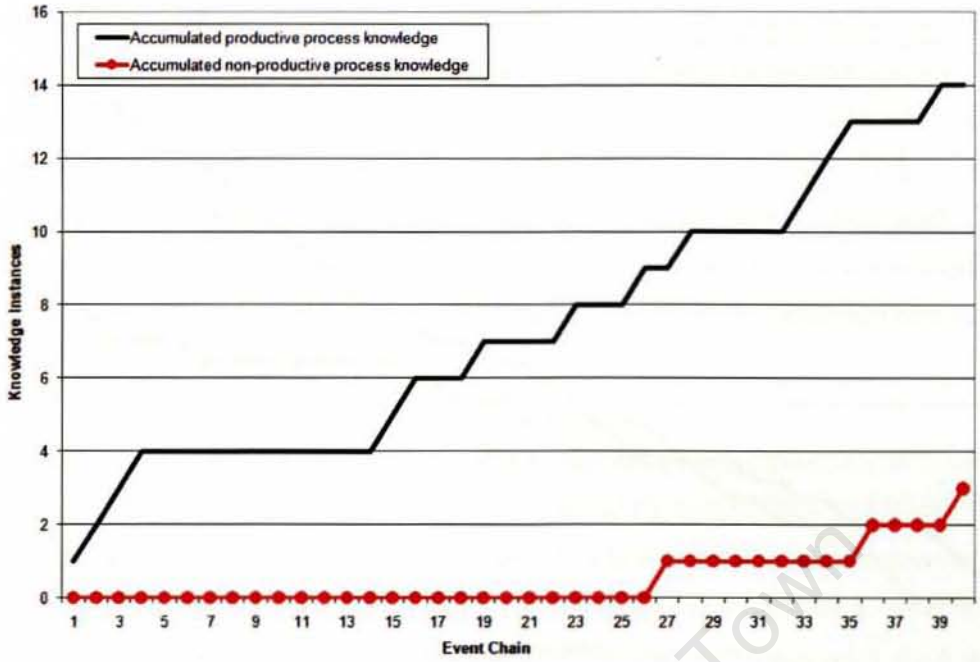
Event chain 29 (see Graph 5.2.11 (c)) indicates where the team successfully implemented the handling of the SMS messages. There were only two occurrences of non-productive innovation knowledge where the team replaced one component with another.

After the false start previously described, the divergence tendency noted on the other graphs emerges toward end of this project too (see Graph 5.2.11 (d)).

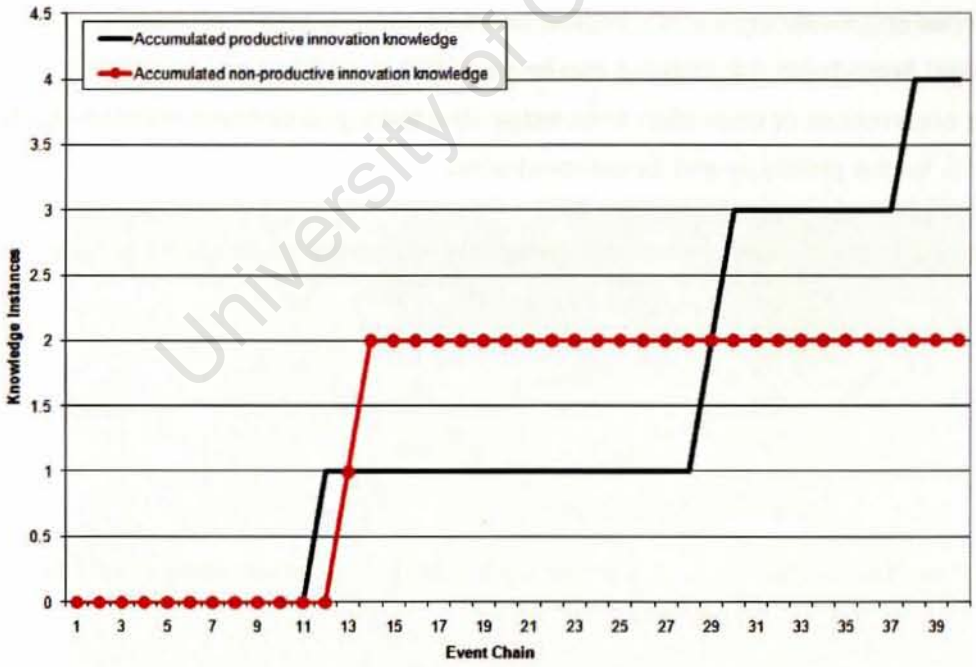


Graph 5.2.11 (a): Data knowledge in P2-11.

University of Cape Town

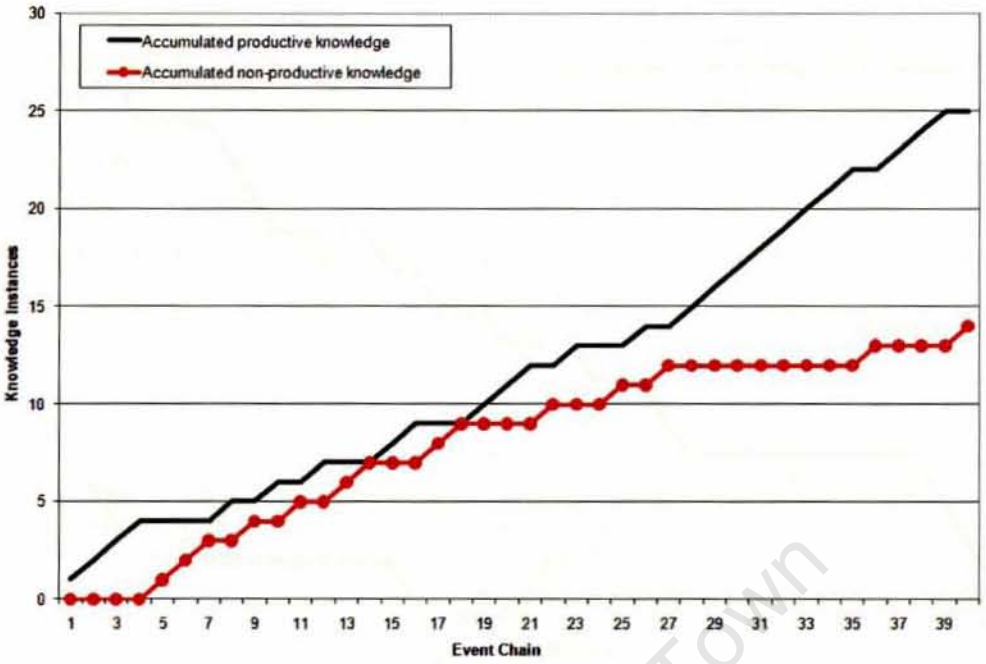


Graph 5.2.11 (b): Process knowledge in P2-11.



Graph 5.2.11 (c): Innovation knowledge in P2-11.

University of Cape Town



Graph 5.2.11 (d): Productive and non-productive knowledge in P2-11.

Table 5.2.11 (a) shows high levels of non-productive data and innovation knowledge (for reasons previously explained), but low levels of non-productive process knowledge. From Table 5.2.11 (b), it can be seen that this team had lower than average occurrences of innovation knowledge. The team achieved a combined score of 80.75% for the prototype and its demonstration.

Table 5.2.11 (a): Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-11	PK	NPK	Total
Data Knowledge	44%	56%	100%
All Process Knowledge	82%	18%	100%
Role	24%	0%	24%
Logistics	12%	0%	12%
Engineering methods	47%	18%	65%
Innovation Knowledge	67%	33%	100%

Table 5.2.11 (b): Proportions of data, process and innovation knowledge in P2-11.

Knowledge Type: P2-11	PK	NPK	Total
Data Knowledge	18%	23%	41%
Process Knowledge	36%	8%	44%
Innovation	10%	5%	15%
TOTALS	64%	36%	100%

University of Cape Town

5.2.12 P2-12 Personal Protection Device (PPD)

This device operated as a remote panic button. Instead of having to press a fixed panic button at a specific location, the user would be able to press the panic button on the Personal Protection Device (PPD). Receivers would then triangulate the user's position based on the strength of the RF transmission. Components used by the PPD team included LEDs, pushbuttons, an alarm siren and an improvised transistor-based circuit for simulating wireless communications. The team used an additional development tool, the SimIT open-source ARM instruction set simulator [SimIt-ARM, 2008].

In terms of data knowledge, the team researched different systems, some of which ended up as 'dead ends'. These dead ends resulted in most of the non-productive data knowledge that occurred in this project. The team had trouble establishing roles and allocating tasks. It was also difficult to mobilize the team members, most of whom were generally reluctant to become fully involved with the project. As a result, the team was not effective and this project was not entirely successful. There was only one event chain in which occurrences of innovation knowledge were noted (see Graph 5.2.12 (c), event chain 4). During this knowledge chain, the team successfully ran software on the CSB337, essentially transmitting an alarm message when the pushbutton was pressed. Graph 5.2.12 (d) shows an emerging divergence trend, but this only started at a late stage in the project.

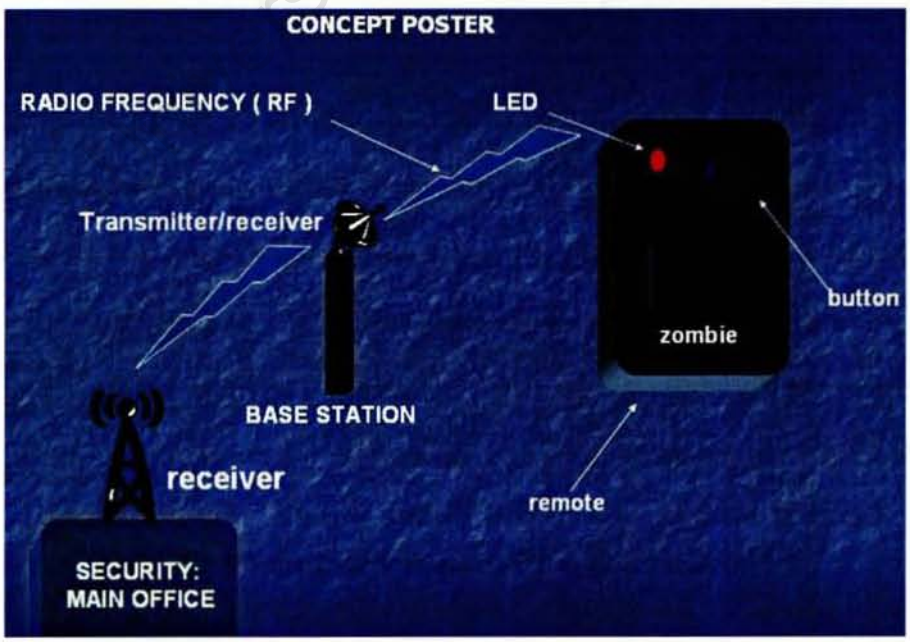
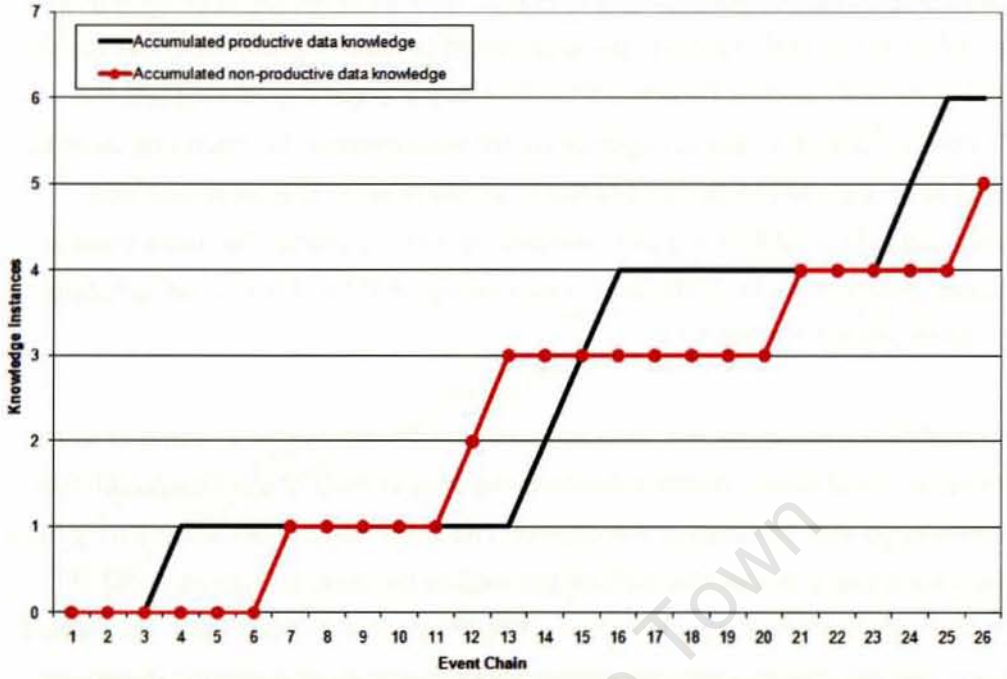
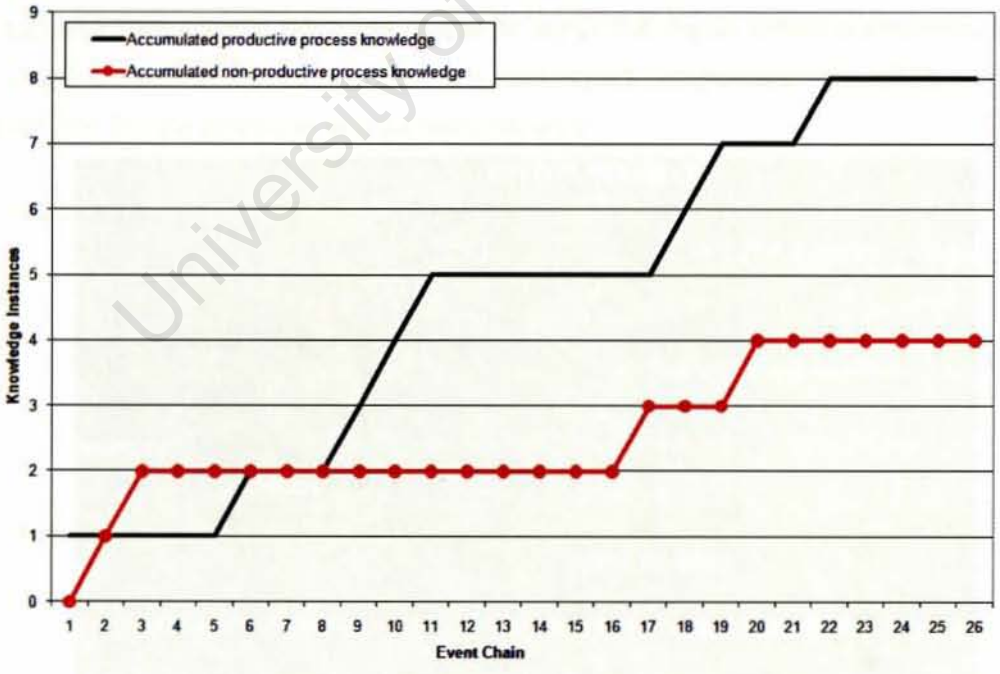


Figure 5.2.12: Concept poster for the Personal Protection Device (Project P2-12).

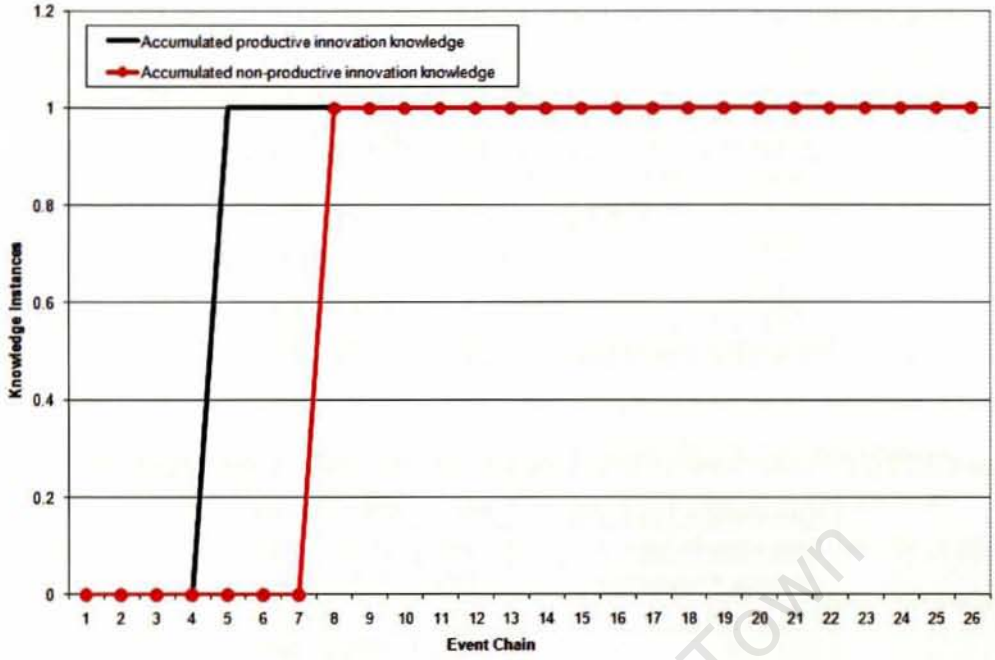


Graph 5.2.12 (a): Data knowledge in P2-12.

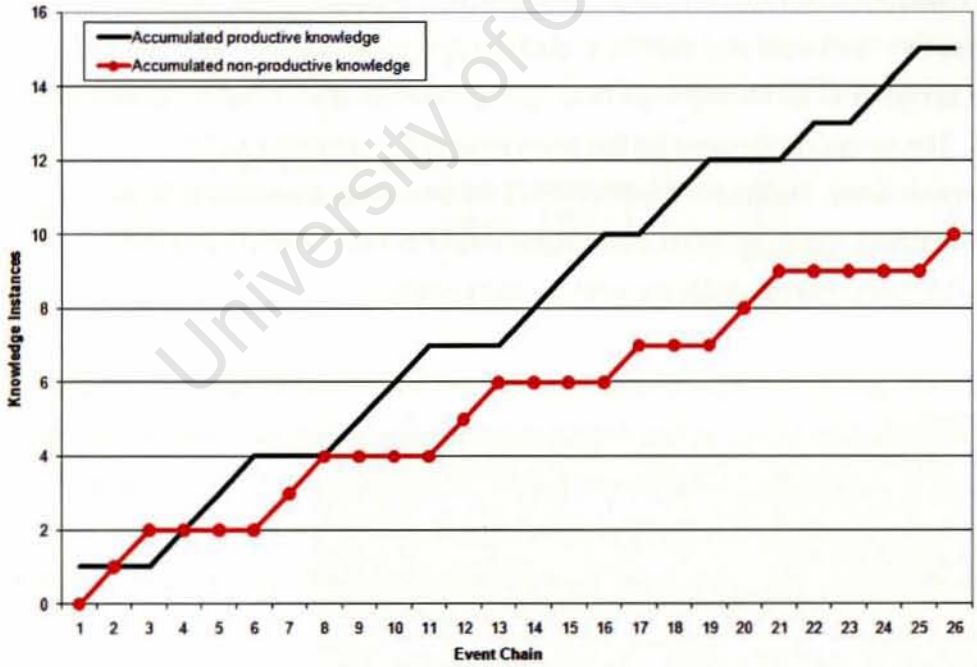


Graph 5.2.12 (b): Process knowledge in P2-12.

University of Cape Town



Graph 5.2.12 (c): Innovation knowledge in P2-12.



Graph 5.2.12 (d): Productive and non-productive knowledge in P2-12.

Table 5.2.12 (a) shows relatively high levels of non-productive knowledge within the different knowledge types. Table 5.2.12 (b) indicates that this team had the lowest

University of Cape Town

occurrence of productive innovation knowledge. The prototype and the demonstration achieved a combined mark of 79%.

Table 5.2.12 (a): Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-12	PK	NPK	Total
Data Knowledge	55%	45%	100%
All Process Knowledge	67%	33%	100%
Role	17%	17%	33%
Logistics	17%	0%	17%
Engineering methods	33%	17%	50%
Innovation Knowledge	50%	50%	100%

Table 5.2.12 (b): Proportions of data, process and innovation knowledge in P2-12.

Knowledge Type: P2-12	PK	NPK	Total
Data Knowledge	24%	20%	44%
Process Knowledge	32%	16%	48%
Innovation	4%	4%	8%
TOTALS	60%	40%	100%

5.2.13 P2-13 Vehicle Usage Tracker (VUT)

The VUT was intended as a DIY kit to be installed in a motor vehicle and to provide supplementary read-outs and statistics, such as fuel consumption, rev counter, average speed, and audio-warnings (e.g., going over the speed limit or excessive revving). The components used by the team included: 7-segment LCD[3] alphanumeric array, Multipoint RS485/RS422 transceivers, a watchdog timer, DS1338 real time clock, an ADXL202 (digital output accelerometer) and the 28F640J3A flash memory (8Mbyte Intel StrataFLASH).

The initial non-productive data knowledge occurrences at event chains 2 – 7 (see Graph 5.2.13 (a)) had to do with the need to research a variety of drivers for the 485 chip, not all of which were used. At event chain 7, the team understood how to access the flash memory from their embedded software. Productive data knowledge occurred, when they found out about RedHat (event chains 15-22) and SimIT (event chain 29).

[3] The team planned to use a bitmapped LCD display, but decided to follow a simpler design using a seven segment alphanumeric LCD array (providing 10 characters).

University of Cape Town

The initial non-productive process knowledge had to do with initial difficulties with the RS485. There were initial process difficulties in investigating the eCos operating system driver for the 485 chip designed for the Agilent AAED-2000. The initial productive process knowledge occurred as the team successfully installed and configured Linux on the CSB337 evaluation board. At event chains 9 – 11 (see Graph 5.2.13 (b)) the team successfully implemented a TCP/IP socket connection, which enabled the communication between the CSB337 and their PC.

No non-productive innovation knowledge occurred in this project (See Graph 5.2.13 (c)). Occurrences of productive innovation knowledge began at event chain 13, after the successful TCP/IP socket connection and the enabling of communication between the CSB337 and the PC. They were trying to install ESAOA tools on RedHat Linux and configure Linux. The team was able to generate vehicle usage statistics, display these on the Motorola JK board's LCD, and download the statistics to a PC over an Ethernet connection.

Graph 5.2.13 (d) indicates the initial convergence at event chains 1 – 8 and thereafter strong divergence.

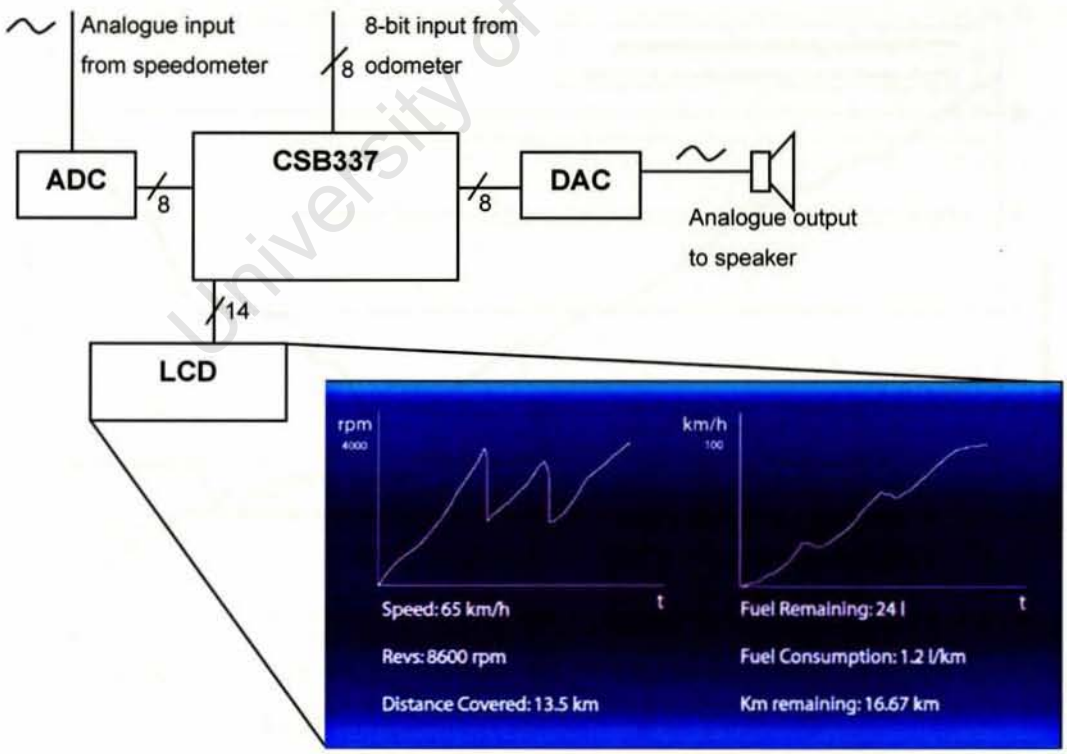
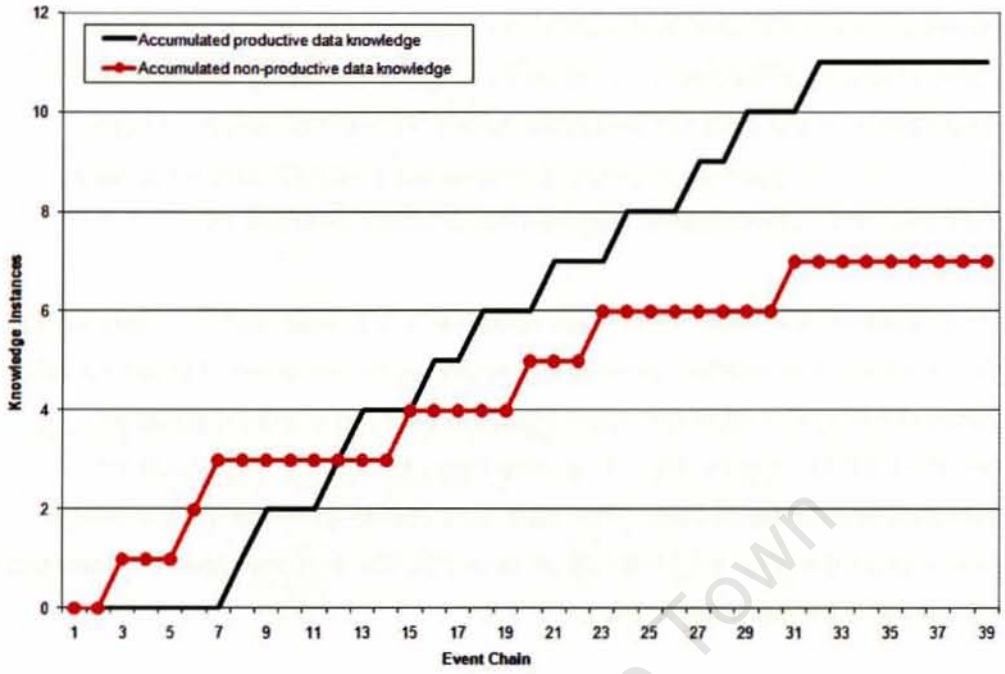
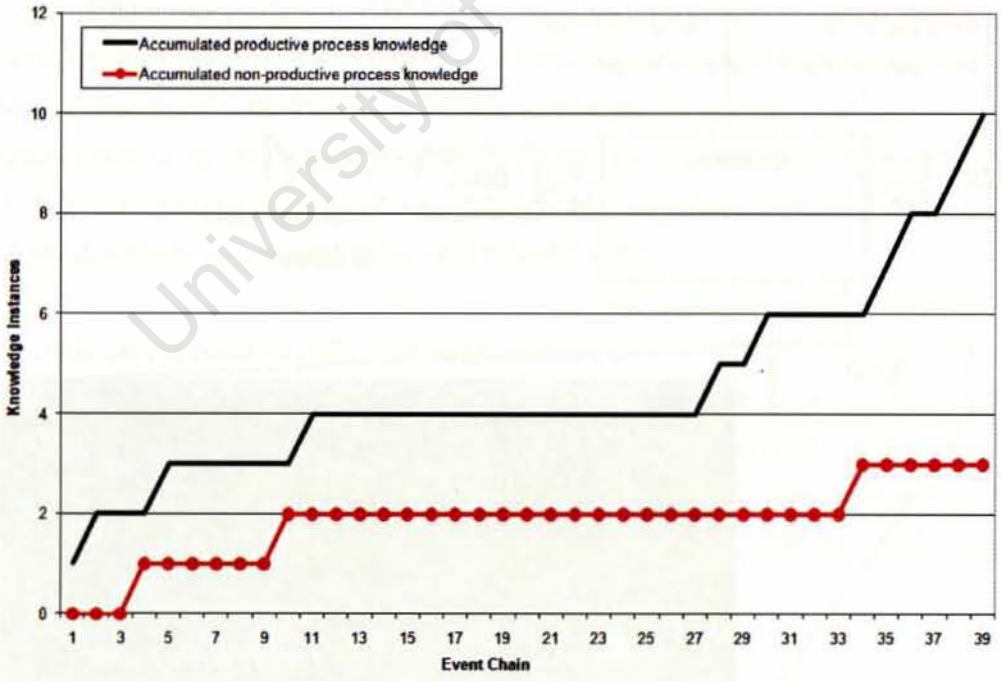


Figure 5.2.13: Component interconnection diagram for Vehicle Usage Tracker (Project P2-13).

University of Cape Town

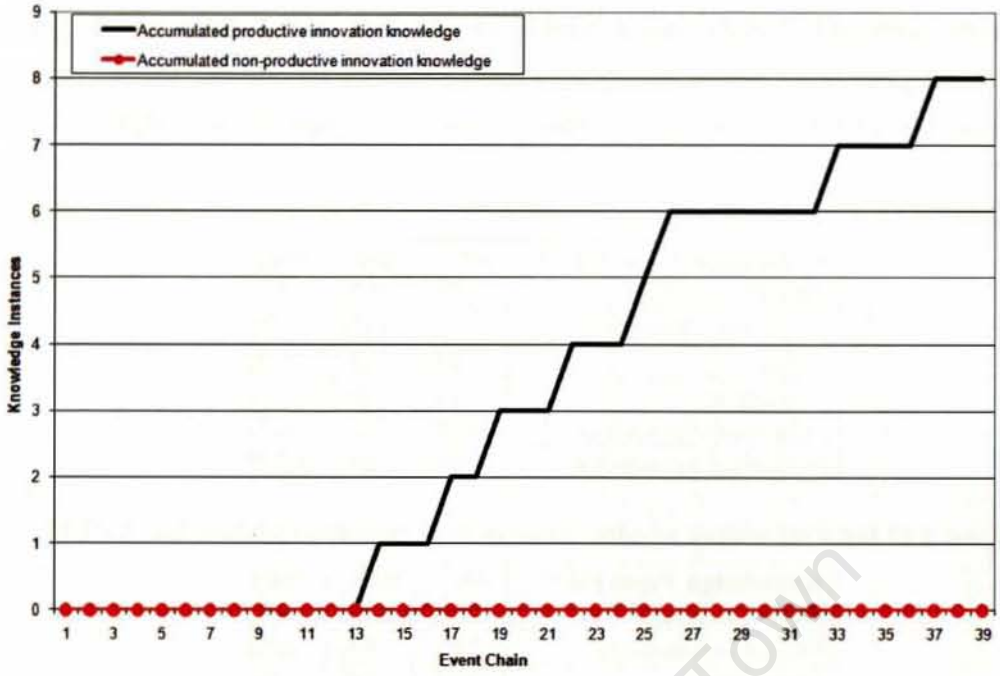


Graph 5.2.13 (a): Data knowledge in P2-13.

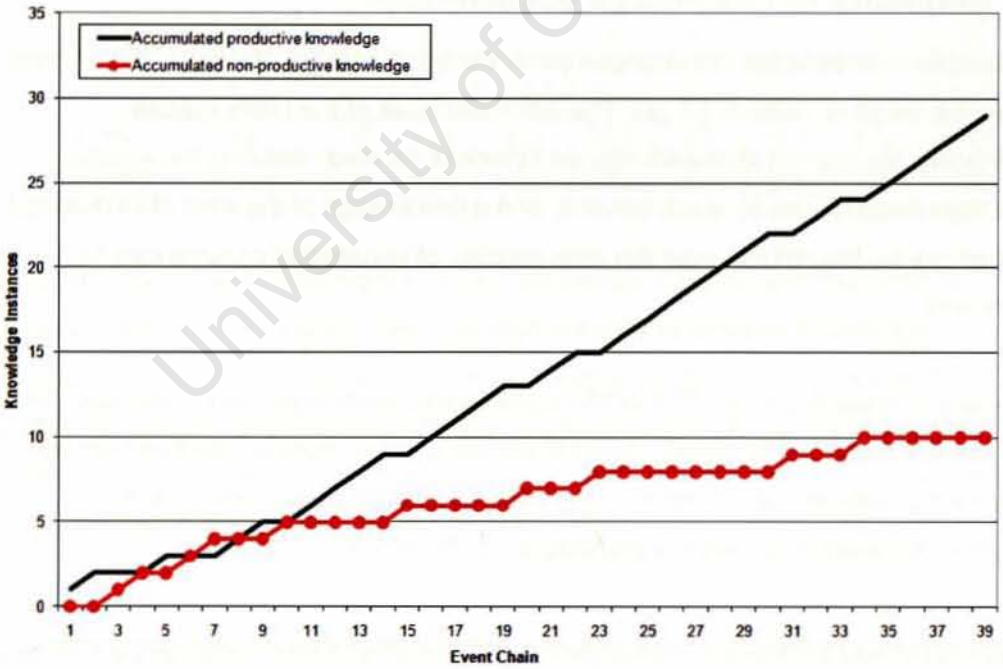


Graph 5.2.13 (b): Process knowledge in P2-13.

University of Cape Town



Graph 5.2.13 (c): Innovation knowledge in P2-13.



Graph 5.2.13 (d): Productive and non-productive knowledge in P2-13.

University of Cape Town

Table 5.2.13 (a) shows the relative high levels of non-productive data knowledge, as previously explained. Occurrences of productive innovation knowledge were slightly below average in P1-13, nevertheless the prototype was judged to be successful by the researcher and the review panel, receiving the third highest score of 95%.

Table 5.2.13 (a): Productive and non-productive knowledge per knowledge type.

Knowledge Type: P2-13	PK	NPK	Total
Data Knowledge	61%	39%	61%
All Process Knowledge	77%	23%	77%
Role	15%	0%	15%
Logistics	0%	0%	0%
Engineering methods	62%	23%	62%
Innovation Knowledge	100%	0%	100%

Table 5.2.13 (b): Proportions of data, process and innovation knowledge in P2-13.

Knowledge Type: P2-13	PK	NPK	Total
Data Knowledge	28%	18%	46%
Process Knowledge	26%	8%	33%
Innovation	21%	0%	21%
TOTALS	75%	26%	100%

5.3 Summary of knowledge occurrences

The number of knowledge occurrences per knowledge category for each Experiment 2 project is listed in Table 5.3.1 (a). The last three rows of the table indicate respectively: the sum of all knowledge occurrences for each column, the average knowledge occurrences for each column, and a percentage of the sum of knowledge occurrences for the column over the total number of knowledge occurrences for the experiment.

As shown in Table 5.3.1 (a), 75% of the occurrences were productive, whereas 25% were non-productive; this clearly shows there were many more productive than non-productive knowledge occurrences. The average ratio for productive to non-productive knowledge across the projects is 75:25 (or 3:1).

The distribution of productive data, process and innovation knowledge occurrences across the projects was in the ratio of 23:32:20 (see last row in Table 5.3.1 (a)). The projects produced 12% more instances of productive process knowledge than innovation knowledge, and 9% more occurrences of productive process knowledge than data knowledge. These ratios may be explained by the knowledge intensive nature of ES product development [Hughes & Cotterell, 2005; Kettunen, 2003;

University of Cape Town

Henninger & Schlabach, 2001], whereby data knowledge, that is, knowledge of components and tools, is needed before ES product development can occur, and development processes are needed before practical innovative ideas can be effectively tested.

Table 5.3.1 (a): Knowledge occurrences for each Experiment 2 project, separated by knowledge category.

Project No.	Total knowledge		Data knowledge		Process knowledge		Innovation knowledge		Total
	Productive	Non-productive	Productive	Non-productive	Productive	Non-productive	Productive	Non-productive	
P2-1	41	17	12	5	15	9	14	3	58
P2-2	75	27	19	17	40	10	16	0	102
P2-3	31	13	10	5	13	5	8	3	44
P2-4	42	4	12	1	14	2	16	1	46
P2-5	30	12	11	5	9	6	10	1	42
P2-6	30	5	5	3	22	2	3	0	35
P2-7	38	14	14	6	11	7	13	1	52
P2-8	39	10	9	3	19	7	11	0	49
P2-9	22	5	9	4	8	1	5	0	27
P2-10	29	10	12	8	9	2	8	0	39
P2-11	25	14	7	9	14	3	4	2	39
P2-12	15	10	6	5	8	4	1	1	25
P2-13	29	10	11	7	10	3	8	0	39
Sum	446	151	137	78	192	61	117	12	597
Averages	34.3	11.6	10.5	6.0	14.8	4.7	9.0	0.9	45.9
Percentage occurrences	75%	25%	23%	13%	32%	10%	20%	2%	100%

Generally, the projects had more process knowledge occurrences than data or innovation knowledge occurrences. The higher levels of process knowledge occurrence could be ascribed to difficulties in applying the data knowledge to the specific tools, components and embedded platform used in the projects [Winberg, 2006e]. Application of engineering methods generally comprises a major part of the effort involved in product prototyping [Henkel, 2006].

5.4 Evaluations of artefacts, prototypes and demonstrations

This section reports on three assessments of the 13 Experiment 2 projects. The first assessment relates to the continuous assessment model implemented by the researcher during code and design reviews. For each code and design review, the processes and progress of each team, as well as the current state of their prototype,

University of Cape Town

were rated (see Section 3.8.1). The other two assessments relate to the demonstrations that took place at the end of the experiment; demonstration check sheets and requirements check sheets were completed by the *review panel* during these assessments (see Section 3.8.6). These check sheets were then used to obtain quantitative ratings for the functionality of the prototype, the quality of the artefacts, and for how effectively the team performed their demonstration. Section 5.4.1 reports on the code and design review assessments performed by the researcher, whereas Section 5.4.2 reports on the evaluations performed using demonstration check sheets and requirements check sheets.

5.4.1 Evaluations of code and design reviews

The researcher evaluated each of the projects at various stages during code and design reviews. Three evaluations were performed in total, one in each code and design review, as described in Section 3.8.1. The reviews intentionally correspond to phases of the ES lifecycle model given in Section 2.2, whereby Review 1 partly relates to the requirements phase (as well as judging creativity and understanding of planned prototypes); Review 2 relates to the specification and design phase (for which ES design documents and enclosure models are rated); and Review 3 relates to the implementation and integration phases (for which type of artefacts, quality of artefacts, and relations between artefacts are checked).

5.4.1.1 Results from evaluation forms

Tables 5.4.1 (a), (b) and (c) respectively present the breakdown of ratings for Review 1, Review 2 and Review 3. The last two columns show averages (Avg) and standard deviations (Std) across the projects. Sample evaluation forms filled in by the researcher for these reviews are given in Appendix B.4 (the criteria are elaborated on in the appendix). The last row of each table is the sum of the criteria ratings, which gives an overall score for each project that indicates how well the team did for the code and design reviews.

The total ratings for evaluations performed for the code and design review are shown in Table 5.4.1 (d). The last column of the table gives average ratings for each project. All ratings are sorted from the highest value (at the top) to the lowest value (the second last row).

University of Cape Town

Table 5.4.1 (a): Breakdown of creativity ratings per project.

Criteria	Max	P2-1	P2-2	P2-3	P2-4	P2-5	P2-6	P2-7	P2-8	P2-9	P2-10	P2-11	P2-12	P2-13	Avg	Std
Not commonplace	15	12	14	13	14	12	14	15	13	11	14	14	13	12	13.2	1.14
Context	5	4	5	4	2	3	5	5	3	2	2	5	4	4	3.69	1.18
Playing with ideas	10	9	4	7	9	8	8	9	4	9	8	9	8	10	7.85	1.86
Usefulness	10	8	8	8	8	7	9	9	7	9	9	8	8	8	8.15	0.69
Challenge	5	4	5	5	4	5	5	5	4	3	5	5	4	5	4.54	0.66
Sophistication of techniques/parts	10	8	6	7	7	6	9	9	8	7	8	7	7	8	7.46	0.97
Interfacing & connections	5	5	3	5	4	4	4	4	4	5	4	5	4	5	4.31	0.63
Diversity of parts	10	7	6	7	7	7	8	7	6	9	8	8	8	9	7.46	0.97
Enclosure	10	6	5	9	7	8	10	7	5	7	7	8	6	6	7.00	1.47
Cost-saving ideas	10	5	5	8	8	8	9	9	5	9	8	8	8	7	7.46	1.51
Drawings & general clarity	10	7	6	10	8	9	9	9	6	8	7	9	7	8	7.92	1.26
TOTAL	100	75	67	83	78	77	90	88	65	79	80	86	77	82	79	7.31

Table 5.4.1 (b): Breakdown of design ratings per project.

Criteria	Max	P2-1	P2-2	P2-3	P2-4	P2-5	P2-6	P2-7	P2-8	P2-9	P2-10	P2-11	P2-12	P2-13	Avg	Std
Introduction	5	5	5	5	4	5	5	5	5	4	5	5	4	5	4.77	0.44
Block diagram	10	9	10	9	9	10	9	10	10	9	9	9	7	10	9.23	0.83
Use cases	5	5	5	5	4	5	5	5	5	5	5	5	3	5	4.77	0.60
Scenarios	5	5	5	4	4	5	5	5	4	4	4	5	4	5	4.54	0.52
UML class & object design	10	9	10	9	8	10	8	8	7	6	8	9	7	9	8.31	1.18
Schematic/circuit	10	8	10	7	10	10	10	8	9	6	8	9	7	10	8.62	1.39
Component list	5	5	5	4	5	5	5	4	5	4	5	5	5	5	4.77	0.44
Enclosure design	5	5	5	5	5	5	4	5	5	4	3	5	2	4	4.38	0.96
No unwanted redundancy	5	5	5	5	5	5	5	5	5	4	5	5	5	5	4.92	0.28
Accuracy	10	9	10	8	10	8	10	9	9	8	8	9	7	10	8.85	0.99
Completeness	5	5	5	5	5	5	5	5	4	4	5	4	4	5	4.69	0.48
General: clarify	10	9	9	10	9	10	8	9	8	8	8	8	7	10	8.69	0.95
General: internal consistency	5	5	5	5	5	5	5	4	4	5	5	4	4	5	4.69	0.48
General: external consistency.	5	5	5	5	4	5	5	5	3	4	5	4	4	5	4.54	0.66
General: cross-ref / traceability	5	5	4	5	4	5	5	5	3	5	5	5	5	5	4.69	0.63
TOTAL	100	94	98	91	91	98	94	92	86	80	88	91	75	98	91	0.72

University of Cape Town

Table 5.4.1 (c): Breakdown of artefact ratings per project.

Criteria	Max	P2-1	P2-2	P2-3	P2-4	P2-5	P2-6	P2-7	P2-8	P2-9	P2-10	P2-11	P2-12	P2-13	Avg	Std
Naming artefacts	10	10	6	6	10	10	9	8	8	9	10	8	7	10	8.54	1.51
Organisation	10	10	6	8	10	10	9	8	7	8	8	7	7	10	8.31	1.38
Traceability	10	10	7	7	10	10	10	10	8	8	8	8	7	10	8.69	1.32
Modifiability	10	10	7	10	10	10	10	10	8	10	10	9	8	10	9.38	1.04
Reusability	5	5	3	4	5	5	5	5	4	5	5	5	5	4	4.62	0.65
General readability	10	10	7	7	10	9	9	8	10	9	7	7	8	9	8.46	1.20
Readability of code	5	5	3	3	5	5	5	4	5	5	4	4	5	4	4.38	0.77
Relating artefacts	5	5	4	5	5	5	5	4	4	4	4	5	4	5	4.54	0.52
Accessibility/ease of location	10	10	6	9	10	10	10	10	7	10	10	8	7	10	9	1.47
Owner/author, change history	5	5	3	3	4	3	3	4	3	5	5	3	5	4	3.85	0.90
Non-redundancy	5	5	3	4	5	5	5	5	5	4	5	4	5	5	4.62	0.65
Cross-referencing	5	5	4	4	5	5	5	5	4	5	5	5	4	5	4.69	0.48
Use of ESAOA tools & other IT tools/scripts	10	10	6	8	10	9	9	9	7	8	9	8	7	9	8.38	1.19
TOTAL	100	100	65	78	99	96	94	90	80	90	90	81	79	95	87.5	1.01

Table 5.4.1 (d): Totals of the code and design review ratings per project.

Rating 1: Creativity		Review 2: Design		Review 3: Artefacts		Average (%)	
Project	Rating	Project	Rating	Project	Rating	Project	Rating
P2-6	90	P2-2	98	P2-1	100	P2-6	93
P2-7	88	P2-5	98	P2-4	99	P2-13	92
P2-11	86	P2-13	98	P2-5	96	P2-5	90
P2-3	83	P2-1	94	P2-13	95	P2-7	90
P2-13	82	P2-6	94	P2-6	94	P2-1	90
P2-10	80	P2-3	92	P2-7	90	P2-4	89
P2-9	79	P2-7	92	P2-9	90	P2-10	86
P2-4	78	P2-4	91	P2-10	90	P2-11	86
P2-5	77	P2-11	91	P2-11	81	P2-3	84
P2-12	77	P2-10	88	P2-8	80	P2-9	83
P2-1	75	P2-8	86	P2-12	79	P2-8	77
P2-2	67	P2-9	80	P2-3	78	P2-12	77
P2-8	65	P2-12	75	P2-2	65	P2-2	77
Avg.	79	Avg.	91	Avg:	87	Avg:	86

Average score: 86% (standard deviation 5.73 between averaged ratings)

According to Table 5.4.1 (d), the Project P2-6 team performed best on average (having an average of 93% for all three reviews evaluations). During Review 1, the Project P2-6 team was also judged to have the most creative concept. Project P2-6

University of Cape Town

was only a few percentage points below the other projects for the second and third review, confirming its place as the top rated project on average.

Projects P2-2, P2-5 and P2-13 had the highest design ratings, each project receiving 98% for Review 2. For Review 3, Project P2-1 was found to have the best quality artefacts, albeit only slightly better (one percentage point) than those of Project P2-4.

Project P2-2, P2-8 and P2-12 received the lowest ratings on average; nonetheless, this averaged rating was 77% and, considering that all these teams achieved at least 65% for any one review, it does indicate that they performed adequately overall.

Across all the reviews, the ratings were generally high, i.e. an average of 86% all together. The standard deviation was fairly low at 5.73%, indicating that there was *less* than 10% difference between the ratings on average.

5.4.1.2 Comments from knowledge production questions

During design review 2, the teams were asked to respond orally to questions related to their knowledge production methods and information sources (see Section 3.8.1). The researcher noted responses during the design reviews. After the final design review the researcher reviewed all these notes and compiled a list of the most commonly recurring methods; this list is given in Table 5.4.1(e) and has been arranged according to the question that respectively gained insights in data, process and innovation knowledge production methods. The responses for each question are ordered accordingly to how often they reoccurred by different teams, or which were given the most emphasis (most common and strongly emphasised response first).

The most common methods employed for acquiring data knowledge, as listed in Table 5.4.1(e) included web searches and use of PDFs datasheets, user manuals and other documentation that was either downloaded from the internet or provided in the ESAOA communal workspace. Frequently used methods for obtaining process knowledge included tasks where developers were determining the correct sequence of hardware register assignments and procedure calls, compiling code, adapting code, editing makefiles, and testing downloaded sample code. In terms of innovation knowledge production, there was recurrent mention of getting the right sequence of steps, adapting techniques used earlier to get new features to work, discarding prior solutions, and difficulty in discovering the correct sequence of procedures to use.

University of Cape Town

Table 5.4.1 (e): Commonly reported knowledge production methods.

Question	Common responses
Q1: Methods used during data knowledge production and sources of information	Web searches, used PDFs in ESAOA communal workspace, shared downloads folder, saved web pages, datasheets, manuals, downloaded PDF documents, rename/relocate files, manufacturer websites (ATMEL, Phillips semiconductors, Microchip), added comments, search forums (e.g., what components other people used), search in PDFs, Wikipedia, retailer websites (e.g., Sparkfun, Avnet), searching pricelists (for finding low-cost component options), add metadata using fclass, use find in downloaded sample code files and manuals, made notes on product numbers, looking at table of contents in datasheets, used esaoa-find, highlighted sections of docs, C language guide, browser bookmarks, bookmarks in printouts.
Q2: Methods used during process knowledge production	Determining correct sequence of hardware register assignments and procedure calls, change/rewrite/add code (esp. init.c, start and main function, config.make and other makefiles), compiling code, changing values sent to registers, searching though code, adding comments to record settings/parameter options, correcting/finding addresses to use, adding new #defines and constant values, finding sample code and drivers on the web that works, noting down important steps (in logbook / textfile), trying tutorials, changing downloaded drivers, wiring-up and soldering hardware, searching forums, checking pinouts, assigning GPIO pins, testing and debugging downloaded code, programming flash boot memory, configuring boot loader, checking components work (as planned), used multimeter to test voltage levels and check connections, posting to group forum, using oscilloscope to capture signals and do debugging, created scripts to record settings / sequence of steps, downloading template files, emailing code to teammate, esaoa-find, testing different tool parameters, made scripts, installing drivers, editing configuration files, reconfiguring driver options, changing sequence drivers are loaded in uCLinux, printing out and checking code, looking through and testing menus (in KDevelop), comparing to backups, posting to external forums, looking-up in books (e.g., Linux handbooks), C programming online guides and books, shared textfile of commands/tool params.
Q3: Methods used during innovation knowledge production	Usually needed new sequence of steps to get new feature working, reused/adapted things done earlier, often couldn't find steps that worked, had to do various tests – more than one, tried different techniques for same requirement, design ideas were easy but implementing them was hard, modified/reused methods/code for previous design issues, discarded many of our solutions, though of most solution in the lab, various drawings (e.g., 'concept cartoons') and point-form notes to explain/record ideas and solutions, Q3 usually came first and Q1 and Q2 after, thought of ways to implement design choices while implementing other design aspects, though of many ideas/solutions outside the lab, though of solutions/ideas when looking at websites while solving other problems, proposed and minuted ideas at meetings, group email to CC all members, ideas occurred while making concept poster, writing code more difficult than thinking of design ideas, difficult to find what (method?) was wanted, waste of time looking for ready-made solutions online, shared methods while sitting together in lab – though it pointless and too time-consuming to write down steps ("once we know the method, we knew the method"), expected input/output files, emailed suggestions, noted parts of solutions/ideas in logbook, used chat/instant messaging while working simultaneously on different parts, came up with our own approaches because ones online didn't worked for us, printed diagrams/code and added to them in meetings – later updated files, milestones helped keep us focus and finished on time.

University of Bath

5.4.1.3 Notes from design review 3

Design review 3 involved rating the quality of artefacts produced by teams, in addition to investigating to what extent ESAOA support tools (i.e., ESAOA scripts or programs) were modified, or added to, by the teams (see Section 3.8.1). Results from design review 3 evaluation forms were presented in Section 5.4.1.1.

Many of the teams added to the collection of ESAOA support tools that were provided in the baseline communal and team workspaces. All these new scripts were in the form of Bash scripts, and were evidently (based on the comments at the top of the scripts) crafted from the script template provided in the ESAOA communal workspace. None of the teams used the KIT API to create their own C-based support tools. Project teams P2-1, P2-4 and P2-6 added the most number of scripts, an average of six scripts each. Project team P2-2 made the least use of scripts, only adding one script (which in actuality appeared not to be an executable script, but rather a text file that listed commands and their arguments that the team found useful or used often – no comments were included in the file). The second last row in Table 5.4.1(c) in Section 5.4.1.1, which relates to the extent to which teams used ESAOA tools, also closely corresponds to the extent to which teams made changes to ESAOA scripts. In general, teams (except Projects P2-1 and P2-4) that added new scripts provided little or no comments to explain their purpose.

5.4.2 Review panel's evaluations

As mentioned in Section 3.8.6, the review panel watched and evaluated demonstrations performed by the project teams towards the end of Experiment 2. After these demonstrations had taken place, two different check sheets were completed by the review panel for each project. The demonstration check sheet was used primarily to rate the quality of the team during the demonstration; whereas the requirements check sheet was used to rate the success of a team's prototype and the quality of their project artefacts. These evaluations were performed in order to obtain quantitative ratings of a team's performance, and of the team's prototype, in order to relate these performance measures of the end results to knowledge production and the use of the ESAOA KMS.

The following two subsections present the ratings, first for the demonstration check sheet, and then for the requirements check sheet. Section 5.5.1 compares these two, in addition to discussing relations between these scores and the knowledge production results.

University of Cape Town

5.4.2.1 Demonstration check sheet results

The ratings for each criterion of the demonstration check sheet, for each of the 13 Experiment 2 projects is shown in Table 5.4.2 (a). The last column of the table shows each team's total score; this is labelled 'functionality of team', as the criterion as a whole relates to how well the *team* functioned during the presentation (as can be seen in the table, the functionality of the prototype is not intended to be evaluated using this form; the requirements check sheet is used for that purpose). The bottom row of the table gives averages for each column; for example, teams on average scored 9/10 for their level of readiness.

Table 5.4.2 (a) shows that Project P2-1 (the Location-aware Tourist Information System) was given the highest score (a final score of 100%) by the review panel. This high score can likely be attributed to the fact that the team gave an excellent demonstration, although their concept was not the most creative (see Section 5.4.1).

Table 5.4.2 (a): Demonstration check sheet scores for each project.

Project No.	1) Preparedness				2) Summary of project			3) Set-up of components		4) Experimentation			5) Code	6) General			Functionality of team (% Total)
	Readiness	Order	Flow	Time keeping	Clarity	Explanation	Description	Explanation	Description	Method	Results	Analysis	Explanation	Recommendations	Questions	Language	
P2-1	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	100
P2-2	0	7	5	7	8	7	8	5	6	8	6	10	4	8	7	8	65
P2-3	10	7	6	10	7	7	8	8	8	6	6	8	8	7	9	10	78
P2-4	10	10	9	10	10	10	10	10	10	10	8	10	10	10	10	10	98
P2-5	10	8	9	10	9	9	10	10	7	8	10	8	8	9	9	10	90
P2-6	10	10	9	10	8	8	10	10	8	8	10	10	9	8	10	10	93
P2-7	10	10	8	10	8	8	8	8	8	9	9	10	10	10	8	10	90
P2-8	10	10	8	10	6	6	8	8	7	5	5	8	8	8	8	10	78
P2-9	10	9	8	10	5	6	10	8	8	7	8	6	8	8	7	10	80
P2-10	10	9	9	10	10	9	10	10	8	8	8	10	8	8	8	10	91
P2-11	10	8	8	10	9	7	9	8	8	5	8	8	8	8	7	10	82
P2-12	10	4	6	10	6	8	9	7	8	4	8	6	8	8	8	10	75
P2-13	10	10	8	9	10	10	10	9	8	8	10	8	10	10	8	10	93
Avg.	9	9	8	10	8	8	9	9	8	7	8	9	8	9	8	10	86

The Project P2-4 team came a close second with a total score of 98%. The third-highest score was a tie between Projects P2-6 and P2-13, both of which were given a demonstration rating of 93%. The project with the lowest score was Project P2-2, with 65% (which was 10% below the second lowest score).

University of Cape Town

5.4.2.2 Requirements check sheet results

The requirements check sheets focus on evaluating the functionality and quality of the teams' prototypes and project artefacts. Appendix B.2 provides the complete set of criteria ratings decided by the review panel for the requirements check sheets of each project of Experiment 2. Appendix B.3 lists the comments that the review panel added to the check sheets for each of the projects. As shown in Section 3.8.6, the criteria for the requirements check were grouped into four sections, with each section given a weighting based on its importance. The sections were: 1) functional requirements, 2) temporal requirements, 3) quality of artefacts, and 4) quality of the prototype enclosure. The total scores for each of these sections, and the weighting of each section, are shown for all the projects in Table 5.4.2 (b). The last row gives the weighted total requirements score for each project.

The average ratings for each section, across all projects, were the following: the average score for functional requirements was 74% (with a maximum of 90% for Projects P2-1 and P2-5); the average score for temporal requirements was 72% (with Project P2-10 receiving the highest score of 100%); the average score for the quality of artefacts was 77% (with Project P2-1 receiving the maximum of 96%); and the average prototype enclosure score was 59% (with Project P2-13 receiving the highest enclosure score of 100%).

University of Cape Town

Table 5.4.2 (b): Section scores for each requirements check sheet.

Section	Section weight	Project No.												
		P2-1	P2-2	P2-3	P2-4	P2-5	P2-6	P2-7	P2-8	P2-9	P2-10	P2-11	P2-12	P2-13
1. Functional requirements	44%	90%	77%	75%	78%	90%	81%	65%	51%	64%	70%	74%	83%	66%
2. Temporal requirements	11%	97%	63%	70%	63%	97%	37%	53%	67%	67%	100%	73%	57%	87%
3. Quality of artefacts	40%	96%	94%	89%	80%	94%	85%	73%	73%	55%	62%	57%	66%	72%
4. Quality of enclosure	5%	33%	80%	87%	33%	73%	73%	20%	33%	67%	100%	53%	20%	100%
Weighted total:		90%	82%	81%	75%	91%	77%	64%	60%	61%	72%	66%	70%	72%

Project P2-5 (the Home Automation System) had the highest total score for the requirements check sheet, viz. a total weighted score of 91%. This project also received the highest score for functional requirements (a score of 90% shared with Project P2-1), and it furthermore received one of the highest ratings for temporal requirements and for the quality of the final project artefacts. The enclosure rating for the prototype produced in the project received only an above-average rating.

Projects P2-8 had the lowest overall requirements score, with Projects P2-7 and P2-9 only slightly ahead. These projects consistently received below average marks for each of the four rating sections.

5.5 Comparisons

This section uses correlations between knowledge production statistics (the tabulated results in Sections 5.3) and the results of project assessments (i.e., results shown in Sections 5.4.1 and 5.4.2). The subsections below summarise the results of the correlations, including discussion of the strongest relations and of how these appear to influence one another. The correlation method is described in Section 3.12.

5.5.1 Comparing requirements and demonstration check sheets scores

Table 5.4.3 (a) lists the final scores for the demonstration and requirements check sheets per project. The last column indicates the absolute differences between the scores for each project. The correlation coefficient appears at the bottom of the table.

Table 5.4.3 (a): Demonstration check sheet scores compared to requirements check sheet scores.

Project No.	% Demonstration check sheet score	% Requirements check sheet	Abs. Difference
P2-1	100	90	10
P2-2	65	82	17
P2-3	78	81	3
P2-4	98	75	23
P2-5	90	91	1
P2-6	93	77	16
P2-7	90	64	26
P2-8	78	60	18
P2-9	80	61	19
P2-10	91	72	19
P2-11	82	66	16
P2-12	75	70	5
P2-13	93	72	21

Correlation: $r = 0.246$

The correlation coefficient indicates a weak relation between the sets, indicating that, even if a project satisfied most requirement checks, it was no guarantee that it would achieve a correspondingly high demonstration rating. Since the correlation coefficient is positive, though, higher requirements scores do tend towards higher demonstration results. Projects P2-7 and P2-4 had the highest difference between the demonstration and requirements scores; in both cases the teams scored much

University of Cape Town

higher for the demonstration than they did for the requirements check. This was largely attributable to Project P2-4 team's particularly professional presentation, and to Project P2-7 team's particularly gripping product concept (which received the second highest creativity score in Table 5.4.1.(d)) – in both cases, the review panel was less critical of limitations of the prototypes due to the teams' good impressions.

5.5.2 Comparing design reviews ratings to check sheet scores

In this section, comparisons are made between the averaged design review ratings (shown in Table 5.4.1 (d)) and the scores awarded by the review panel for the demonstration and requirements check sheets (see Tables 5.4.2 (a) and 5.4.2 (b)). Table 5.5.2 (a) lists the scores to be compared; the last column gives an average of the demonstration score and the requirements score. Table 5.5.2 (b) shows the correlation coefficients for comparing each column of scores to the other columns. Comparisons between the check sheets were already discussed in Section 5.5.1.

Table 5.5.2 (a): Design review averages compared to review panel scores.

Project No.	% Design review average score	% Demonstration check sheet	% Requirements check sheet	% Average of check sheet scores
P2-1	90	100	90	95
P2-2	77	65	82	74
P2-3	85	78	81	80
P2-4	89	98	75	87
P2-5	90	90	91	91
P2-6	93	93	77	85
P2-7	90	90	64	77
P2-8	77	78	60	69
P2-9	83	80	61	71
P2-10	86	91	72	82
P2-11	86	82	66	74
P2-12	77	75	70	73
P2-13	92	93	72	83

Table 5.5.2 (b): Correlation results.

	% Design review average score	% Demonstration check sheet	% Requirements check sheet	% Average of check sheet scores
% Design review average score		0.860	0.325	0.745
% Demonstration check sheet	0.860		0.246	0.783
% Requirements check sheet	0.325	0.246		0.795
% Average of check sheet scores	0.745	0.783	0.795	

The highest correlation coefficient *r* in Table 5.5.2 (b) is 0.860 (where r2 = 74%). This indicates a strong relation in which projects that scored highly for their design reviews

University of Cape Town

also did well for their demonstration. This finding is corroborated by the literature, which indicates that design reviews improve the success of development products [Schach, 2005].

5.5.3 Comparing design reviews to knowledge production statistics

In this section the code and design review scores are compared to the numbers of knowledge occurrences per project (Section 5.5.3.1), while code and design review scores are compared to proportions of knowledge productions (Section 5.5.3.2).

5.5.3.1 Comparing code and design reviews to productive knowledge occurrences

The code and design review ratings (Table 5.4.1 (d)) were correlated with the number of knowledge occurrences for each project, separated into categories of data, process and innovation knowledge. The resulting correlations coefficients are given in Table 5.5.3 (a). The highest correlations are shown in bold.

Table 5.5.3 (a): Correlations between code and design review scores and productive knowledge occurrences across all projects.

Design Review	Productive data knowledge	Productive process knowledge	Productive innovation knowledge
Concept rating	-0.416	-0.425	-0.460
Design rating	0.493	0.414	**0.526**
Artefacts rating	-0.218	**-0.569**	0.067
Average rating	-0.362	**-0.614**	-0.178

The strongest correlation in Table 5.5.3 (a) is -0.614, between average review ratings and productive process knowledge. This yields a coefficient of determination (*r* squared) of 38%. This result suggests the proportion of productive process knowledge is a potential indication of how well design reviews will be rated. This linear correlation is a fairly weak relation, though, and it is a negative (i.e. an inverse correlation). This correlation was not expected to be negative; it was expected that higher proportions of process knowledge, especially productive process knowledge, would lead towards a more successful project. However, these results appear to imply that the reverse is more correct, specifically that *higher* levels of process knowledge may lead to *lower* review ratings. This may appear counter-intuitive at first. However, there are likely causes for this observation. For example, spending too much time perfecting development methods may block progress instead of speeding it up. In this regard, Project P2-2 experienced a more profound problem due to a lack of coordination, in that different team members independently developed their own

University of Cape Town

effective practices (which counted towards productive knowledge), but this duplication wasted time, which ultimately led to the reviews indicating poor performance. Other possible reasons for this inverse correlation include the following: Firstly, the occurrence of 'dead end' data knowledge does not negatively affect the quality of the final product (as long as appropriate productive data knowledge is eventually found). Secondly, the fact that the team engaged in non-productive processes does not necessarily influence the final artefact produced (as long as the team did eventually use appropriate methods). Thirdly, there is also a potential link between the quality of the ES artefacts and the occurrence of productive innovation knowledge, as suggested by the third strongest correlation discussed below.

The second strongest correlation, -0.569 (r^2 of 32%), is between artefact ratings and productive process knowledge. This negative correlation implies that projects with high quality artefacts produced smaller portions of process knowledge; or conversely, most projects that produce larger portions of process knowledge had artefacts of poorer quality. In this experiment, though, the former is more valid: projects with a larger portion of process knowledge (namely P2-8 and P2-2) spend less effort on the quality of their artefacts (supported by the artefact scores in Table 5.4.1 (d)). It is important to note that the reviewers did not assess the maintainability of the prototype – the developers were aware that the success of their prototype would be primarily based on its functionality and usability. Consequently, some teams (particularly Projects P2-3 and P2-12) sacrificed the quality of their artefacts for the improved functionality of the prototype – for Experiment 2 projects, this trade-off was generally beneficial and led to higher scores (provided the prototype was sufficiently functional). Project P2-2 members furthermore have sacrificed artefact quality to achieve a working prototype, but were also rated as less successful because they produced a poorly functioning prototype and gave a weak demonstration.

The third strongest correlation was positive, the value 0.526 (r^2 of 28%), between the proportion of innovation knowledge and the design rating. This is a fairly weak relation. It potentially indicates that a *higher* proportion of innovation knowledge leads to a *higher* design rating. This appears to correspond to the way in which good designs are identified, for which the difference between mediocre and excellent products is often a matter of innovative ideas [Kopetz, 1997; Goossens *et al.*, 1997].

Project P2-1 was rated the highest by the review panel, and also had the top score for the quality of its artefacts. This indicates that a project team still had to have good

University of Cape Town

underlying design artefacts in order to achieve exceptional results from the review panel. It is likely that the quality of the artefacts would have become even more important if maintenance issues had been considered too (as is recommended by many authors, such as Stamelos *et al.* [2002]).

Other correlations indicate that the proportion of innovation knowledge has little or no impact on either quality of artefacts or the overall success of Experiment 2 projects.

5.5.3.2 Comparing code and design reviews to knowledge occurrences

Correlations between the code and design review ratings and the total amount of knowledge occurrences per knowledge category are shown in Table 5.5.3 (b) (note that the previous section focused only on the *proportions* of productive knowledge). The last row shows correlations between the knowledge types and averages for all review ratings. The four highest correlations are in bold text.

Table 5.5.3 (b): Correlations between review scores and knowledge occurrences.

Design Review	Total data knowledge	Total process knowledge	Total innovation knowledge
Concept rating	-0.369	-0.500	-0.400
Design rating	0.460	0.443	**0.525**
Artefacts rating	-0.497	**-0.569**	0.081
Average rating	**-0.537**	**-0.650**	-0.178

The strongest correlation in Table 5.3.1 (b) is -0.650 (r^2 of 42%), between total process knowledge and the average design review ratings. The previous section similarly showed the strongest correlation between productive process knowledge and design ratings. However, as for the case in the previous section, this correlation is also a negative one, thus implying that, for Experiment 2, higher proportions of productive knowledge led to lower design ratings.

The second strongest correlation is a negative one of -0.569 (r^2 of 32%), between artefact ratings and proportion of process knowledge. As per the previous section, projects with better artefacts seem to have lower proportions of process knowledge, regardless of whether that process knowledge is productive or non-productive.

The third strongest correlation, -0.537 (r^2 of 29%), is between the average of all ratings and the proportion of data knowledge produced for a project. This is an inverse correlation, indicating that projects, which on average achieve a high

University of Cape Town

progress score, may be accumulating a smaller proportion of data knowledge or, conversely, that projects with higher proportions of data knowledge are likely to achieve lower progress ratings. For this experiment, the latter is more accurate: teams (e.g., Projects P2-2 and P2-10) that accumulated higher proportions of data knowledge tended to have lower progress review scores.

A positive correlation of 0.525 (r^2 of 28%) exists between design ratings and the proportion of innovation knowledge for the projects. This indicates a possibility that projects with higher proportions of innovation knowledge get higher design scores.

5.5.4 Comparisons with productive innovation knowledge

The positive correlation between innovation knowledge and design scores, observed in Section 5.5.3.2, is further investigated in this section by comparing proportions of productive innovation knowledge to check sheet scores.

5.5.4.1 Comparing productive innovation knowledge and averaged scores for check sheets

The averaged check sheet scores (which were given earlier in Table 5.5.2 (a)) are correlated with the percentage proportions of productive innovation knowledge for each project in Table 5.5.4 (a).

Table 5.5.4 (a): Percentage of productive innovation knowledge occurrences compared to average of demonstration and requirements check sheet scores.

Project No.	% Productive innovation occurrences	% Average of check sheet scores
P2-1	24	95
P2-2	16	74
P2-3	18	80
P2-4	35	87
P2-5	24	91
P2-6	9	85
P2-7	25	77
P2-8	22	69
P2-9	19	71
P2-10	21	82
P2-11	10	74
P2-12	4	73
P2-13	21	83

Correlation: $r = 0.426$

As Table 5.5.4 (a) shows, the correlation coefficient for the two sets is 0.426 (r^2 of 18%), indicating a fairly weak positive relation. This potentially indicates the overall success of a project, in terms of satisfying its requirements and the team delivering a

University of Cape Town

successful demonstration, and it was higher for teams that produced a higher proportion of innovation knowledge.

5.5.4.2 Comparing productive innovation knowledge to demonstration check sheet scores

The proportions of productive innovation occurrences and the demonstration check sheet scores for each project are listed together in Table 5.5.4 (b).

Table 5.5.4 (b): Percentage productive innovation knowledge occurrences compared to demonstration check sheet scores.

Project No.	% Productive innovation occurrences	% Demonstration check sheet score
P2-1	24	100
P2-2	16	65
P2-3	18	78
P2-4	35	98
P2-5	24	90
P2-6	9	93
P2-7	25	90
P2-8	22	78
P2-9	19	80
P2-10	21	91
P2-11	10	82
P2-12	4	75
P2-13	21	93

Correlation: $r = 0.520$

As shown in Table 5.5.4 (b), teams who scored 80% and above tended to have a high percentage of innovation knowledge occurrences, whereas teams who scored below 70% tended to have low occurrences of productive innovation knowledge. The correlation coefficient between the sets is 0.520 (r^2 of 27%). This indicates a potential trend for the projects concerned (which is slightly stronger than the one found in Table 5.5.4 (a)) between occurrences of productive innovation knowledge and the success of the prototype demonstrations.

5.5.4.3 Comparing productive innovation knowledge to requirements check sheet scores

The proportions of productive innovation occurrences and the requirement check sheet scores for each project are listed together in Table 5.5.4 (c). The correlation coefficient of 0.146 between the sets indicates that there is no relation. If a team obtained larger proportions of innovation knowledge, it appears to give no indication as to whether or not the requirements will be well satisfied.

University of Cape Town

Table 5.5.4 (b): Percentage productive innovation knowledge occurrences compared to requirements check sheet scores.

Project No.	% Productive innovation occurrences	% Requirements check sheet score
P2-1	24	90
P2-2	16	82
P2-3	18	81
P2-4	35	75
P2-5	24	91
P2-6	9	77
P2-7	25	64
P2-8	22	60
P2-9	19	61
P2-10	21	72
P2-11	10	66
P2-12	4	70
P2-13	21	72

Correlation: $r = 0.146$

5.5.5 Comparing check sheet scores and knowledge occurrences

This section investigates correlations between knowledge occurrences for each of the projects, as well as correlations between check sheet scores and knowledge occurrences. Table 5.5.5 shows the resultant correlation coefficients for comparing total knowledge occurrences (i.e. both productive and non-productive knowledge category). The strongest correlation coefficients are shown in bold text.

Table 5.5.5: Correlations between check sheet scores and categories of knowledge production.

	Demonstration check sheet score	Requirements check sheet score	Total data knowledge	Total process knowledge	Total innovation knowledge
Demonstration check sheet scores		0.246	-0.401	**-0.455**	0.193
Requirements check sheet scores	0.246		0.249	0.297	**0.408**
Total data knowledge	-0.401	0.249		0.673	0.517
Total process knowledge	-0.455	0.297	**0.673**		0.471
Total innovation knowledge	0.193	0.408	0.517	0.471	

The strongest correlation in the table is 0.673 (r^2 of 45%) between process knowledge and data knowledge. This indicates that more data knowledge leads to more process knowledge, or that projects with more process knowledge also have

University of Cape Town

more data knowledge. There is a correlation of 0.471 (r^2 of 22%) between process knowledge and innovation knowledge, which indicates that more innovation knowledge occurrences need more process knowledge occurrences. There is virtually no relation between the *number* of process and data knowledge occurrences and how well requirements are met. However, the third strongest correlation, -0.455 (r^2 of 21%), which is a weaker relation, indicates that too much work on refining processes may lead to *less* successful product demonstrations.

5.5.6 Comparing check sheet scores and proportions of knowledge

This section investigates correlations between proportions of knowledge production (combining productive and non-production proportions) for each projects, and correlations between check sheet scores and these proportions of knowledge production. Table 5.5.6 shows the resultant correlation coefficients, with the greatest magnitude coefficients in bold.

Table 5.5.6: Correlations between check sheets and proportions of knowledge production.

	Demonstration check sheet score	Requirements check sheet score	Total data knowledge	Total process knowledge	Total innovation knowledge
Demonstration check sheet scores		0.246	-0.151	-0.259	0.513
Requirements check sheet scores	0.246		-0.256	0.034	0.241
Total data knowledge	-0.151	-0.256		**-0.678**	-0.211
Total process knowledge	-0.259	0.034	-0.678		**-0.575**
Total innovation knowledge	**0.513**	0.241	-0.211	-0.575	

The strongest correlation in the table is -0.678 (r^2 of 46%) between process knowledge and data knowledge. This indicates that higher levels of data knowledge may lead to a smaller proportion of process knowledge. This was evident in Project P2-2, in which members spent large amounts of time searching for information, and consequently spent less time experimenting with possible development methods.

The second strongest correlation, -0.575 (r^2 of 33%), between process knowledge and innovation knowledge is also an inverse relation. Higher levels of process knowledge appear to reduce the proportion of innovation knowledge obtained.

University of Cape Town

Putting too much effort into optimising processes reduces opportunities for innovation and, as discussed in Section 5.5.3.1, this may cause the prototype demonstration to be less successful. The correlation of 0.513 (r^2 of 23%) supports this conclusion, showing that larger portions of innovation knowledge lead to more successful demonstrations.

5.6 Team members' evaluation of ESAOA KMS

On completion of the project, team members were invited to respond to a questionnaire (see Section 3.8.7), which required them to evaluate the ESAOA KMS as they experienced it in their projects. This was a voluntary exercise. The participants were initially given hardcopy versions of the evaluation (after the demonstrations). After a few days, due to a general lack of response, they were also emailed softcopy versions the evaluation forms. In total, only ten of the 39 participants (i.e., 26%) of the participants responded. Some of the participants who did not respond were asked why they had not responded; the main reason was that they had other deadlines approaching (i.e., projects unrelated to Experiment 2 and examinations) and that they had to focus their time on these other responsibilities. Feedback from the participants is described below: firstly in terms of quantitative data (Section 5.6.1) and secondly in terms of qualitative data (Section 5.6.2).

5.6.1 Quantitative data: 5-point scale rankings

Participants as individuals (rather than as teams) were asked to rate the ESAOA KMS on a 5-point Likert scale [Meyers *et al.*, 2005] in terms of the following:

1) Overall impressions of the ESAOA KMS;
2) General issues with regard to using the ESAOA KMS (how they rated their ability to access knowledge, whether they would use the ESAOA KMS in their professional work, the amount of support they received in project meetings and workshops, and the level of difficulty of the project that they had worked on);
3) The ESAOA directory structure (in particular the extent to which they found the *go* utility effective, the ease of navigating the directory structure, whether they found the directory structure logical, and whether they found that the directory lists were becoming more user-friendly as the project progressed);

University of Cape Town

4) The ESAOA enter and exit (in particular whether they found the "source enter-esaoa" strategy effective, the command prompt confusing or clear, the usefulness of actions, and exiting the project);
5) The MakeMake (mm) utility, particularly its usefulness and the ease of changing platforms; and
6) The ESAOA environment variables. (Section 3.8.7 gives the full questionnaire; these points correspond to parts 0 to 5 of the questionnaire).

Table 5.6.1 shows that the participants gave the ESAOA KMS a high overall rating (an average of 4.6/5 or 92%), although there was considerable variation within the ratings given to specific aspects of the ESAOA KMS.

Table 5.6.1: Summary of evaluation data.

Participants	1) Overall	2) ESAOA: General				3) ESAOA: Directory structure				4) ESAOA: Enter and exit				5) mm utility		6) Environment
		Access	Professional	Meetings & workshops	Projects	Go utility	Ease	Logic	Benefit	Enter	Command prompt	Use	Exit	Useful	Chang platform	
1	5	3	3	3	3	3	3	3	1	5	3	3	2	5	4	4
2	5	3	1	1	3	0	0	0	0	5	5	2	2	5	0	1
3	4	3	4	3	3	4	4	5	3	5	5	2	2	4	1	4
4	4	2	4	3	3	4	0	4	0	5	5	0	0	5	0	0
5	5	3	3	3	3	2	3	3	3	3	4	4	2	3	4	4
6	4		3	2	2	0	4	4	4	4	4	3	3	4	0	1
7	5	3	3	3	1	1	5	4	3	5	5	3	3	4	4	0
8	5	4	4	3	3	5	2	3	4	5	4	3	2	5	3	2
9	4	3	3	3	2	4	5	5	4	5	4	3	2	5	4	2
10	5	4	4	2	3	4	4	2	5	5	4	5	5	5	3	1
Mode	5	3	3	3	3	4	4	4	3	5	5	3	2	5	4	4
Median	5	3.5	3	3	3	3	3.5	4	3	5	4.5	3	2	5	3	1.5
Average	4.6	3.6	3.2	2.6	2.6	3.0	2.7	3.5	2.4	4.7	4.4	2.7	2.3	4.5	2.3	1.9
%	92	72	64	52	52	60	54	70	48	94	88	54	46	90	46	38

University of Cape Town

From the row of average values in Table 5.6.1, it can be seen that high ratings were given to the effectiveness and clarity of the *esaoa-enter* command (average 91%), and to the dynamic, context-sensitive ESAOA command prompt (average 88%). The

MM utility scored highly (average 90%) in terms of its overall usefulness to the project, but low (average 38%) in terms of the ease of changing platforms.

The 'ESAOA general' category in Table 5.6.1 concerns: a) access to the ESAOA workspaces; b) the degree of professionalism of the artefacts and tools provided with the workspaces; c) the extent to which the training workshops and consultancy meetings were useful to the participants; and d) an indication of whether the projects were too hard (being unsuitable for the KMS, shown by a rating value of 1 or 2) or very easy (not needing the KMS, shown by a rating value of 4 or 5). Access to the ESAOA workspaces received a rating of 72%; this is likely attributable to the system being easy to access but at times being unavailable due to network problems. The level of professionalism of the tools did not receive a particularly high average rating (namely, 64%); this may be due to the tools of ESAOA KMS version 1 being rough prototype versions that were not sufficiently documented. The participants generally found the consultancy meetings (with the CKO or CKS) and the training workshops to be partly useful (giving a rating of 52%). The median rating of 3/5 for the project indicates that the participants found that their projects were on average at a suitable level of difficulty for use with the ESAOA KMS.

The category labelled 'ESAOA directory structure' in Table 5.6.1 relates mainly to the ESAOA workspaces. Of the ratings in this category, the 'logic' criterion scored highest (70%), which indicates that the participants generally found the structure of the workspaces to be logical. However, they also found that the structure was neither easy to use nor beneficial to their own development work.

The category labelled 'ESAOA: entry and exit' in Table 5.6.1 refers to the entering and exiting of BASH environments that wrap the ESAOA workspaces. The ratings show that the environment was generally easy to enter, but based on a rating of 54% for the 'use' criterion, the participants did not particularly value this feature.

As indicated earlier, the mm utility was found to be the most useful, although the participants either did not understand or did not use the platform tool (used as a way of specifying the hardware platform for which the code would be compiled).

The qualitative data described in the next section provides additional reasons for the discrepancies observed in the ratings.

University of Cape Town

5.6.2 Qualitative data: comments from participants

Comments were invited in terms of the specific difficulties and benefits experienced by the engineers who used the ESAOA KMS. The participants could add comments to the survey discussed in Section 5.6.1, or they could respond to the researcher by email. The participants of the qualitative responses were assigned numbers to match the numbers assigned to the quantitative participants.

5.6.2.1 Difficulties

Respondent 8 identified the main difficulties as "learning the directory structure, and how to code the PDM modules". Respondent 1 felt that the "compiling on ESAOA was often a problem...configuring make files [was] time consuming". Respondent 4 similarly identified "compilation" as difficulty, but added that this could be because he was "rusty with the use of make files". In contrast, Respondent 3 felt that "ESAOA was very helpful with compiling and running a project... however, adding additional files proved difficult in some cases...". Some of the participants experienced specific difficulties. Respondent 9 claimed that the ESAOA "works fine in the lab, but had trouble using on PC at home... problems with SQL". Respondent 5 pointed out that the ESAOA tools "[do not] immediately pick up that additional headers have been included... this give cryptic linking problems later". Respondent 3 indicated that "the data steward role seems less important compared to the innovation engineer... " and "more like an assistant to the other roles". Respondent 7 contributed that "better role titles would be nicer".

According to Respondent 7: "ESAOA helps us with the compiling and running", but that it "was difficult to change files for use of our project". Respondent 6 suggests that "getting used to [the ESAOA tools] in the beginning" was difficult. Respondent 10 claimed that "when it was working it worked fine... but there were cryptic error messages, and I didn't know what do when it breaks, especially for the mm tool". Respondent 10 constructively suggested that "more information or a manual about how the makefile is generated would probably help with the errors".

5.6.2.2 Benefits

There seemed to be general agreement that the ESAOA KMS was beneficial in terms of compilation. Respondent 3 listed "compiling", Respondent 9 listed "compiling and keeping files together", Respondent 7 listed "compiling and changing the platform", and Respondent 4 listed "quick compilation of programs onto any setup platform" as the main benefits of the ESAOA KMS. Respondent 1 stated that, "When

University of Cape Town

the make file works [ESAOA tools] increase [the] speed of compilation considerably". Respondent 5 claimed that the ESAOA tools "served to dramatically speed up the initial setup and to help [with the terminology of] the way code/documentation is archived". Respondent 8 felt that the ESAOA KMS "helped with the make files and porting". Respondent 2 felt that "once understood, it [i.e., the ESAOA KMS] [gave] a good understanding of a development environment". Respondent 6 stated that ESAOA tools "kept everything together, compilation help was fantastic (make files, etc)"; this respondent also noted that "[the] fclass program helps keeping track of what's what" – but the respondent also found "csv files" [i.e., the files maintained by the *fcs* and *fclass* tools] were "unexplained" and the content "cryptic". Participant 9 confirmed that ESAOA helped with "speed of porting between uCLinux and MicroMonitor". This is echoed by Respondent 10, who stated that, "ESAOA was very useful in the compiling, and intelligently combining the code modules and other parts of the programs together... this saved time for the project work".

The respondents mentioned mostly coding and compiling issues; little was mentioned about other issues of the ESAOA KMS, such as the use of the various ESAOA support tools and the functionality classifications (as described in Chapter 4). Both the benefits and difficulties identified by the respondents were taken into account in the further refinement and evolution of the ESAOA KMS (see Chapter 6).

University of Cape Town

5.7 General conclusions for ESAOA KMS version 1

This section begins with a summary of the types and frequencies of productive and non-productive knowledge occurrences in Experiment 2 (see subsection 5.7.1). This is followed by a comparison between results of Experiment 1 and Experiment 2 (in subsection 5.7.3). The main trends emerging from the results of Experiment 2 are then summarised (in subsection 5.7.4) and variables that influenced knowledge production and the use of ESAOA KMS version 1 are analysed (in subsection 5.7.5). Conclusions are drawn (in subsection 5.7.6) regarding the general effectiveness of ESAOA KMS version 1 in terms of the projects studied. Subsections 5.7.7 to 5.7.8 proceeds with additional findings related to the breakdown of knowledge production (i.e., further deconstructing the results).

5.7.1 Summary of knowledge occurrences

The number of knowledge occurrences, and the resultant percentage breakdowns for productive and non-productive knowledge occurrences for Experiment 2 were given in Table 5.3.1 (a) in Section 5.3. Table indicates that Projects P2-4, P2-6 and P2-9 had the greatest overall proportion of productive knowledge occurrences.

The percentage breakdown of knowledge occurrences, out of the total knowledge occurrences for the project concerned, and separated into the data, process and innovation knowledge categories, is provided in Table 5.7.1 (a). The last row of the table corresponds to the percentage values for productive knowledge in Table 5.3.1 (a) (i.e., the average percentage of productive data knowledge across all projects was 23%, similarly it was 13% for non-productive data knowledge, and so on). Table 5.7.1 (a) is further used in Section 5.7.3 for comparing Experiment 1 and Experiment 2 results. Section 5.7.4 discussed further relations between the proportion of innovation knowledge occurrences and the quality of artefacts and prototypes produced in the project.

Table 5.7.1 (a) shows that the project with the highest proportion of productive *data knowledge* was Project P2-9, the project with the highest proportion of productive *process knowledge* was Project P2-6, and Project P2-4 had the highest proportion of productive *innovation knowledge* occurrences.

Figure 5.7.1 provides an alternate visual view of the statistics shown in Table 5.7.1 (a); each bar shows, from the bottom of the figure, proportions of productive data, non-productive data, productive process, up to non-productive innovation knowledge occurrences for a project. In this figure, process knowledge occurrences can clearly be seen to comprise a sizable chunk of the knowledge occurrences in many projects.

University of Cape Town

Table 5.7.1 (a): Percentage of productive and non-productive knowledge occurrences out of total knowledge occurrences per knowledge category.

Project No.	All		Data		Process		Innovation	
	% Productive	% Non-productive	% Productive	% Non-productive	% Productive	% Non-productive	% Productive	% Non-productive
P2-1	71%	29%	21%	9%	26%	16%	24%	5%
P2-2	74%	26%	19%	17%	39%	10%	16%	0%
P2-3	70%	30%	23%	11%	30%	11%	18%	7%
P2-4	91%	9%	26%	2%	30%	4%	35%	2%
P2-5	71%	29%	26%	12%	21%	14%	24%	2%
P2-6	86%	14%	14%	9%	63%	6%	9%	0%
P2-7	73%	27%	27%	12%	21%	13%	25%	2%
P2-8	80%	20%	18%	6%	39%	14%	22%	0%
P2-9	81%	19%	33%	15%	30%	4%	19%	0%
P2-10	74%	26%	31%	21%	23%	5%	21%	0%
P2-11	64%	36%	18%	23%	36%	8%	10%	5%
P2-12	60%	40%	24%	20%	32%	16%	4%	4%
P2-13	74%	26%	28%	18%	26%	8%	21%	0%
Averages	75%	25%	23%	13%	32%	10%	20%	2%

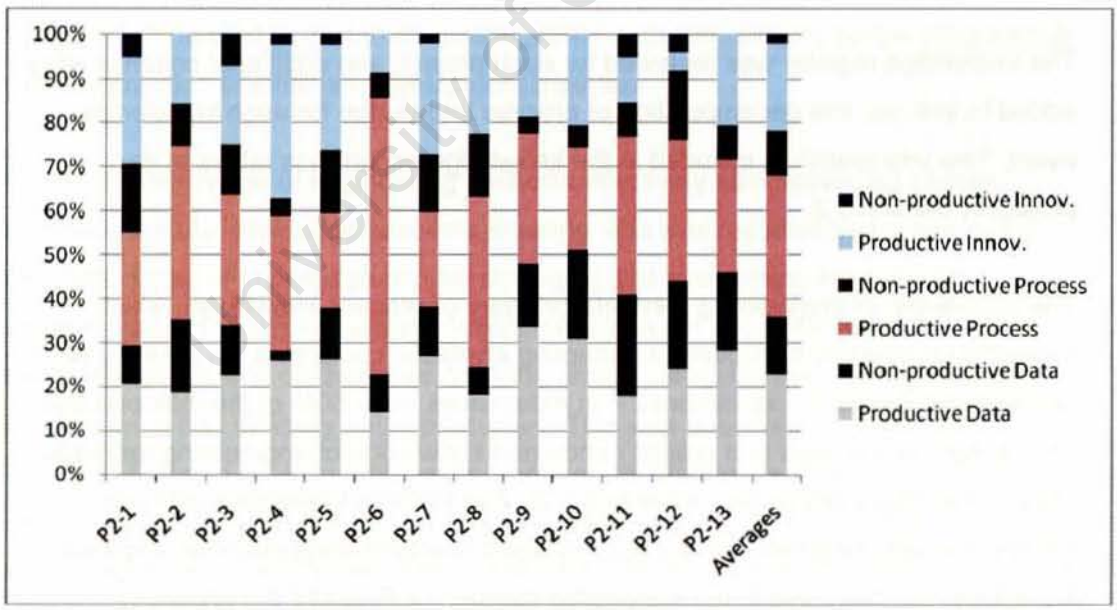


Figure 5.7.1: Bar chart showing percentage breakdown of productive and non-productive knowledge occurrences per knowledge category for each project.

University of Cape Town

5.7.2 Process knowledge components: role, logistics and innovation knowledge

Due to the relatively high proportion of process knowledge occurrences, process knowledge was separated into subcategories. Each occurrence of process knowledge tended to involved activities relating to the acquisition of one of the following three forms of process knowledge:

- **Role knowledge:** refining responsibilities or tasks to be performed by a particular group member (e.g., allocating specific development tasks to an individual), or more abstractly, modifying the responsibilities for a particular KMS role (e.g., the team agreeing that whenever one of the members performs the DS role, they need to make a note of which datasheets they have read).
- **Logistics knowledge:** concerned issues such as scheduling meetings, selecting tools, and ordering components.
- **Knowledge of engineering methods:** these comprise the procedures used to build or adapt artefacts in order to create a working prototype (the term used to describe this knowledge form is based on Brinkkemper *et al.* [1996]).

The knowledge register was reviewed for each project, and additional columns were added to indicate this decomposition of process knowledge for each knowledge event. This information is included in the knowledge occurrence tables of each project in Section 5.2.

The knowledge of engineering methods category of process knowledge, which concerns procedures for building or adapting artefacts, comprised the majority of process knowledge in Experiment 2 – in most cases, over 50% of the total process knowledge occurrences of a project concerned knowledge of engineering methods (as per the results presented in Section 5.2). The logistics knowledge and role knowledge subcategories are distinctly different to engineering methods: logistics knowledge (as indicative in the knowledge register for Project P2-1 shown in Appendix B) involves scheduling events or requisitioning resources, whereas role knowledge chiefly concerns team member responsibilities and allocation of development tasks.

University of Cape Town

The acquisition of role knowledge tends to occur towards the *start* of the project (as shown in the knowledge register for Project P2-1 and discussed in Section 5.7.2), whereas logistics knowledge usually occurs slightly further into the project. For the remainder of the project, all the process knowledge occurrences tend to involve acquiring knowledge of specific engineering methods.

The successful completion of ESAOA activities was clearly most dependent on obtaining knowledge of engineering methods (otherwise the team members would not have been able to construct their prototypes); but completion of these activities was also found to be highly dependent on effective role and logistics knowledge. These dependencies are elaborated on below.

Logistics knowledge related to when and where activities were performed; without this knowledge, team members could, for example, either waste time waiting for one another, thinking that the other person had already produced an effective method. The production, management and reuse of knowledge of engineering methods also tended to depend on knowing *where* methods should be developed, or where they were developed – which can also be seen as a dependency on logistics knowledge. Examples of this situation were seen in most projects; for instance, the developers in Project P2-1 needed to decide on meeting times and components before progressing with the creation of effective development methods.

Obtaining knowledge of engineering methods effectively also depended on role knowledge. Role knowledge concerns knowing who was responsible for producing particular forms of knowledge and for producing and maintaining the ESAOA artefacts related to this knowledge. When the teams did not understand, or disputed the responsibilities of certain roles, then the project tended to become side-tracked, and the team had difficulty being productive. This was observed particularly in Projects P2-2 and P2-8; in both cases, some of the team members lost direction, were misinformed of their assigned responsibilities and had to have further meetings or correspondence with fellow team members to clarify these issues. Role knowledge also related to the knowledge of which ESAOA activities tended to be performed by certain individuals; if the team members had a good understanding of this knowledge, individuals were less likely to waste time developing different development methods, and would instead know whom to call on to explain methods already produced.

University of Cape Town

The comparisons in Section 5.5 identified potential indicators for gauging the progress of the team and the success of the prototype based on the proportions of knowledge produced. For Experiment 2 projects, higher proportions of innovation knowledge (whether productive or non-productive) tended towards better prototype designs. Both higher proportions of productive data knowledge and higher proportions of productive process knowledge tended to have positive influences on design ratings. Teams that put a greater amount of effort into improving the quality of artefacts tended to acquire *lower* percentages of process knowledge and lower demonstration ratings (Project P2-1 was an exception).

While non-productive data, process and innovation knowledge are inevitable, it is important that productive data and process knowledge is obtained, because these are needed for innovation to occur, and for the prototype to be built. The correlations in Sections 5.5.5 and 5.5.6 support this, such as the direct relation between increasing innovation occurrences and increasing process occurrences, and larger proportions of innovation knowledge leading tending to higher demonstration ratings.

A number of correlations unexpectedly showed no, or only very weak, relations between variables, which the researcher had expected to be related. The correlations between artefact ratings and innovation knowledge, for instance, showed no relation, indicating that innovation may not demand high quality artefacts (however, projects that on average received high code and design review ratings did tend to achieve high scores for the review panel). The correlations further showed no relation between high concept ratings and successful demonstrations scores.

In Section 5.5, only a weak correlation emerged between demonstration scores and well-satisfied requirements. This corresponds to how the teams were evaluated, in that some teams spent time doing experiments rather than improving the quality of artefacts – this is unlikely to hold in the general case, especially for production projects; however, in this thesis, successful prototype demonstrations were considered more important than satisfying all the requirements (as explained in Section 1.1.2).

5.7.3 Comparison of Experiments 1 and 2

This section compares the findings from Experiment 1 with those of the Experiment 2 in terms of occurrences of productive and non-productive knowledge (see Section 4.5.3 for Experiment 1 results presented using event chains as a basis for analysis).

University of Cape Town

Table 5.7.3 (a) reproduces the statistics for productive and non-productive knowledge occurrences for Experiment 1 (shown in Table 4.27 in Chapter 4); recall that each Experiment 1 team used their own *ad hoc* KMS. Table 5.7.3 (b) provides the averages for productive and non-productive knowledge occurrences for Experiment 2, in which ESAOA KMS version 1 was applied.

Table 5.7.3 (a): Experiment 1 averages for productive and non-productive knowledge.

Knowledge Type	% Productive knowledge	% Non-productive knowledge	Total %
Data Knowledge	11	29	40
Process Knowledge	16	25	41
Innovation	10	11	20
TOTALS	36	64	100

Table 5.7.3 (b): Experiment 2 averages for productive and non-productive knowledge.

Knowledge Type	% Productive knowledge	% Non-productive knowledge	Total %
Data Knowledge	23	13	36
Process Knowledge	32	10	43
Innovation	20	2	21
TOTALS	75	25	100

A comparison between Tables 5.7.3 (a) and 5.7.3 (b) reveals on average 39% more occurrences of productive knowledge in Experiment 2 projects than in Experiment 1 projects. Experiment 2 had 12% more productive data knowledge occurrences, 16% more productive process knowledge occurrences, and 10% more productive innovation knowledge occurrences.

The percentage breakdown of data, processes and innovation knowledge shifted slightly between the two experiments. In Experiment 2, the proportion of data knowledge reduced by 4%; the proportion of process knowledge increased by 2% and the proportion of innovation knowledge increased by 1%. Clearly, similar proportions of these knowledge categories were used in both experiments.

The doubling of the proportion of *productive* innovation knowledge in Experiment 2 over Experiment 1 is a significant finding: it shows a shift in the Experiment 2 projects toward a greater focus on producing a larger component of productive innovation knowledge than in Experiment 1. However, this is not necessarily directly ascribed to more effective KM strategies used in Experiment 2 as part of using the ESAOA KMS version 1 than the *ad hoc* methods used in Experiment 1 – there is nevertheless a

University of Cape Town

strong indication that use of ESAOA KMS version 1, and its workspaces, support tools, roles and other aspects, may have had an beneficial effect on increasing the proportion of productive knowledge occurrences in Experiment 2.

There is a general improvement in the creation of productive knowledge in Experiment 2 over Experiment 1; this can be attributed to factors such as the pre-selection of development tools and training workshops (in Experiment 2 cross-compiler toolchains were integrated into ESAOA workspaces and used in training sessions). Provision of initial artefacts and template files, all carefully organised into ESAOA workspaces, is likely to have benefited Experiment 2 teams as well.

5.7.4 Trends noted from application of ESAOA KMS (version 1)

This section discusses emerging trends based on results from Section 5.2 and the correlations performed in Section 5.5. The first subsection concerns a relation between the quality of a prototype and the team's production of innovation knowledge. The second subsection concerns trend in event chains across the Experiment 2 projects. The third subsection presents a trend observed in all Experiment 2 projects concerning how teams progressed towards the acquisition of productive innovation knowledge.

5.7.4.1 Emerging relationship between innovation knowledge and quality of prototype

A trend across the projects emerged that suggests a relationship between the quality of the prototype and the production of productive innovation knowledge. Table 5.7.4 (a) shows that, on average, the higher the percentage of innovation knowledge occurrences, the higher the prototype scored. The exceptions are Projects P2-6, P2-11 and P2-12, each of which received high scores but had less than average innovation knowledge occurrences. These exceptions are likely due to the adjudicators not assessing the complexity of prototypes – Project teams P2-6, P2-11 and P2-12 all produced prototypes that were less complex than those created by the other teams. For instance, the Project P2-6 prototype (the Automation Headlights Dimmer) comprised components with simple interfaces, and had little embedded software (i.e., as it was simply sampling an ADC and adjusting a digital variable resistor; the project had no PC-based software). Likewise, the Project P2-11 prototype (the Supermarket Query Device) was fairly simple, as the embedded software mostly needed to communicate with a GPRS modem. Project P2-12 (the Personal Protection Device) similarly required minimal innovation.

University of Cape Town

Table 5.7.4 (a): Emerging trend indicating relationship between productive innovation knowledge and quality of the prototype.

Project No.	Project Title	% Productive Innovation knowledge occurrences	Quality of artefacts (design review 3 score)	Demonstration check sheet score	Comments
P2-1	Location-aware Tourist Information System	24	100	100	P2-1 produced a high quality prototype, and has a 5% higher than average amount of innovation knowledge.
P2-2	GPS Bus Tracker	16	65	65	P2-2 did not produce a high quality prototype, and had 3% less than average occurrences of productive innovation knowledge.
P2-3	Vibynet	18	78	78	P2-3 produced an average quality prototype and had 1% less than average occurrences of productive innovation knowledge.
P2-4	MyIP Phone Station	35	99	98	P2-4 produced a high quality prototype, and had a 16% higher than average amount of innovation knowledge.
P2-5	Home Automation System	24	96	90	P2-5 produced a high quality prototype, and had a 5% higher than average amount of innovation knowledge.
P2-6	Automation Headlights Dimmer	9	94	93	P2-6 is an exception: the task was a relatively simple one and the high mark was awarded based on functionality.
P2-7	Field Sensor for Maglev Trains	25	90	90	P2-7 produced a high quality prototype, and had a 6% higher than average amount of innovation knowledge.
P2-8	Cordless Stereo	22	80	78	P2-8 produced a high quality prototype, and had a 3% higher than average amount of innovation knowledge.
P2-9	Central Alarm Clock	19	90	80	P2-9 produced an average/high quality prototype, and had an average amount of innovation knowledge.
P2-10	Voice Activation System	21	90	91	P2-10 produced a high quality prototype, and had a 3% higher than average amount of innovation knowledge.
P2-11	Supermarket Query Device	10	81	82	P2-11 produced an average prototype, and had a 9% lower than average amount of innovation knowledge.
P2-12	Personal Protection Device	4	79	75	P2-2 produced an average prototype, and had a 15% lower than average amount of innovation knowledge.
P2-13	Vehicle Usage Tracker	21	95	93	P2-13 produced a high quality prototype, and had a 2% higher than average amount of innovation knowledge.
	Average:	19	88	86	

University of Cape Town

5.7.4.2 Emerging trends across event chains

This section concerns trends noted across the event chains, after a comparison across the data, process, and innovation graphs was done. Table 5.7.4 (b) describes the trends noted.

Table 5.7.4 (b): Trends emerging across the event chains.

Project No.	Trends note across event chains
P2-1	Data knowledge produced in the first 2/3 of project; plateaus were evident between process knowledge steps, innovation started in first 1/3, in steady consistent steps with a final steep rise just before completion.
P2-2	Data and process knowledge were produced across whole project (no plateaus), but innovation knowledge was more erratic.
P2-3	Data knowledge plateaus in final 1/3 of project, plateaus in process knowledge, early start to innovation and consistent steps.
P2-4	Data knowledge produced in first 2/3 of project; plateaus between process knowledge, innovation started in first 1/3, in steady consistent steps with final steep rise.
P2-5	Data knowledge plateau in final 1/3 of project, plateaus in process knowledge, innovation started in first 1/3, in steady consistent steps with final steep rise.
P2-6	Late start to data knowledge production, consistent process knowledge, low occurrence of innovation knowledge. The task was a relatively simple one and the high mark was awarded on the basis of functionality.
P2-7	Data knowledge occurrences quite late into the project, plateaus in the process knowledge, steady consistent increases in innovation.
P2-8	Data knowledge occurrences quite late into the project, plateaus in the process knowledge, erratic increases in innovation.
P2-9	Data knowledge occurrences late into the project, plateaus in process knowledge occurrences, few innovation occurrences.
P2-10	Data knowledge occurrences late into the project, plateaus in process knowledge occurrences, some erratic innovation occurrences.
P2-11	Data knowledge produced in first 2/3 of project; plateaus between process knowledge, some erratic innovation occurrences.
P2-12	Data knowledge produced late into the project, process knowledge has plateaus, only one instance of innovation knowledge.
P2-13	Data knowledge produced in first 2/3 of project; plateaus between process knowledge, innovation starts in first 1/3, in steady consistent steps with final steep rise.

In the three most successful projects (P2-1, P2-4 and P2-13) the following trends were noted:

1. Data production began in the first third of the project and flattened out in the final third of the project (see Graphs 5.2.1 (a), 5.2.4 (a) and 5.2.13 (a)). Unproductive data knowledge occurred more frequently at the start than at the end of the project.
2. The three process knowledge graphs (Graphs 5.2.1 (b), 5.2.4 (b) and 5.2.13 (b)), showed productive process knowledge occurrences from the start (usually reflecting the allocation of roles and attending to logistical matters). Thereafter process knowledge reached a plateau as the team members

researched the problem and attempted to find the data they needed. Process knowledge then increased as the teams applied various engineering methods to the problem. Process knowledge thereafter reached another plateau as the team found solutions and produced innovation knowledge.

3. Occurrences of innovation knowledge began in the first half of the project, grew in steady and consistent steps, often with a sudden rise towards the project conclusion (see Graphs 5.2.1 (c), 5.2.4 (c) and 5.2.13 (c)).

These patterns can be detected in all of the thirteen projects, but are most pronounced in the progression of the top three projects. While there were no truly weak projects, the project that was consistently identified by the researcher and the review panel as the least successful one (i.e., Project P2-2) showed the most deviation from the trends previously described.

In the Project P2-2 graphs, occurrences of data and process knowledge increased throughout the project, without the plateaus as previously described. There was a late start in occurrences of process knowledge and data knowledge (see Graphs 5.2.2 (a) and 5.2.2 (b)), with both continuing to the project end. The innovation 'steps' were fewer and more erratic than in the more successful projects.

Team members of Project P2-1 effectively applied the ESAOA version 1 roles from an early stage, and came together towards the final third of the project to support innovation. In Project P2-2, however, ESAOA version 1 roles were ineffectively applied, resulting in an unequal distribution of work and a subsequent lack of support for innovation.

A trend noted across all the productive versus non-productive knowledge occurrence graphs was the divergence trend, showing productive knowledge occurrences growing more rapidly than non-productive knowledge occurrences.

5.7.4.3 Progression towards innovation

All Experiment 2 projects showed a similar, repeating cycle, which comprised two sub-cycles. The first sub-cycle, termed a *progression towards process*, focused on reappearing similar types of activities related to the acquisition of data knowledge and process knowledge. Once effective process knowledge was obtained, the activities quickly changed to ones that predominantly involved the application of

University of Cape Town

process knowledge to formulate innovation knowledge – this second sub-cycle is termed progression towards innovation knowledge.

A progression towards process tended to involve activities such as web searches to find information, reading data sheets, and testing potential development methods – essentially, data knowledge and process knowledge were usually built together. Solving problems that produced process knowledge tended to guide needs for data knowledge. The activities in the progression towards innovation typically concern implementing design ideas that satisfy one or more project requirements, done by applying development methods established in a preceding progression towards process.

All Experiment 2 projects demonstrate examples of these progressions: a particular example is drawn from the sequence of event chains 14 to 18 in Project P2-5 (the Home Automation System). In event chains 12, 13 and 14 of Project P2-5, the team focused on researching components (event chains 12 and 13 involved reading about relays and researching DC motor control techniques in code downloaded from the internet). Event chain 14 involved testing an alternate development method for the control of relays via software (a method that was later discarded because a previously established method was easier to use). Event chain 16 involved production of innovation knowledge, in which an idea for controlling the relays in actuator nodes was successfully implemented (thereby achieving a requirement of the project). The collection of event chains 12, 13, 14 and 18 are thus related, and together can be seen as leading towards the innovation knowledge in event chain 18. In this sequence of events, event chains 12, 13, and 14 essentially related to establishing the data knowledge and process knowledge (i.e., a progression towards process) that was required to produce the innovation knowledge of event chain 16 (i.e., a progression towards innovation).

Progressions do not always result in success outcomes. For example, a progression towards process might not lead towards a progression towards innovation (e.g., the developers may be unable to build the knowledge needed to achieve a goal). Similarly, a progression towards innovation may end unsuccessfully, such as the developers running short of time focus on other priorities. Consequently, these progressions can be further qualified:

University of Cape Town

- **Productive progression:** a progression that was successful (e.g., a productive process progression that led to process knowledge later used in an innovation progression).
- **Non-productive progression:** an unsuccessful progression.
- **Episode:** one or more process progressions that led to one or more innovation progressions.
- **Productive episode:** an *episode* involving productive progressions.
- **Non-productive episode:** an *episode* comprising non-productive progressions.

The concepts of progressions and episodes defined above can be modelled visually. Figure 5.7.4 (a) shows a productive episode, and Figure 5.7.4 (b) illustrates a non-productive episode. The upper portion of each figure visually models phases of knowledge production within an episode (as described earlier in this section), which usually starts with an idea, such as a design concept to test, followed by acquisition of data and process knowledge (during a progression towards process, where data and process knowledge event chains are modelled as rectangles), and finally innovation knowledge is produced (during a progression towards innovation, where the innovation knowledge event chain is shown as a star). The lower portion of each figure shows productive knowledge occurrence graphs resulting from the sequence of event chains modelled (suggesting visual pattens in these graphs that allude to the identification of these progressions).

A productive episode (see Figure 5.7.4 (a)) typically involves a steady build-up of productive data and process knowledge occurrences, followed by an innovation occurrence. In Experiment 2 many of the more successful projects produced mainly productive episodes as the project progressed, which resulted in the divergent trend discussed in Section 5.7.4.2. Trends of this type indicate the team is working *productively* (i.e., producing what is termed *productive knowledge* in this thesis); this may indicate that the KMS is operating effectively during these activities.

University of Cape Town

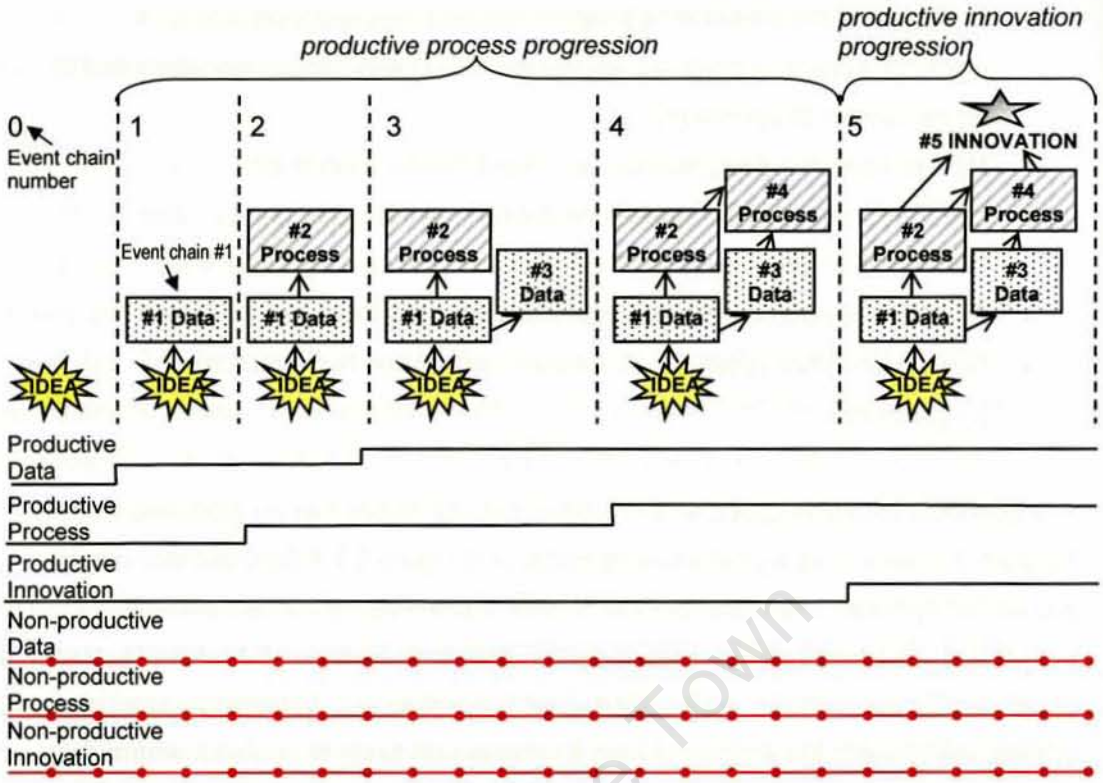


Figure 5.7.4 (a): Model of a *productive episode* (top) and corresponding knowledge occurrence graphs (bottom).

A non-productive episode (see Figure 5.7.4 (b)) typically involves one or more plateaus in the occurrence of productive knowledge (and increments in the non-productive knowledge) seen in knowledge occurrence graphs. These episodes were more common in less successful Experiment 2 projects (e.g., Project P2-2). Trends of this type indicate a team is either performing *unproductively* (as in producing *non-productive knowledge* as defined previously), or the KMS is not operating effectively. Note that the knowledge occurrence graphs were produced after the project had ended (hence it is known which knowledge events are productive, thus there are no increments and later decrements in productive trend lines shown in Figure 5.7.4 (b)).

University of Cape Town

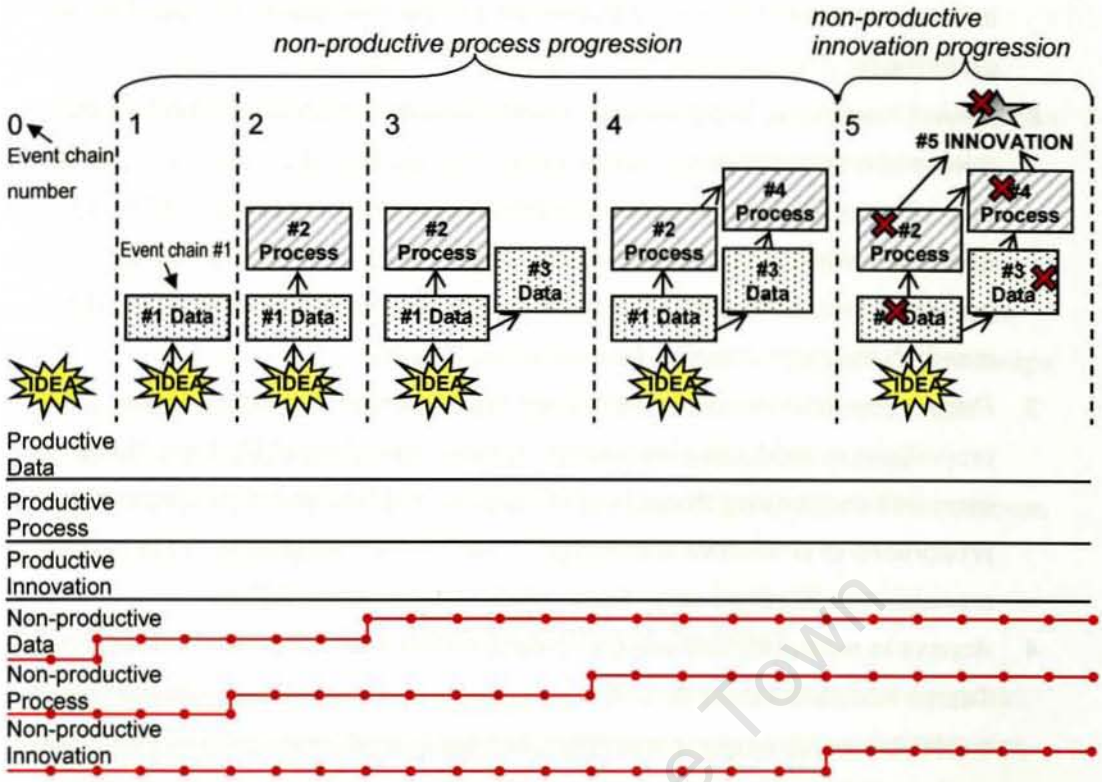


Figure 5.7.4 (b): Model of a *non-productive episode* (top) and corresponding knowledge occurrence graphs (bottom).

5.7.5 Variables that affected the ESAOA KMS (version 1)

The evaluations of the prototypes, done by the review panel (see Section 5.4), showed that not all productive innovation knowledge was at the same level of sophistication and complexity. Consequently, if a project achieves a higher proportion of productive innovation knowledge than another does, it does not necessarily mean that the first project will result in a more successful product. For instance, Project P2-13 had 21% productive innovation knowledge compared to 22% for Project P2-8 (i.e., a fairly insignificant difference in the proportion of innovation knowledge), yet the review panel considered Project P2-13 superior to Project P2-8. There was a range in the appropriateness, elegance, attention to detail, and functionality in the ES prototypes and solutions used. The following list identifies a number of variables that are likely to affect the amount and quality of productive knowledge, and productive innovation knowledge in particular:

1. *Level of difficulty of development task*. Simple or routine tasks will probably not generate much, if any, innovation knowledge (such as in the case where Project P2-1 developers wired up a GPS module following a datasheet – little

University of Cape Town

innovation knowledge was produced around the time that these tasks were performed).

2. *Project leadership*. Experiment 2 results indicate that an understanding of the roles and responsibilities leads to a higher proportion of productive knowledge (e.g., Project P2-2 experienced the most leadership difficulties and had the relatively lowest productive innovation knowledge). This finding confirms more general principles of good leadership leading to productive work as per the research literature (such as Nonaka *et al.*, 2001a).
3. *Project coordination*. Well-coordinated teams tended to produce higher proportions of productive knowledge; for example, Project P2-1 members were well coordinated throughout the project and had one of the highest proportions of productive knowledge. This finding corresponds to the project management literature (e.g., Ford, 1995; Wagner *et al.*, 2009).
4. *Access to resources* (including knowledge resources, adequate facilities, etc). Teams that established good filing practices, and made their resources easily accessible to fellow group members, tended to work more productively. This was particularly noticeable in Project P2-4; the team made good use of ESAOA artefact classification tools, maintained well structured directories and file names, and had the highest proportion of productive innovation knowledge.
5. *Level of understanding of ESAOA scripts, programmes and directory structures*. Teams, such as Project P2-1 and P2-4, which put a large effort into understanding and applying the ESAOA system had higher levels of productive knowledge production compared to other projects.
6. *Level of ability to install the ESAOA distributions on alternate Linux-based operating systems*. The results indicate that teams that included developers proficient in Linux were at an advantage in productive knowledge production (likely attributable to ESAOA being designed for use on such a platform).
7. *Level of the data stewards' research skills* (and willingness to engage in information searches and build a knowledge base for the project). Teams in which the data steward role was engaged in information searches and were willing to contribute communal artefacts (such as team P2-1), also tended to have higher levels of productive knowledge production.
8. *Level of the process engineers' knowledge of engineering methods* (including script-writing and experimentation methods).

University of Cape Town

5.7.6 Effect of the ESAOA KMS (version 1)

The findings of Experiment 1 were confirmed by Experiment 2 with regard to the presence of data, process, innovation knowledge categories and productive and non-productive knowledge types across these categories. Knowledge events (as explained in Experiment 1 and confirmed in Experiment 2) can be clustered into knowledge event chains. Overall, more productive knowledge event chains than non-productive knowledge event chains were found across all thirteen projects in Experiment 2. There was an improvement in the proportion of productive knowledge occurrences in Experiment 2 in comparison with Experiment 1. For the thirteen Experiment 2 projects, the proportion of productive innovative knowledge occurrences for a project has been shown to be a possible indicator for the success of the prototype developed.

5.7.7 Study of knowledge forms contributed by roles

In Experiment 2, it was found that the contribution of data knowledge and process knowledge was made by both data steward (DS) and process engineer (PE) roles (the inspiration that led to this observation was made only after the knowledge register was complete, and occurred while investigating project P2-11). Although it was not hypothesised in the methodology, it was assumed that the DS would contribute notably more data knowledge than the PE and, similarly, that the PE would produce substantially more process knowledge than that provided by the DS. A difficulty arose in determining the contribution of knowledge produced according to role because the knowledge register did not show this information. Determining the answer to this problem would have involved reinvestigating the original data, which would have taken a significant amount of time to complete if all thirteen projects were studied. Instead, to get an impression of how knowledge forms are contributed per role, the original data for a selection of projects was revisited. The choice of projects to review was based on the artefacts quality rating (see Table 5.4.1 (a)). The project with the highest score, the project with the lowest score, and a project with the median score was chosen. The projects were P2-1 (which scored 100%), P2-2 (which scored 65%) and P2-10 (which scored 90%).

In order to conduct this analysis, additional columns were added to the knowledge registers for P2-1, P2-2 and P2-11. Recall that each row in the knowledge register refers to a single knowledge occurrence, and that it is described as knowledge of data (KD), knowledge of process (KP) or knowledge of innovation (KI) occurrence.

University of Cape Town

The data source for each knowledge occurrence had to be reviewed, in order to determine which members were involved in the knowledge management activities concerned. The columns added to the knowledge register for the three projects are illustrated in Table 5.7.7 (a). The columns are defined as follows:

- *KO No.:* Knowledge occurrence number (same as the first column)
- *DS con:* A value of '1' if the DS role was involved in producing knowledge for this data occurrence (a value '0' otherwise).
- *PE con:* A value of '1' if the PE role contributed to producing the knowledge.
- *IE con:* A value of '1' if the innovation engineer (IE) role played a part in producing the knowledge.
- *Delta:* A value of '1' if the DS provided process knowledge, or the PE provided data knowledge, or the IE provided either data or process knowledge.
- *DS KD:* A value of '1' if DS contributed data knowledge.
- *DS KP:* A value of '1' if DS provided process knowledge.
- *DS KI:* A value of '1' if DS provided innovation knowledge.
- *PE KD:* A value of '1' if PE provided data knowledge.
- *PE KP:* A value of '1' if PE provided process knowledge.
- *PE KI:* A value of '1' if PE provided innovation knowledge.
- *IE KD:* A value of '1' if IE provided data knowledge.
- *IE KP:* A value of '1' if IE provided process knowledge.
- *IE KI:* A value of '1' if IE provided innovation knowledge.

University of Cape Town

Table 5.7.7 (a): Excerpt from Project P2-1 knowledge register showing columns added to track contribution of knowledge form per role for each knowledge occurrence.

KO No.	DS con	PE con	IE con	Delta	DS KD	DS KP	DS KI	PE KD	PE KP	PE KI	IE KD	IE KP	IE KI
1	0	1	0		0	0	0	0	1	0	0	0	0
2	0	1	0		0	0	0	0	1	0	0	0	0
3	1	1	0	1	0	1	0	0	1	0	0	0	0
4	0	1	0		0	0	0	0	1	0	0	0	0
5	1	0	0		1	1	0	0	0	0	0	0	0
6	0	1	0		0	0	0	0	1	0	0	0	0
7	1	0	0		1	1	0	0	0	0	0	0	0
8	0	1	0		0	0	0	0	1	0	0	0	0
9	1	0	0	1	0	1	0	0	0	0	0	0	0
10	0	1	0		0	0	0	0	1	0	0	0	0

The third entry in Table 5.7.7 (a) shows a situation in which the data indicate that both the DS and PE produced logistics process knowledge (note the Delta value of '1' flagging the entry). According to the source email (email number 11), this involved deciding on times and a venue in which to meet. The ninth entry concerned further logistics process knowledge, which was contributed by the DS and not the PE.

The contribution of knowledge forms per role for projects P2-1, P2-2 and P2-11 is shown respectively in Table 5.7.7 (b), Table 5.7.7 (c) and Table 5.7.7 (d). Each table indicates the number of knowledge occurrences that each role contributed, together with a percentage showing the proportion of knowledge occurrences a role was involved with for a certain category of knowledge.

Table 5.7.7 (b): Contribution of knowledge forms per role for Project P2-1.

Role contributing	DK occurrences	DK %	PK occurrences	PK %	IE occurrences	IK %
Contributed by DS	12	71%	4	17%	4	24%
Contributed by PE	6	35%	19	79%	3	18%
Contributed by IE	1	6%	2	8%	17	100%
Produced together	2	12%	1	4%	7	41%
Total occurrences	17		24		17	

In Project P2-1 (according to Table 5.7.7 (b)), the DS was involved in contributing 71% of data knowledge, the PE 35%, and the IE 6%. 12% of data knowledge occurrences were produced by roles working together (and the data artefacts were located or created and worked on jointly), only 4% of process knowledge was produced by multiple roles, and 41% of innovation knowledge was produced by multiple roles (24% by the DS and 18% by the PE). The DS was involved in 20 knowledge occurrences, the PE in 28 and the IE in 20. Table 5.7.7 (c) indicates the percentage contribution of each role in this project.

Table 5.7.7 (c): Contribution of knowledge forms per role for Project P2-2.

Role contributing	DK occurrences	DK %	PK occurrences	PK %	IE occurrences	IK %
Contributed by DS	17	47%	0	0%	2	13%
Contributed by PE	13	36%	22	44%	1	6%
Contributed by IE	14	39%	32	64%	15	94%
Produced together	8	22%	4	8%	2	13%
Total occurrences	36		50		16	

The role contributions for knowledge occurrence in Project P2-2 (see Table 5.7.7 (c)) differed markedly to those for Project P2-1. In the case of Project P2-2, the DS was

University of Cape Town

involved in contributing 47% of data knowledge, the PE 36%, and the IE 39%. 22% of data knowledge occurrences were produced by roles working together. The IE contributed more process knowledge than the PE did, and the DS provided no process knowledge at all. The DS contributed to 13% of innovation knowledge occurrences. The DS was involved in 19 knowledge occurrences, the PE in 36 and the IE in 61. Table 5.7.7 (d) indicates the percentage contribution for the roles.

Table 5.7.7 (d): Contribution of knowledge forms per role for Project P2-10.

Role contributing	DK occurrences	DK %	PK occurrences	PK %	IE occurrences	IK %
Contributed by DS	17	85%	4	36%	3	38%
Contributed by PE	5	25%	8	73%	1	13%
Contributed by IE	1	5%	1	9%	7	88%
Produced together	3	15%	2	18%	3	38%
Total occurrences	20		11		8	

The DS was involved in 85% of data knowledge occurrences in Project P2-10 (see Table 5.7.7 (d)), while the PE contributed to 25% of the occurrences and the IE to 5%. 15% of data knowledge occurrences were produced jointly. The PE contributed 73% of process knowledge and the DS 36%. The DS contributed to 38% of innovation knowledge occurrences. The DS was involved in 24 knowledge occurrences, the PE in 14 and the IE in 9. Figure 5.7.7 indicates the percentage contributions for these roles.

Based on these results, the contribution to data, process and innovation knowledge was not distributed equally among the DS, PE and IE roles in the three projects investigated. The distribution also varied between projects. In P2-2, the IE contributed more process knowledge occurrences than the PE did, whereas the IE participated in almost as many data knowledge occurrences as did the DS. Project P2-1, in comparison to Project P2-2, had a more balanced spread of role participation, in that the DS contributed the most data knowledge, the PE the most process knowledge, and the IE created most of the innovation knowledge.

In projects P2-1 and P2-10, over one third of innovation knowledge was produced jointly (41% in P2-1 and 38% in P2-10). However, in Project P2-2 only 13% of the innovation knowledge was jointly produced. The greatest proportion of jointly contributed data knowledge was 22% (in Project P2-2), while the largest percentage of jointly produced process knowledge was 18% (in Project P2-10). Based on these results, it appears that the more successful projects (P2-1 and P2-10) had multiple

University of Cape Town

roles participating in innovation knowledge production (i.e., innovation knowledge was enhanced by other roles being involved in its management). This finding suggests it is necessary to change the support structure for the ESAOA KMS, to promote a flow of innovation knowledge from the DS and PE towards the IE – in addition to transferring data knowledge from the DS to the PE, and process knowledge from the PE to the IE.

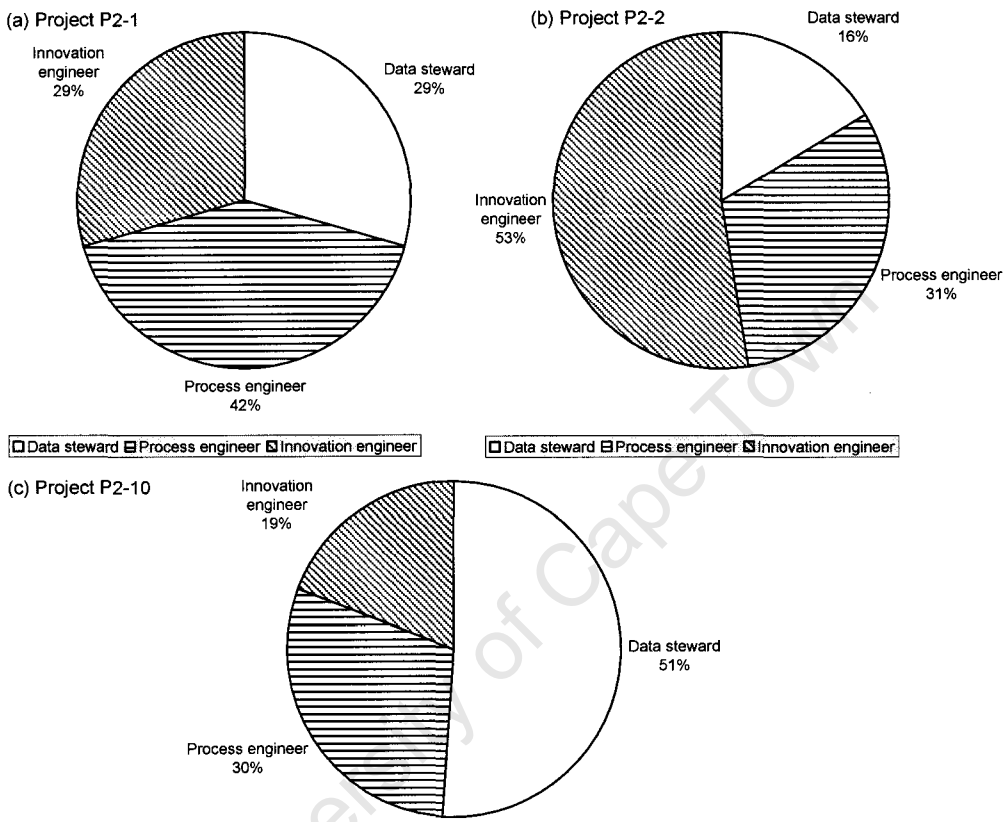


Figure 5.7.7: Pie charts showing contribution of each role for (a) Project P2-1, (b) Project P2-2 and (c) Project P2-10.

University of Cape Town

5.7.8 Tool versus component knowledge occurrences

As discussed in Section 5.7.7, it was found that the contribution of data and process knowledge was made by both DS and PE roles, and it was concluded that these role needed to be revised in the second version of the KMS. The knowledge register for each project in Experiment 2 has a component column and a tool column. A value of 1 in the component column indicates the knowledge occurrence related mainly to researching a component, whereas a value of 1 in the tool column implies that the knowledge occurrence mainly concerned study of a specific tool. A value of 0 was placed in both tool and component columns if the knowledge occurrence involved

more use, than study, of a tool or component. The tool and component columns were summed to indicate an overall view of the amount of component study compared to the amount of tool study that took place in the project.

From the Experiment 2 data, it can be seen that more knowledge occurrences relate to acquiring knowledge of components rather than knowledge of tools. It was therefore decided to replace the DS with a role that focuses on researching components, and to shift the responsibility of acquiring data knowledge of tools to the PE. In order to validate this plan, the Experiment 2 knowledge registers were processed to compare the proportion of knowledge events involving researching components to those that concerned researching tools. Table 5.7.8 (a) provides an example (from Project P2-1) that shows how the tools and component columns were used. In this example, the second row shows an occurrence of productive data knowledge, which involved the study of a component. The fourth row shows another case of productive data knowledge, but one that concerned both study of a component and a tool.

Table 5.7.8 (a): Excerpt from Project P2-1 knowledge register showing the tools and components columns.

No.	Type	...	KD	KP	R	L	KI	PK	NPK	...	KO No.	...	Tool	Component
240	E	...	1					1		...	30	...	0	0
467	E	...	1					1		...	31	...	0	1
414	E	...					1	1		...	32	...	0	0
227	E	...	1					1		...	34	...	1	1

University of Cape Town

Knowledge occurrences that involve the study of tools are referred to as 'tool knowledge occurrences', while research of components are referred to as 'component knowledge occurrences'. Although most of these involved production of data knowledge, they could also involve the creation of process knowledge. The total number of component knowledge occurrences and the sum of tool knowledge occurrences for each project in Experiment 2 are shown in Table 5.7.8 (b). The column titled 'Both' combines the number of tool knowledge occurrences and component knowledge occurrences in order to provide a percentage split for these.

Table 5.7.8 (b) shows that Project P2-2 involved the highest number of tool knowledge occurrences, specifically 27, which is more than three times the average number of tool knowledge occurrences for all projects. Project P2-2 also had the greatest number of component knowledge occurrences, viz. 47, which is also more

than three times the average number of component knowledge occurrences for all projects. The large accumulation of data knowledge (especially the amount of non-productive data knowledge) produced in Project P2-2, the many dead-ends, the difficulties in finding useful sample code, and the various abandoned strategies, all account for this project's comparatively large number of component knowledge occurrences and tool knowledge occurrences.

Table 5.7.8 (b): Tool versus component knowledge occurrences.

Project	Tool knowledge occurrences	Components knowledge occurrences	Both	% Tool knowledge occurrences	% Components knowledge occurrences
P2-1	13	21	34	38%	62%
P2-2	27	47	74	36%	64%
P2-3	3	11	14	21%	79%
P2-4	8	20	28	29%	71%
P2-5	3	12	15	20%	80%
P2-6	4	10	14	29%	71%
P2-7	11	17	28	39%	61%
P2-8	6	18	24	25%	75%
P2-9	2	7	9	22%	78%
P2-10	14	4	18	78%	22%
P2-11	4	10	14	29%	71%
P2-12	3	7	10	30%	70%
P2-13	8	13	21	38%	62%
AVERAGE	**8.15**	**15.15**	**23.31**	**35%**	**65%**

According to Table 5.7.8 (b), component knowledge occurrences took place more frequently (approximately twice as often) as tool knowledge occurrences. In most of the projects, with the exception of P2-10, there were approximately twice as many component knowledge occurrences than tool knowledge occurrences. This finding implies that more work is involved in studying components than studying tools. In the case of Project P2-10 (the Voice Activation System), the team, unlike most of the other project teams, started building process knowledge right at the beginning of the project and furthermore read up on a comparatively small number of components (in fact, this team read up on the smallest number of components).

5.7.9 Logistics and role process knowledge

In Experiment 2, process knowledge was divided among the categories of role, logistics and other. General trends across the projects for these categories of process knowledge are summarised in Table 5.7.9. Generally, the team leader was involved in producing both role (i.e., refining the responsibilities of each team member) and logistics knowledge (i.e., deciding on venues and times to perform

University of Cape Town

ESAOA activities) – the team leader's involvement was generally associated with decisions related to this knowledge. The process engineer created most other forms of process knowledge (Section 5.7.8 discusses exceptions that were noted, such as the innovation engineer producing most of the process knowledge in Project P2-2).

Table 5.7.9: Separation of role, logistics, and other knowledge.

Project No.	Role	Logistics	Engineering Methods	Combined	% Role	% Logistics	% Methods
P2-1	7	4	13	24	29%	17%	54%
P2-2	10	8	32	50	20%	16%	64%
P2-3	3	7	8	18	17%	39%	44%
P2-4	4	2	10	16	25%	13%	63%
P2-5	3	3	9	15	20%	20%	60%
P2-6	8	5	11	24	33%	21%	46%
P2-7	3	2	13	18	17%	11%	72%
P2-8	4	2	20	26	15%	8%	77%
P2-9	3	2	4	9	33%	22%	44%
P2-10	2	3	6	11	18%	27%	55%
P2-11	4	2	11	17	24%	12%	65%
P2-12	4	2	6	12	33%	17%	50%
P2-13	2	0	11	13	15%	0%	85%
AVERAGE	4.38	3.23	11.85	19.46	23%	17%	60%

The results in Table 5.7.9 show that the bulk of process knowledge (60% across projects) on average fell into the category of other process knowledge. On average, 23% of the process knowledge occurrences involved role knowledge, while 17% involved logistics occurrences. Considering that the team leader was involved in most of the role and logistics knowledge occurrences, it can be deduced that the team leader was involved in 40% of the process knowledge occurrences. This involvement concerned writing emails, updating task allocation lists, scheduling meetings, chairing meetings, and taking minutes.

Occurrences of other process knowledge mostly involved the production of development methods (such as how to adapt or interface components using a specific set of development tools). Generally, this knowledge was considerably more complex and time consuming to produce than that of the role or logistics knowledge (this aspect necessitated the need to separate out the role and logistics knowledge from other forms of process knowledge). While the time taken to produce each occurrence of role or logistics knowledge was in the order of hours (e.g., two hours to schedule a meeting and plan an agenda), each occurrence of process knowledge took considerably longer, more in the order of days than in the order of hours. This observation is valuable because it provides a broad guideline for deciding the way in

University of Cape Town

which roles can be allocated fairly. In particular, team management duties are likely to entail significantly less time than ESAOA activities in which process knowledge is created.

5.8 Implications for ESAOA KMS version 2

The findings of Experiment 2 suggest a number of potential refinements to ESAOA KMS version 1, in order to create ESAOA KMS version 2. The implications of the findings for changes to the ESAOA distributions (i.e., the install packages) are described in Section 5.8.1, whereas modifications to ESAOA roles are discussed in Section 5.8.2.

5.8.1 ESAOA distribution

A number of suggestions for the refinement of the ESAOA distribution emerged from the project and evaluation data. These subsections that follow focus on these refinements.

5.8.1.1 ESAOA tools – technical installation guidelines (version 2)

The *ESAOA-Tools* distribution was originally implemented using the Knoppix 4.2 (Debian-type) Linux operating system. Teams 12 and 13 attempted to install *ESAOA-Tools* on a different operating system (namely, Redhat Linux); although they succeeded in doing so, it was a lengthy procedure. Team 2 succeeded in booting Knoppix from a Knoppix Live CD and setting up and using ESAOA communal and team distributions (and the included ESAOA tools) on a USB flash memory stick. Notes from these teams on accomplishing these tasks should be incorporated into the ESAOA manual (i.e., the user manual for the ESAOA KMS).

5.8.1.2 Increased flexibility in ESAOA tools (version 2)

ES engineering teams are likely to undertake a range of projects, some of which might comprise more straightforward development tasks, while others may be more complex. ESAOA KMS version 2 should be able to accommodate different levels of complexity within and across projects. More straightforward development tasks could be 'fast-forwarded' through the ESAOA KMS by enabling the developers to skip options or steps in the procedures. The training program should be adapted accordingly.

University of Cape Town

5.8.2 ESAOA roles

Additional defined roles emerged from Experiment 2, for example, that of workspace administrator (who is in charge of maintaining the team's ESAOA workspace). The data steward role was removed and replaced by the component researcher (responsible for finding information and reading up on ES components). The process engineer was redefined to focus on methods engineering, which includes knowledge of development tools. The team leader's role was expanded to include the tracking and storing of logistics and role knowledge. The innovation engineer in ESAOA KMS version 2 was more clearly focused on applying process knowledge to test innovations, while relying on the other roles (i.e., the process engineer and component researcher) to provide support as well as suggesting ideas to test. The ES engineers in both Experiment 1 and 2 were novice engineers; ESAOA KMS version 2 should take into account both novice roles and more experienced ES engineering roles, as well as issues in role extension (see Section 6.1.3.4).

5.8.2.1 Training

Training was offered to all team members at the start of Experiment 2. This training tended to be specific to the ESAOA tools and should be extended to include more focused and specialised training for the ESAOA roles. The ESAOA manual should include specific guides for the different roles, and the ways in which interaction between team members could be facilitated.

5.8.3 ESAOA technical manual

A simplified ESAOA technical manual (an addition to the more complete ESAOA user manual) would assist ES engineers in the first stages of learning to use the ESAOA KMS, and in understanding the ESAOA scripts, programmes and directory structures. The manual could include strategies for information finding and building the integrated knowledge base. The manual should guide the users with regard to common engineering methods, including script-writing and experimentation methods, and typical problem-solving repertoires. Some of these items could be accomplished by using 'concept cartoons' (which the Project P2-10 team came up with as an idea of exploring and expressing the functionality of their system).

5.8.4 ESAOA project management

The ESAOA KMS version 2 should be supported by a more complete user guide (termed the *ESAOA manual*), which will enable team members to understand the ESAOA roles in terms of allocation, functions, and role support. The user guide

University of Cape Town

should also address processes, in particular the need for the ES development team to begin developing productive data knowledge in the first third of the time allocated, and to end all data searches by the final third of the project for optimal enabling of innovation knowledge occurrences.

5.8.5 Team workspace

The team workspace (implemented as a shared file directory) was an effective strategy for enabling teams to learn from one another (within and across projects), although there were some problems in its usability. The team workspace should be further developed in ESAOA KMS version 2 to encourage effective knowledge sharing (see Sections 6.3.7 and 6.4.4).

5.8.6 Towards ESAOA version 2

The refinements to ESAOA version 1 listed in Sections 5.8.1 to 5.8.5 are addressed in ESAOA KMS version 2, which is discussed in the next chapter (Chapter 6).

University of Cape Town

University of Cape Town

Chapter 6:

ESAOA KMS version 2

This chapter presents the second version of the ESAOA KMS and constitutes the second iteration of the framework construction phase described in the methodology (Chapter 3). This chapter follows a similar layout to that of Section 4.6 in Chapter 4, in which the first version of the ESAOA KMS was described. Sections of this chapter highlight major changes that were applied to the first version of the KMS to establish the second version, including the reasons for these changes that were based on results of Experiment 2. This chapter also presents findings concerning knowledge worker roles because the responsibility assigned to roles, the artefacts that they used, and the collaboration that occurred between roles, all had a significant influence on the KM tools and KM practices used in a project. Consequently, technical tool design and knowledge worker roles are presented in this Chapter.

This chapter starts, in Section 6.1, with an overview of ESAOA KMS version 2 in which it is explained that the high-level structure of the KMS is separated into technical processes, artefacts, and roles. The way in which these parts are integrated using ESAOA workspaces is explained in this section.

The ESAOA knowledge ontology is summarised in Section 6.2; this defines the specialised terminology structure for the ESAOA KMS that is used in subsequent sections of this chapter.

The design and implementation of ESAOA version 2 workspaces is the focus of Section 6.3. This section highlights the new features that were added to the first version of the ESAOA workspaces. This section includes issues related to ESAOA workstations, which are used to access workspaces, and the use of ESAOA distributions in installing ESAOA workspaces.

The information contained in the preceding sections (i.e., the ESAOA knowledge ontology and ESAOA workspaces) is drawn on in Sections 6.4 to 6.7 when detailing

University of Cape Town

the design and implementation of ESAOA KMS version 2. More specifically, Section 6.4 refines the collection of ESAOA support tools. Section 6.5 provides details concerning roles and role support structures. Section 6.6 describes processes of ESAOA version 2 that are performed by the roles, and lastly, Section 6.7 introduces a collection of artefacts, and artefact classifications, used in the KMS.

6.1 Overview of ESAOA KMS version 2

The workspace-based design was found to be an effective means to implement version 1 of the ESAOA KMS, which was tested in Experiment 2. Consequently, the workspace-based approach has been retained as the underlying design approach for version 2 of the ESAOA KMS. Furthermore, the support tools, integrated knowledge base and artefact classification techniques used in the second version of the ESAOA KMS are revised versions of those used in the first version. Revisions, based on the findings of Experiment 2, have been applied to the roles, processes and ESAOA knowledge ontology of ESAOA KMS version 2. The initial collection and organisation of the communal and team artefacts, which are the starting point for new installations of the KMS, have been revised and the *ESAOA* distributions modified accordingly.

The development of the ESAOA workspaces for ESAOA KMS version 2, which comprised the second iteration of the framework construction step described in Section 3.5, followed a similar approach to that used for the construction of ESAOA KMS version 1. Consequently, soft artefacts were taken from Experiment 2 teams, and these artefacts were supplemented and improved upon for incorporation into ESAOA version 2 workspaces. Changes were made to ESAOA workspace and workstation design, and to the KMS operation in response to Experiment 2 results.

Although the first and second versions of the KMS utilised the same workspace-based approach, the designs of the two versions differ in a variety of ways. Most importantly, the second version of the KMS is more extensive than the first version (Section 4.6 describes the first version). This chapter progresses through the design of each aspect of ESAOA KMS version 2, highlighting changes made in relation to version 1 and the reasons for these changes. ESAOA models are used to illustrate the operation and structure of ESAOA KMS version 2, as was done in Chapter 4.

The subsections below outline major changes made to version 1 of the ESAOA KMS in the course of constructing version 2. Section 6.1.1 starts by recapping the types of

University of Cape Town

ESAOA workspaces used in the ESAOA KMS, and the notion of an ESAOA workstation. Section 6.1.2 summarises changes made to ESAOA roles and role support structures. Section 6.1.3 reviews roles and artefact classifications for ESAOA version 2. Section 6.1.4 describes how the ESAOA support tools were upgraded, and Section 6.1.5 concerns the additions to the documentation.

6.1.1 Use of ESAOA workspaces and workstations

The second version of the ESAOA KMS utilises the same three workspaces as the first version does, although each workspace has been extended. As before, each workspace comprises an ESAOA shell environment, a directory structure, an integrated knowledge base, and a collection of ESAOA support tools. Each workspace also depends on a collection of externally stored software programs (such as cross compilers and text editors) that the user needs to be able to access in order to view or work on artefacts in the workspace concerned.

Synchronization scripts have been added to version 2 of the ESAOA KMS support tools, which automate the process of synchronizing a team member's personal workspace with the team member's corresponding team workspace. Version 2 workspaces support multiple projects folders in a personal workspace that are built by different teams (but a new team workspace needs to be created if there is no existing team workspace with the same set of team members defined). Personal workspaces can consequently contain links to multiple team workspaces. See Section 6.3 for detail concerning ESAOA version 2 workspaces and workstations.

6.1.2 Changes to roles and role support structures

The design of ESAOA KMS version 1 followed a predominantly *feed-forward* flow of knowledge. This approach was based on the Experiment 1 observations, in which innovation was seen to happen when effective process knowledge was available, and such knowledge in turn required effective data knowledge. Consequently, the three team member roles established in ESAOA KMS version 1 followed a hierarchical structure that focused on supporting the innovation engineer (IE). The IE was thus at the top of this hierarchy, the process engineer (PE) one level down, and the data steward (DS) at the bottom. The chief knowledge officer (CKO) and communal knowledge steward (CKS) supported and advised the other roles.

When ESAOA KMS version 1 was applied in Experiment 2, it revealed both potential advantages, and shortcomings, in this hierarchical feed-forward KMS design.

University of Cape Town

6.1.2.1 Reducing priority of innovation and flattening the role hierarchy

The Experiment 2 results indicated that the hierarchical approach of ESAOA KMS version 1 had certain advantages. For example, teams that produced higher proportions of innovation knowledge often obtained higher prototype ratings (see Section 5.7.4.1). However, correlations in Section 5.5 also showed that focusing narrowly on innovation, and vigorously driving innovation knowledge, can cause detrimental omissions in other parts of a project (such as producing less process and data knowledge, thus leading to experiments being incomplete or product requirements left dissatisfied – Project P2-7 is an example). The design of ESAOA KMS version 2 compensates for these risks by reducing (but *not* eliminating) the emphasis on innovation knowledge, thus preventing teams becoming overly engrossed in driving innovation. Accordingly, the hierarchical role structure of version 1 has been flattened in the second version so that no particular role is considered less important than any other role.

ESAOA KMS version 1 was based on a premise that each role was expected to produce the largest portion of a certain type of knowledge; for example, the role of the DS was to produce the most data knowledge, the PE the most process knowledge, and so on. But Experiment 2 showed that this expectation was inaccurate. For example, while the results in Section 5.7.7 generally showed that the PE contributed most of the process knowledge and the IE contributed most of the innovation knowledge, both also contribute large portions of data knowledge. The difference between the expected types of knowledge produced per role, and the actual types of knowledge produced per role, was especially noticeable in Project P2-2, where 53% of the project's data knowledge was produced by the PE and IE (*without* DS involvement). Consequently, the roles of ESAOA KMS version 2 have been changed to establish a better correspondence between the expected behaviour of roles (and the knowledge that they produced) and the actual behaviour of the roles carried out during a development project.

6.1.2.2 Towards a bi-directional flow of innovation

In Experiment 2, it was observed that the KMS used by the more successful project teams followed a bi-directional flow of knowledge (Section 5.7.6 discusses the analysis of data in which this observation was made). Innovation still drove process knowledge and indirectly data knowledge too. However, there was also a flow in the opposite direction, predominately between the DS and IE (often without involving the PE) – this flow concerned enhancement of innovation techniques and knowledge

University of Cape Town

provided by the DS. This effect was most noticeable in projects P2-1 and P2-5, in which the DS thought of innovative ideas for the IE to test, while simultaneously acquiring and understanding data knowledge needed by the PE. Consequently, the predominately feed-forward flow of the first version of the KMS was found to be an oversimplification. For this reason, changes were made to the role support structures (i.e., the DS, PE, IE hierarchy), to accommodate a more bi-directional flow of innovation knowledge. These changes involved eliminating the DS role and incorporating two new roles into ESAOA KMS version 2 (see Section 6.1.3).

Experiment 2 observations showed that the creation of process knowledge also inspired and guided innovation; an example of this occurred when activities of the PE sometimes directed decisions relating to innovation. But this situation occurred less frequently than situations where the IE or DS led innovation (show in Section 5.7.7). The design of ESAOA KMS version 2 was therefore changed to reflect the predominant flow of innovation from DS to IE, and from PE to IE, with more emphasis given on the acquisition of data knowledge inspiring innovation. The IE role thus retains responsibility for testing innovations, and making decisions on what to test.

In Experiment 2, team members performing the DS role considered the role title inappropriate (see comments in Section 5.6.2), and indicated that the role appeared unimportant compared to the other roles, and that it was dominated by the PE and IE roles. These results suggest that the DS role was misconstrued as an assistant role, where the individuals performing the DS role were expected to receive instructions from the PE and IE – but this issue was generally expressed informally by the participants, and only one team member considered the point important enough to leave a written comment in the Experiment 2 survey. Further Experiment 2 findings (shown in Section 5.7.8) indicated that the DS role tended to concentrate on researching components (especially datasheets for electronic components and information about device drivers), and acquired comparatively little data knowledge related to tools. Both the DS and PE roles produced data knowledge, but these roles generally performed different types of information-seeking tasks. Principally, the PE used more forum postings, sample code, and manuals to build data knowledge that related to development tools and operating systems. The DS mainly used manuals and web-based documentation to gather data knowledge. These findings provide further rationale for changing the roles and data knowledge management strategy used in KMS version 1.

University of Cape Town

6.1.3 Revised roles and artefact classifications

The roles and role support structures of ESAOA KMS version 2 have been structured according to the reasons discussed above. Hence, roles of ESAOA KMS version 2 support a bi-directional flow of innovation, specifically by ensuring that all roles can contribute innovation knowledge, although the building of this knowledge is still largely driven by the IE. The revised system also avoids detracting from other forms of knowledge production that may weaken the overall success of a project. The changes to the ESAOA roles are explained below, and are expected largely to eliminate the issue of role dominance mentioned earlier.

6.1.3.1 Component researcher (CR) and workspace administrator (WA)

Given the inadequacies of the DS role as observed in Section 6.1.2 above, the DS role was removed for the second version of the ESAOA KMS and replaced by two new roles. These new roles are the *component researcher* (CR) and the *workspace administrator* (WA).

Activities of the former DS role that relate to researching components have been assigned to the new CR role. The term *DS artefact* (previously used to classify artefacts as being maintained by the DS) has been replaced with the term *D artefact* (i.e., data knowledge artefact – see Table 6.8 for examples of D artefacts).

The new WA role is responsible for maintaining the consistency of artefacts in the team workspace as a whole. In version 1 of the KMS, the maintenance of the team workspace was to be divided between the various roles, where each role is responsible for artefacts according to their role's applied classification (e.g., the DS maintaining only DS artefacts). However, in Experiment 2 developers tended not to assign role classification (probably because this task had to be done manually and was easily forgotten). Among the Experiment 2 teams that used version control software (such as CVS [Grune, 2007]), the maintenance of team artefacts became mostly automatic. The WA role is responsible for maintaining and understanding the structure of team workspaces. Further explanations of the CR and WA roles are given in Section 6.5.1, and the main processes they carry out are described in Sections 6.6.4 and 6.6.6.

6.1.3.2 Revisions to the PE and IE roles

The PE role was modified to account for the production of data knowledge and to avoid the hierarchical-type structure that resulted in Experiment 2. Changes to

University of Cape Town

processes of the PE are intended to place the CR and PE at the same level of authority. By giving the WA a more accurate title, as well as authority over the structure of the team workspace, it is expected that an individual filling the WA role will also not feel dominated by other roles. Processes and artefacts classifications used by the PE and IE roles were modified to accommodate the new roles and support structures.

The revised role support structure for ESAOA KMS version 2 is illustrated in Figure 6.1. The figure shows that the CR supports the PE, and the PE supports the IE (represented by the *role support associations,* i.e., lines that end in a solid circle). It is expected that the PE will build most of the project's process knowledge, whereas the CR will build most of the data knowledge related to components, and the IE will build most of the *productive* innovation knowledge used for prototyping concepts. The *knowledge use* associations model this behaviour, and these are shown as single line arrows linking a role circle to a *knowledge atom* in Figure 6.1 – the knowledge atoms are the blocks of three rectangles. The support structure also accounts for both the CR and the PE providing innovation knowledge (shown in Figure 6.1 as *mentoring flows*, the double line arrows, in which *tacit junctions* are associated with *innovation knowledge atoms*). The model further shows that the CR is expected to obtain innovation knowledge mainly related to components, whereas the PE is intended to supply innovation knowledge chiefly related to development procedures.

Although version 2 of the ESAOA KMS has more roles than version 1, this does not imply that version 2 teams need to be have more people in them. A particular individual can fill more than one role (as was the case in version 1). Thus, the KMS can still be run by teams of three members. For example, the same person could fill both the PE and WA roles. However, certain combinations of roles performed by one person can be ineffective, causing project work to be unfairly shared. For instance, in a three-person team, an individual filling both the CR and PE roles would likely do considerably more work than the other members would.

University of Cape Town

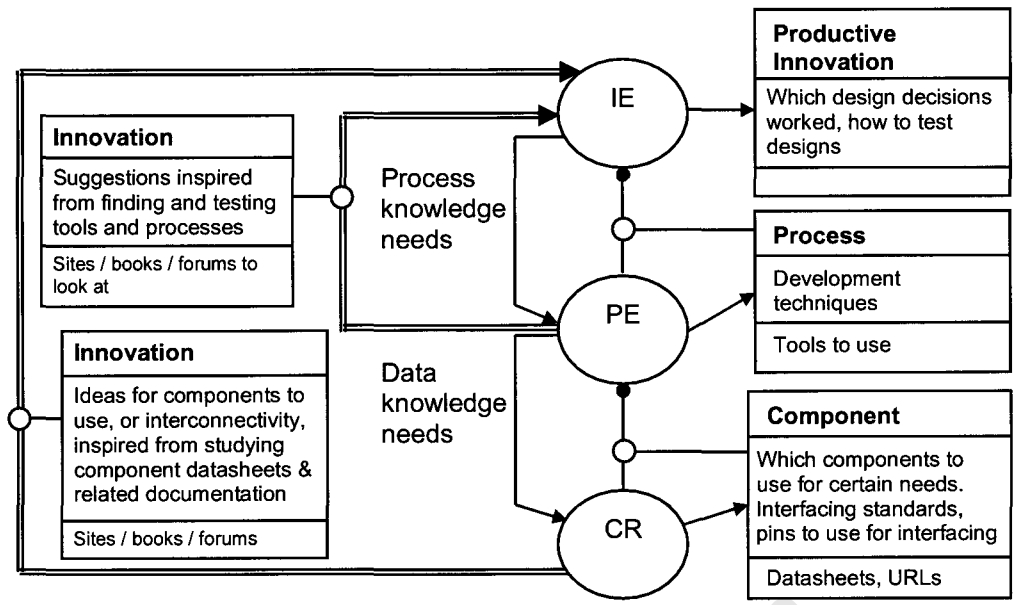


Figure 6.1: The role support structure and flow of knowledge around which ESAOA KMS version 2 is designed.

6.1.3.3 Revision to the TL role

The team leader (TL) role is responsible for the managerial duties of a team, for example ensuring that all team members are performing their duties. No processes were specified for the TL in ESAOA KMS version 1; but certain TL processes related to the performance assessment of ESAOA activities have been added to version 2 of the KMS. Section 6.5.2 elaborates on TL responsibilities and Section 6.6.3 describes processes of the TL.

6.1.3.4 Role extension

Role extension involves adding processes and responsibilities to a role, or moving responsibility from one role to a different role. An example of role extension, applied to the PE role, is to make the PE maintain all manuals related to development tools that reside in the team workspace (in effect this involves moving some of the responsibility of the WA, who was initially tasked to maintain the team workspace, over to the PE).

Role extension occurred in Experiment 2 (see Section 5.8.2). Project P2-1 included a number of successful instances of role extension, such as when the PE relieved the DS from the responsibility of acquiring data knowledge relating to tools, which then enabled the DS to give additional support to the IE.

University of Cape Town

Project P2-2 included some examples of less effective role extension. For instance, Section 5.5.6 indicated a case where the IE took over most of the PE's duties, which resulted in misdirected effort and weakened the overall effectiveness of the team.

Based on the role extensions observed in Experiment 2, limits clearly need to be imposed on role extensions to ensure that no team member is burdened with too many responsibilities. The TL, possibly in consultation with the CKO, should approve role extensions to ensure that responsibilities are shared fairly and suitably (as is emphasised in the project management literature, such as Turner *et al.* [2008]).

6.1.4 Upgrading of support tools

Experiment 2 participants found that a number of the ESAOA tools were beneficial to their development projects and KM needs, whereas others were infrequently used or not known about (Section 5.6 summarises the results of the relevant evaluations). Shortcomings, such as lacking tool support and having ineffective tools, were also identified. The provision and operation of ESAOA version 2 support tools have thus been modified to address many of these limitations. The extent of these modifications was restricted by the time limits of this research project; therefore, not all of the tool design changes have been implemented at the time of writing this thesis. Section 6.4 details the tool design changes, and includes screen captures to illustrate the parts of their implementation that have been completed.

6.1.5 Improving ESAOA documentation

Participants in Experiment 2 found the ESAOA tools had a number of shortcomings in terms of documentation and access to documentation. ESAOA KMS version 2 addresses these shortcomings by improving the help integrated into the ESAOA support tools, in addition to the provision of a comprehensive user manual. Section 6.4 highlights additions made to the documentation.

6.2 ESAOA knowledge ontology

A KMS typically incorporates a knowledge ontology in addition to roles, processes and artefacts. The concept of an ontology for facilitating the reuse and exchange of technical knowledge was discussed in Section 2.8.3.2. As mentioned in Section 4.6.3, the term *ESAOA knowledge ontology* refers to a specialised terminology structure used to specify the ESAOA KMS. The knowledge ontology for ESAOA KMS version 2 has the same structure as that of version 1 (see Section 4.6.3). This

University of Cape Town

section begins by describing additions incorporated into the second version of the ESAOA knowledge ontology, and then explains how the ontology evolved.

6.2.1 Additions to the ESAOA knowledge ontology

ESAOA KMS version 2 contains additional terms to those found in version 1. Two additional top-level terms were added, namely:

- **Classifications:** this refers to the way in which an artefact is categorised. Types of classifications include *form classifications*, *functionality classifications*, *workspace classifications*, and *maintainer classifications*.
- **Episode:** this refers to a sequence of related chronologically close activities, within a larger sequence of activities, leading to a final development solution.

The UML model depicted in Figure 6.2 visualizes part of the higher level ESAOA knowledge ontology. The figure shows that an *episode* is essentially an aggregation of activities. An *artefact classification* is a specialised type of *classification* applied to soft artefacts within an ESAOA workspace, and it is used to identify artefacts.

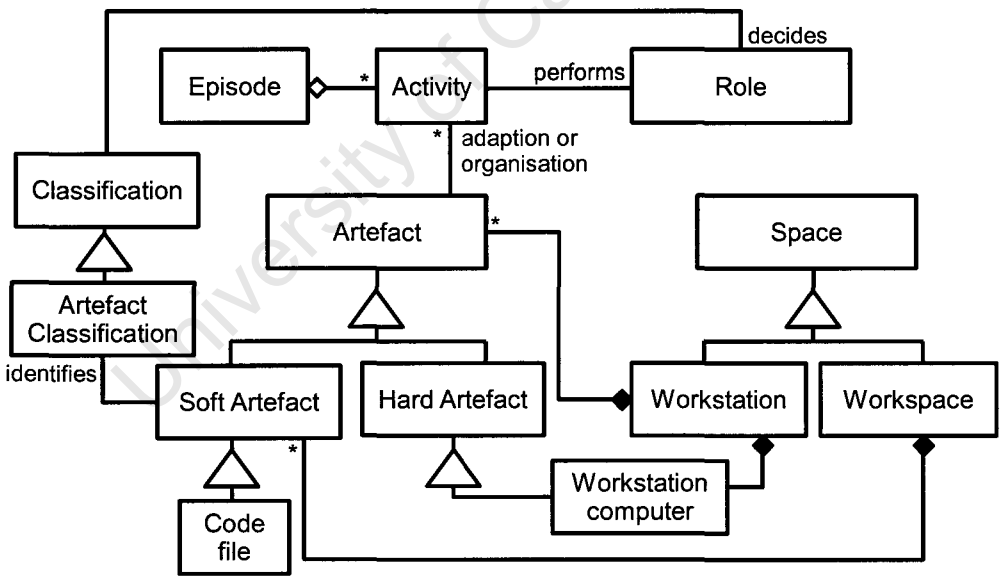


Figure 6.2: UML model for part of the high level ESAOA knowledge ontology.

The complete list of terms used in the ontology, their definitions, and how they fit into the ontology itself, is detailed in Appendix C.1 (additions for version 2 are highlighted by asterisks).

University of Cape Town

6.2.2 Evolving the ESAOA knowledge ontology

The ESAOA knowledge ontology is separated into two parts: a higher level and a lower level. The lower level is adjusted depending on project needs, whereas the higher level remains largely the same between different teams and projects. The higher and lower parts of the ontology are maintained using separate procedures: the higher level is represented as a Microsoft Word document that is edited manually, whereas the lower level is in the form of metafiles used to classify soft artefacts in workspaces.

6.2.2.1 Maintaining the lower level of the knowledge ontology

The lower level is by using the *fclass* program (one of the ESAOA tools). Experiment 2 participants found that *fclass* was an effective means to maintain artefact classifications; consequently, an upgraded version of the same tools is used for maintaining file classifications in version 2 of the ESAOA KMS.

Experiment 2 participants did not suggest changes to the operation of the *fclass* tool, but they did ask for the documentation to be improved. In order to address this request, online help has been added (both help messages are displayed by the program and a PDF help document). The new version of *fclass* is described in Section 6.4.1.

6.2.2.2 Maintaining the upper level of the knowledge ontology

Participants were dissatisfied with the cumbersome way in which the higher-level part of the ontology was maintained using text documents. Consequently, version 2 of the KMS is designed to use a custom CGI web-based program to manage the high level part of the knowledge ontology. The tool is referred to as the *ESAOA Ontology Manager* (or OM tool). Figure 6.3 provides a screenshot of the tool. It shows the main page of the online OM tool, which users use to specify their team, and thereafter choose to view or add terms to their team's ontology.

University of Cape Town

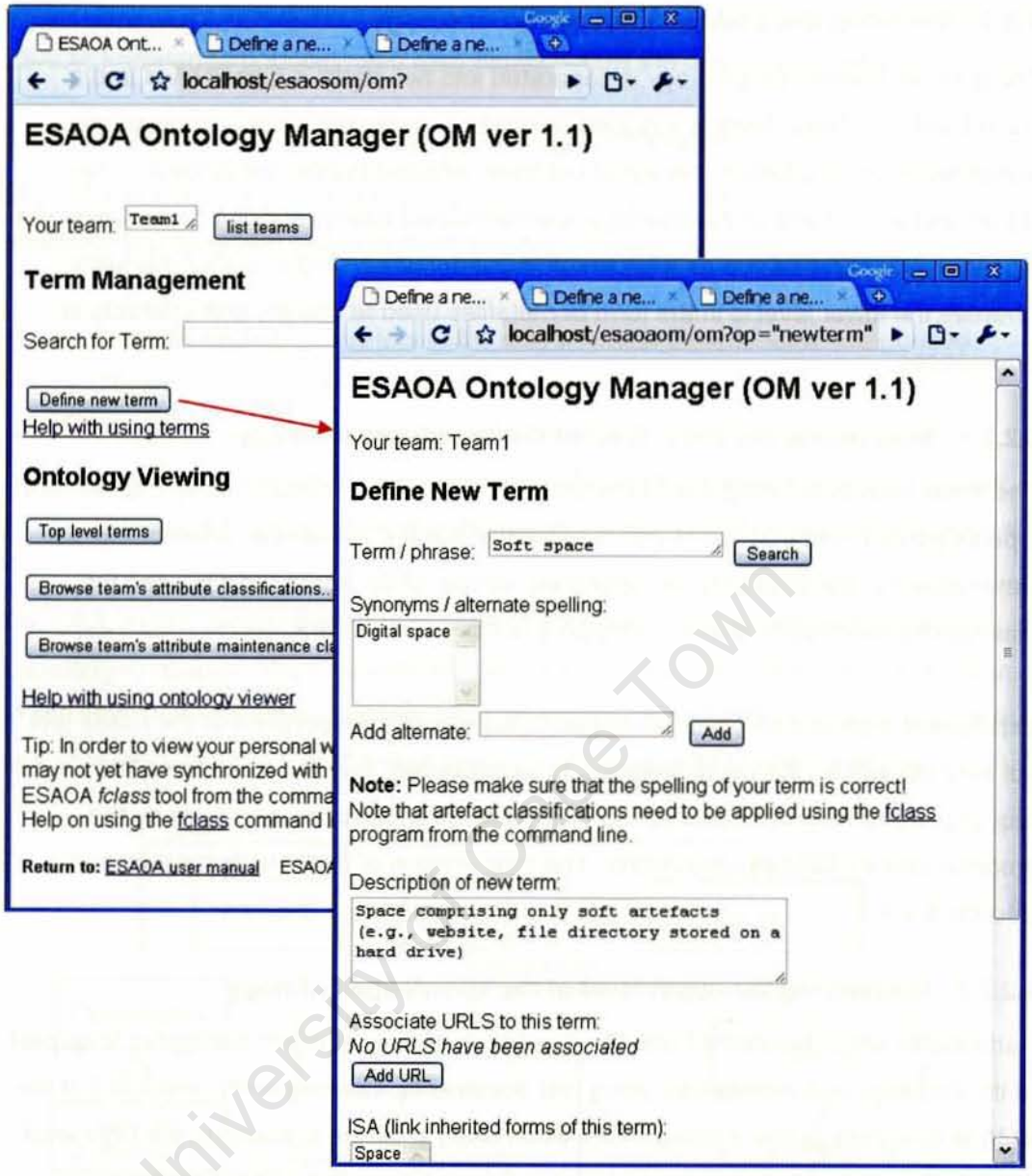


Figure 6.3: Screenshots from ESAOA ontology manager.

The OM tool is currently in the form of a limited functionality prototype system that has been used in a pilot project to test the OM tool[1] [Winberg, 2009]. This pilot test involved a small development team that tested the planned OM tool. The results showed that the OM tool is likely to be less effective than simply using a text document to maintain the ontology. The developers that experimented with the prototype reported it was "too time consuming to use...", "...had too much clicking through menus" and commented that "documentation [is] minimal" [Winberg, 2009,

[1] The prototype OM tool has no security features, implying that any user can view and modify a different team's ontology. There is also no search facility, but this limitation can be overcome by requesting a list of all categories.

University of Cape Town

pg. 5]. These participants suggested using Excel document instead, especially as this tool can be set to check spelling, and auto-complete words typed in – however, considering that Microsoft Word provides the same features mentioned by the participants, the researcher has decided that there is little point in changing the format of the files used to store the high level ontology. Consequently, for version 2 of the KMS, the higher level terms will continue to be stored in Microsoft Word files.

6.3 ESAOA version 2 workspaces

ESAOA workspaces unite the aspects (i.e., roles, processes and artefacts aspects) of the ESAOA KMS in order to present knowledge workers with a defined and usable system. Users of the ESAOA KMS access ESAOA workspaces via ESAOA workstations. ESAOA KMS version 2 uses an extended version of the ESAOA version 1 workspace design – the most significant addition is the refinement made to the integrated knowledge base.

This section begins by recapping the definition of ESAOA workspaces and workstations. The subsections that follow elaborate on the design and implementation of ESAOA KMS version 2 workspaces and workstations.

6.3.1 Definition of an ESAOA workspace

An ESAOA workspace is a digital, computer-based work area that comprises the following parts:

1. A shell environment (an extended version of the Bash shell [Bash, 2009]);
2. Soft artefacts (i.e., computer files) organised into an ESAOA directory structure (following a specific layout and classification schema);
3. An integrated knowledge base;
4. ESAOA support tools; and
5. A related collection of externally stored and maintained development tools (e.g., compilers and CAD software).

Multiple workspaces are implemented in the ESAOA KMS as a means to facilitate sharing between teams, by having the same access, organisation and classification systems operating at the different levels of access. This scheme also avoids inadvertent sharing of artefacts between teams by having procedures in place for releasing artefacts to all teams, or to a particular individual's own team. In both

University of Cape Town

versions 1 and 2 of the ESAOA KMS, there are three types of ESAOA workspaces to accommodate this sharing approach; these are summarised below:

- *Communal workspace:* contains soft artefacts accessible to any team members. There is only one communal workspace in the ESAOA KMS. The CKO and CKS are responsible for maintaining it.
- *Team workspace:* each team has a team workspace that is accessible only to members of that particular team. The team workspace contains master versions of soft artefacts worked on by a team, and it is used to share artefacts among team members. The WA is responsible for maintaining the consistency and placement of artefacts in team workspaces.
- *Personal workspace:* each user of the ESAOA KMS has a personal workspace. The personal workspace of a team member tends to be a copy of the member's team workspace. The CKO and CKS are likely to modify the communal workspace directly, when arranging items or naming files – but each of these roles also has a personal workspace for working on soft artefacts taken from, or destined for, the communal workspace. Each team member (in collaboration with the WA) is responsible for maintaining the consistency of artefacts that they are working on.

6.3.2 Definition of an ESAOA workstation

The term *ESAOA workstation* refers to the combination of a computer system (termed the *workstation computer*), which provides the human/computer interface to an ESAOA workspace, together with the surrounding physical artefacts that developers use during the development of an ES (e.g., datasheet printouts, books, and the ES prototype being worked on). Essentially, an ESAOA workstation is much the same as any normal work area with a computer that is used during ES development. The term ESAOA workstation is used in relation to the ESAOA KMS simply to emphasise that a particular workstation is intended for use with an ESAOA workspace. Section 4.6.2.2 provides further detail and illustrations of ESAOA workstations.

Only minor changes were made to the structure of ESAOA workstations between versions 1 and 2 of the ESAOA KMS. The most significant changes includes the provision of workstation-side scripts (described in Section 6.4.5) and the facility to use a web browser to access the *Ontology Manager* (see Section 6.2.2.2) to change the communal ontology.

University of Cape Town

6.3.3 ESAOA workspaces implementation and access levels

Each workspace, be it a communal, team or personal workspace, has the same top-level part structure as described in Section 6.3.1. The UML model shown in Figure 6.4 models the composition of an ESAOA workspace. As the model shows, it comprises a directory structure that contains the soft artefacts of a project. Development tools, such as cross-compilers and integrated development environments, and other software tools used during development are not stored within the ESAOA directory structure; rather, these programs (which tend to take up a large amount of disk space) are stored and maintained externally (note that the Bash shell itself, on which the ESAOA environment depends, is categorised as external software). ESAOA support tools (described in Section 4.6.7.1) are stored within the ESAOA directory structure.

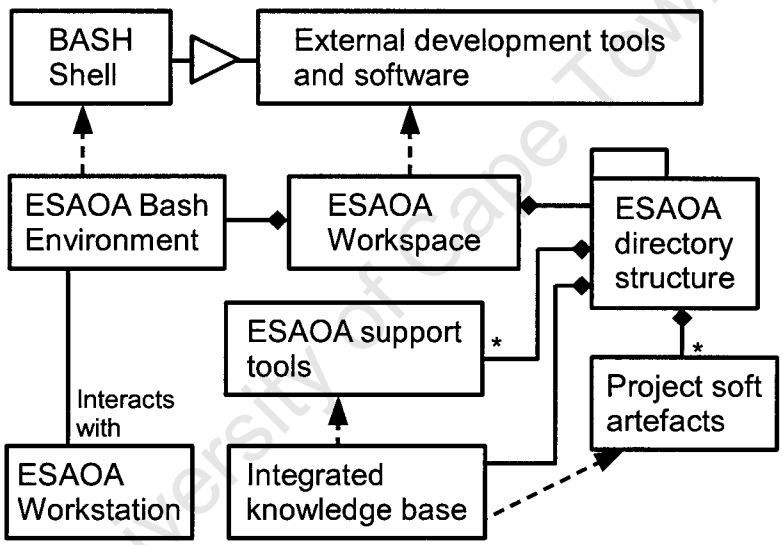


Figure 6.4: Model showing composition of an ESAOA workspace.

The ESAOA Bash environment (detailed in Section 4.6.2.1) involves starting a Bash shell that automatically configures various environment variables to modify the behaviour of the shell. The shell is configured to work with the different workspaces, and to facilitate use of the ESAOA KMS and access to ESAOA tools. As Figure 6.4 shows, the integrated knowledge base of an ESAOA workspace is stored within the ESAOA directory structure. This knowledge base is composed of the soft artefacts in the workspace, the soft artefact classifications and other 'metadata' [Kitamura *et al.*, 2005], maintained or viewed by ESAOA tools. The metadata is stored in comma separated value (CSV) files in the workspaces (see Section 6.4.1). ESAOA tools, such as the *flcass* and *esaoa-find* tool, provide the interface to the knowledge base.

University of Cape Town

Personal workspaces can have links to team workspaces, and similarly, team workspaces can have links to the communal workspace. Personal workspaces are synchronized with team workspaces, either using version control tools or supplied ESAOA scripts. Only certain sections of a team workspace are synchronized with a communal workspace – these synchronization activities are done manually.

The model in Figure 6.5 illustrates which roles generally perform the maintenance of artefacts in the different workspaces. As shown in the figure, the CKO and CKS generally work on the communal workspace. The WA checks and corrects the consistency of artefacts in his or her team workspace. The other roles generally work on artefacts in their own personal workspace.

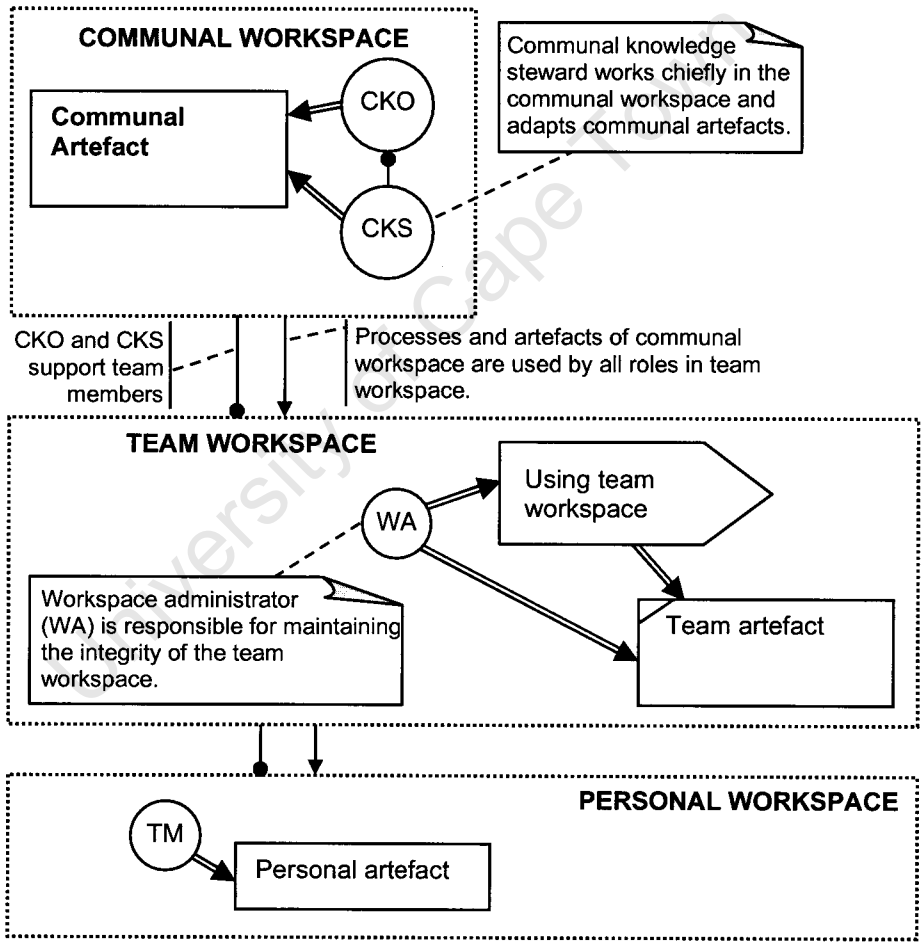


Figure 6.5: The three types of ESAOA workspace.

The different types of workspace can also be considered as being at different *access levels*. Three access levels are defined, namely: a) communal level; b) team level; and c) personal level. Communal level access defines the file access permissions a user places on artefacts in the communal workspace. Similarly, team level and

personal level relate to file access permissions for team and personal workspaces respectively.

All users of the ESAOA KMS are by default given communal level read only access (i.e., they can read any file in the communal workspace, but cannot modify, delete or move any files in the workspace). The CKO and CKS are given full communal level access (i.e., they can read, write, delete and move files in the communal workspace). Each team member is by default given full team level access, as well as full personal level access to his or her own personal workspace. The WA, in collaboration with fellow team members, may decide to impose certain team level restrictions (e.g., to avoid important documents being inadvertently deleted). A new ESAOA support script, called *esaoa-access*, has been added to facilitate the WA's responsibility of specifying team level access (the script is intended only for Linux-based systems).

6.3.4 Installing workspaces using ESAOA distributions

ESAOA workspaces are put in place using ESAOA distributions. These distributions are compressed archives that include soft artefacts organised into an ESAOA directory structure (Section 4.6.8 gives more detail). ESAOA version 1 had three distributions: 1) a workstation distribution (intended as a convenient collection of feely available software programs that are installed on a workspace computer); 2) a team distribution; and 3) a communal distribution. ESAOA version 2 has the same three types of distribution, and an additional personal workspace distribution.

While the installation of distributions for version 1 was mainly done manually (with only the team distribution having an installation script), version 2 has automated installation scripts for each distribution. These installation scripts automatically perform operations to configure the Bash shell, to create ESAOA baseline directory structures, and to copy files to the appropriate destination folders. These automated installation scripts are intended to make it much easier to install and use ESAOA workspaces; these additions were made to address the installation difficulties reported in Section 5.8.1.

The four types of ESAOA distribution are summarised in Table 6.1. ESAOA support tools are provided only in the communal distribution – the other workspaces automatically inherit all ESAOA tools and scripts in the communal workspace (this is achieved using the PATH environment variable). Additional tools and scripts can be added to workspaces by placing them into relevant locations of a workspace. All the

University of Cape Town

baseline ESAOA support tools are in the form of either Bash scripts or compiled executable programs (implemented using the Standard C library [GNU, 2008a] and the KIT API [Winberg, 2006a] – the KIT API is described in Appendix C.4). Team members can implement additional support tools using other languages or libraries (e.g., Python scripts [Lutz, 2006]) – but doing so can make the tools less portable between ESAOA workspaces.

Each ESAOA workspace distribution contains template files, an initial workspace knowledge base, ESAOA scripts and ESAOA tools that form a baseline workspace. Baseline workspaces on their own cannot be used to construct design artefacts and compile code – the workspaces depend on a variety of software applications, termed *external tools*, which are not included in the ESAOA distributions, namely: the Bash command shell, GNU bintools [GNU, 2005], and appropriately configured cross-compilers (e.g., [Kegel, 2007]), which are needed to compile embedded software. All these external tools need to be installed on the computers that host an ESAOA workspace (Bash and GNU bintools need to be installed prior to installing workspaces; but the cross-compilers, IDEs and other external software can be installed later). Similarly, software needs to be installed on the workstation computers in order to access ESAOA workspaces.

University of Cape Town

Table 6.1: ESAOA version 2 distributions.

ESAOA Distribution	Archive file	Contents
ESAOA communal workspace distribution	ESAOA-communal-2.0.tar.gz	Baseline for communal distribution and templates of communal soft artefacts.
ESAOA team workspace distribution	ESAOA-team-2.0.tar.gz	A baseline collection of soft artefacts provides a starting point for developer teams.
ESAOA personal workspace distribution	ESAOA-personal-2.0.tar.gz	A small collection of soft artefacts, and an installation script to assist the user in connecting to a team workspace.
ESAOA workstation distribution	ESAOA-workstation-2.0.tar.gz	A convenient collection of freely available software applications that facilitate access to ESAOA workspaces and are likely to be used during development.

6.3.5 GUI installation tool for ESAOA personal workspaces

The installation scripts for the ESAOA workspaces are in the form of Bash scripts [Bash, 2009]. Since the personal workspaces install scripts are likely to be the most commonly used (i.e., run by each team member), a graphic user interface (GUI) script was designed to facilitate installation of personal workspaces. The Tk/Tcl

scripting and GUI toolkit [Tcl Developer Site, 2007] was used to do so. A Tcl script was built that calls the personal workspace installation Bash script using suitable command line parameters. Figure 6.6 shows a screen shot of running the GUI personal workstation installation script using ActiveTcl [ActiveState, 2007].



Figure 6.6: Screenshot of prototype personal workstation installation program.

6.3.6 ESAOA version 2 distribution support documentation

During Experiment 2, some of team members favoured a decentralised approach in which they independently installed and maintained ESAOA distributions on their own computers. The GUI installation tool described in Section 6.3.5 is intended to assist with the process of installing ESAOA workspaces (but this tool does not provide support for installing workstation programs, nor is it guaranteed to work on all Linux distributions). The following additions to the ESAOA version 2 documentation are planned (these are based on the survey results of Section 5.6.2):

- *ESAOA workspace quick start guide:* a short step-by-step guide for installing the team workspace distribution with an introduction to using the ESAOA workspace and its integrated knowledge base.
- *ESAOA script writers guide:* a document (mainly for the PE) that explains the structure of ESAOA scripts and simple steps for writing scripts.

University of Cape Town

- *ESAOA workspace maintenance methods:* a document that details the administration of ESAOA workspaces, describing organisational layouts and how to administer the knowledge base stored within these workspaces.
- *ESAOA workspace developer's guide:* a document to facilitate modification and development of ESAOA workspaces and ESAOA support tools.

6.3.7 ESAOA version 2 workspace directory structures

The directory structures of the ESAOA version 2 workspaces, which are put in place using the ESAOA distributions, remained largely the same as in the first version (see Section 4.6.5.4). Most of the changes were made to the directory structure of the team workspace (and additional documentation and template artefacts were added to the communal workspace). Changes to the team distribution are described below.

In Experiment 2, many teams added data files that captured test cases or test procedures (e.g., data files and scripts that automated regression tests); in some cases, these testing files were placed in a separate directory. Consequently, version 2 of the baseline team workspace provides test cases for the sample applications included (specifically, test cases were added to the sample application called 'SimpleApp'). Two additional sample applications were included in the team distribution, namely: 'BitBanger' and 'HostSer'. The BitBanger application (taken from Vibynet, Project P2-3) is a more elaborate example than SimpleApp (which was the only sample application in version 1 of the team workspace). The HostSer application executes on a PC, and implements simple serial communications (HostSer is a refined version of the 'PCControl' application initially developed by the HAS team, Project P2-5). Furthermore, additional platform deployment modules (PDM modules) were added to the team workspace (these modules provide some commonly used software/hardware interfacing routines). Figure 6.7 shows the revised directory structure of the team workspace (asterisks in the figure mark additions to the structure in comparison to the first version).

University of Cape Town

```
+-ProjectX/ (ProjectX initial team workspace ver2)  ← project root
  +-Software/                ← Root for software artefacts
  | +-Templates/             ← Templates directory
  |   +-Communal/            ← Soft link to communal templates
  |   +-Team/                ← Templates produced by team
  | +-Applications/          ← Application code modules
  | | +-SimpleApp/           ← Application code directory containing code for SimpleApp
  | | | +-Testing/           ← Test cases / descriptions for testing SimpleApp *
  | | +-BitBanger/           ← Application code directory containing code for BitBanger *
  | | | +-Testing/           ← Test cases / descriptions for testing BitBanger *
  | | +-HostSer/             ← Application code directory for host-side serial comms.*
  | |   +-Testing/           ← Test cases / descriptions for testing HostSer *
  | +-Utils/                 ← Directory for utility code modules
  | +-CCW/                   ← Directory "common component wrappers" (CCWs)
  | +-PDM/                   ← Directory "platform deployment modules" (PDMs)
  | | +-CSB337/              ← Directory containing PDMs for CSB337 platform
  | | | +-MicroMonitor/      ← PDMs for MicroMonitor variant of CSB337 platform
  | | | +-uCLinux/           ← PDMs for uCLinux variant of CSB337 platform
  | | +-PC/                  ← Directory containing PDMs for PC architecture *
  | |   +-Linux/             ← Platform deployment modules for PC version of Linux *
  | +-Documentation/         ← Software documentation
  |   +-Communal/            ← Soft link to communal software documentation
  |     +-MicroMonitor/      ← Documentation for the MicroMonitor O/S
  |     +-uCLinux/           ← Documentation for the uCLinux O/S
  |     | +-MemDevInfo/        and related tools / drivers
  |     +-gnu-assembler/     ← Documentation for the Gnu assembler
  |   +-Team/                ← Supplementary documentation obtained by team
  +-Tools/                   ← Supplementary tools developed by team
  | +-Scripts/               ← Supplementary ESAOA Bash scripts
  | +-Programs/              ← Supplementary ESAOA C++ programs
  +-Hardware/                ← root for artefacts most strongly related to hardware
  | +-Documentation          ← Hardware documentation
  |   +-Communal/            ← Communal hardware documentation
  |     +-Datasheets/        ← Component datasheets
  |     | +-CSB337/          ← Directory for specific component uses the component's ID
  |     | +-AT91RM9200/
  |     | +-Peripherals/     ← Directory for various peripherals (subdirectories not shown)
  |     | +-ARM_920/
  |     +-Overview/          ← Product overview sheets / brochures
  |     +-QuickRef/          ← Quick reference manuals (ESAOA docs moved; see below)*
  |   +-Team/                ← Team hardware documentation
  |     +-Datasheets/        ← Component datasheets obtained by team
  +-Copyright/               ← Copyright / licensing information for the project
  +-Documentation/           ← Product / other documentation
    +-Communal               ← Communal documentation
      +-ESAOA/               ← Documentation for ESAOA (includes ESAOA quickref) *
      +-Forum/               ← Documentation about using the forum
    +-Team                   ← Team's product documentation
  +-Templates/               ← Other templates
```

Figure 6.7: ESAOA version 2 team distribution directory structure.

University of Cape Town

6.3.8 The knowledge base within ESAOA workspaces

A knowledge base exists within an ESAOA workspace (as modelled in Figure 6.4). A knowledge base generally comprises a collection of interrelated concepts used to convert forms of tacit knowledge into explicit knowledge in order to archive and share this explicit knowledge [Krishna, 1992]. A common method of implementing a knowledge base involves using a database system to capture the information used in an organisation [Krishna, 1992]. This point is corroborated by Hahn & Subramani [2000], who describes a knowledge base as a system that typically provides specialised classification and archiving facilities, together with a search engine, and that is often used as an aid in problem-solving tasks.

The knowledge base used in the ESAOA KMS is similar to the type described by Krishna [1992], in that the ESAOA knowledge base differs from those more commonly associated with expert systems and artificial intelligence. The ESAOA knowledge base should thus be considered a "system that supports at the conceptual level structures for representing knowledge, while providing, in addition, all the services we have come to expect from a database system" [Krishna, 1992, pg. 18].

The knowledge base in an ESAOA workspace is focused on storing explicit knowledge relating to the adaptation and organisation of ES artefacts (as opposed to the broader range of organisation-wide knowledge). The ESAOA knowledge base comprises a collection of knowledge artefacts interwoven with (or in the form of) soft artefacts stored in ESAOA workspaces. Both the first and second versions of the ESAOA support tools provided means of searching, viewing and manipulating the knowledge base within an ESAOA workspace (e.g., by accessing file meta-data and searching code comments). For example, the *fclass* tool doubles as a knowledge base management tool, in that it can be used to search for files with particular artefact classifications, and the ESAOA *find* tool can be used to search for text in either all files or a restricted subset of files within an ESAOA workspace. However, the support tools have limited functionality; for instance, there is no facility to combine their results or to automate updates to knowledge artefacts based on query results (i.e., if the *fclass* and *find* tools supported SQL commands, each would only support the 'SELECT' command applied to one table at a time). Nevertheless, the ESAOA tools do produce text output that can be combined and they do allow automated updates to be performed using text-processing tools such as *grep* [GNU, 2009]. Later

University of Cape Town

versions of the ESAOA KMS are expected to support more complex knowledge base querying and administration.

A scenario is presented in Figure 6.8 that demonstrates how the knowledge base within a workspace can be accessed. The scenario begins with the user applying knowledge by editing a device driver file that is stored in an ESAOA workspace. Next, the user issues a command that links classification keywords to the file that records the type of knowledge applied to the file. The third command issued links a description to the file that was edited (the 'desc' feature of *fclass* allows all descriptions maintained in this way to be searched in a uniform manner regardless of the type of file concerned)[2]. Later, the user executes queries of the knowledge base (again using the *fclass* tool to access file metadata – in the example, both queries return the same file name, namely the file 'rs232.c').

Command issued	**Narration**
$> vim rs232.c	User edits a file to apply knowledge learned about controlling a hardware device
$> fclass rs232.c + driver + serial	User adds classifications of 'driver' and 'serial' to the file edited.
$> fclass rs232.c desc \ "serial port driver for csb337 port0"	User adds description to record what the file relates to, in this case indicating the platform and specific hardware component concerned.
$> fclass . qclass serial rs232.c	User queries the workspace (using the –f find option of *fclass*) for all files with classification 'serial' applied.
$> fclass . -qdesc csb337 rs232.c	User runs query to find all files with the word 'csb337' in its *fclass* description field.

Figure 6.8: Scenario showing access to integrated knowledge base using *fclass*.

6.4 ESAOA support tools

This section describes how ESAOA support tools for version 1 of the ESAOA KMS were adapted, removed or added to in the course of developing version 2 of the

[2] Generally, a detailed description of code file is easier, and more natural, to maintain within the file itself using comments. The 'desc' option of *fclass* is applicable to maintaining short descriptions that are edited or searched in the same way, regardless of the file type.

University of Cape Town

KMS. The *fclass* tool, used in classifying and tagging files in an ESAOA workspace, was changed significantly from the earlier version; Section 6.4.1 provides detail on these changes and explains the addition of the PEP service used to optimise performance of *fclass*. Section 6.4.2 focuses on the *hsl* hotspot logging tool, which supplements *fclass* by providing additional file tagging facilities. Improvements to the *esaoa-project* tool, which is used for maintaining project information, are described in Section 6.4.3. The addition of *esaoa-checkin* and *esaoa-checkout* scripts, which are used for synchronizing workspaces, are summarised in Section 6.4.4. Lastly, workstation-side scripts, and the associated WSS and WSC programs developed to activate these scripts, are described in Section 6.4.5. Refer to Appendix C.3.4 for the full list of ESAOA version 2 support tools and scripts. Detail concerning testing of individual ESAOA support tools is not in the scope of this thesis, as this issue would distract from the focus of this work.

6.4.1 The ESAOA file classification (fclass) tool

The ESAOA command line file classification tool (named *fclass*) was initially used in the first version of the ESAOA KMS. Participants of Experiment 2 found the tool suitable, but poorly documented. For this reason, an upgraded version of *fclass* is included in the second version of the KMS. The *fclass* tool is a command line program that is executed from within the ESAOA environment of an ESAOA workspace (Figure 6.9 shows a screenshot of the *fclass* tool).

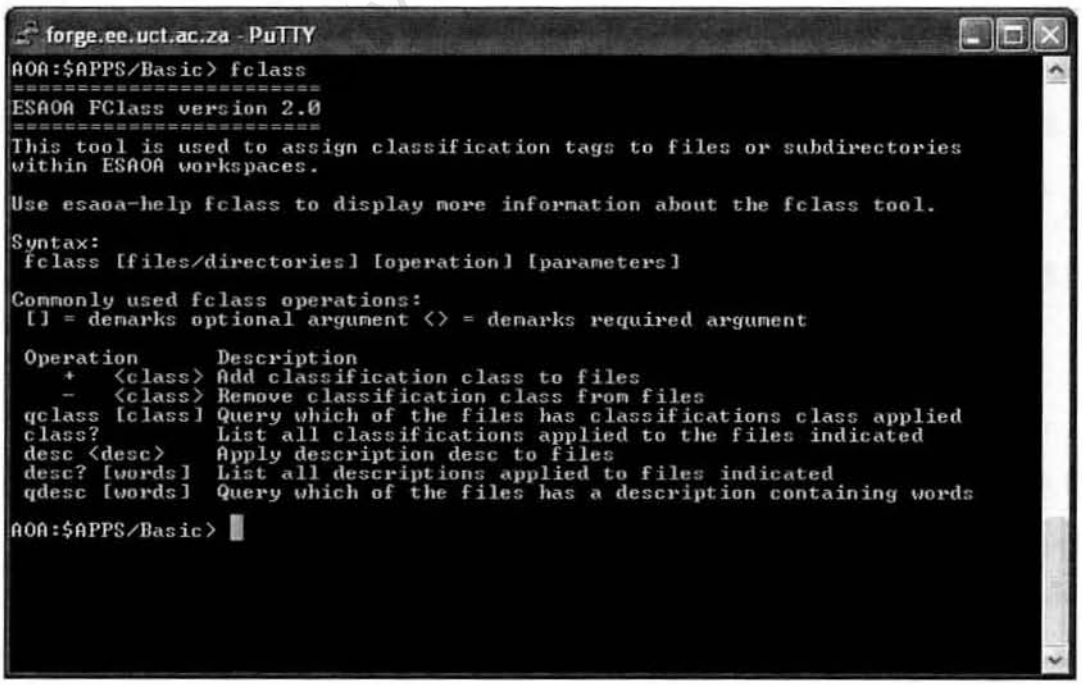


Figure 6.9: Screenshot of *fclass* version 2.

It was noted (in Section 5.6.2.2) that the CSV file format used by *fclass* to maintain file classifications was difficult to understand. For this reason, the operation of the revised *fclass* was designed to be closer to that of a database system. An initial prototype for *fclass* was developed that utilised a mysql database [MySQL, 2009] via the mysql C API [Vaswani, 2004]. A lengthy development process followed, which led to the implementation of a mysql-based *fclass* program. The program worked within personal workspaces; but attempts to synchronize the file metadata with the team workspace, or to copy it to different workspaces, as requested during testing of version 1, were unsuccessful. Furthermore, difficulties were experienced with moving and renaming files – a design decision, which could not easily be changed, was a major cause for this problem, namely, the fact that the mysql database tables used absolute path names to associate files with their classifications. Since it was taking too long to remedy this difficulty, it was decided to discard the initial, mysql-based revision to *fclass*, and to revert instead to the simpler and more portable strategy of using CSV files. The next subsection recaps the design of *fclass* version 1, which then leads into the revised design for version 2 of the tool.

6.4.1.1 Review of version 1 of *fclass*

CSV files of version 1 of the *flcass* program made use of two types of files: the file *classification lookup file* (FCL) and the *file classification index* (FCI) file. All the file classifications used in an ESAOA workspace are listed in a single FCI file that is named '.fci' and that is stored in the root directory of the active workspace (examples of the file classifications used in an ESAOA workspace are listed in Section 4.6.5.2).

Each directory in an ESAOA workspace has one FCL file, named '.fcl'. This ',fcl' file is a CSV file used to associate one or more of the classifications (recorded in the workspace's .fci file) with files in the same directory as the one in which the '.fcl' file is located. Each row in the '.fci' CSV file (which corresponds to a classification association) has three columns, respectively titled 'code', 'description' and 'inherit'. The 'code' column is a code value that refers to a particular classification (these code values are assigned automatically by the *fclass* program according to the description given). The 'description' column contains one or more words, which describe the classification. The 'inherit' column is effectively a list of zero or more code values, which indicate the super-classes of the classification concerned (i.e., they indicate which existing classifications the new classification inherits). Table 6.2 gives an excerpt from an .fci file in the form of a table (Figure 4.15 in Chapter 4 shows the actual contents of the file).

University of Cape Town

In order to understand the structure of the .fci file illustrated in Table 6.2, begin by considering the second row of the table (i.e., the line reflecting the value 'ACM' in the code column). This row indicates the classification, where the code value 'ACM' is described as an 'application code module', and it inherits the higher level classification with code value 'C'. Further investigation of the table shows that classification code 'C' is described as a 'Code' artefact, so clearly an application code module is a type of code artefact.

Table 6.2: A tabular view of a .fci file in an ESAOA personal workspace.

Code	Description	Inherit
ACM	Application code module	C
BE	Binary executable	S
C	Code	S
CMF	Code map file	S
CD	Concept drawing	D

The FCL file is used to link files to the classifications described in the workspace's FCI file. Each row in the '.fcl' CSV file has two columns titled 'filename' and 'function'. The 'filename' column corresponds to the name of a file that is within the same directory as the .fcl file itself. The 'function' column includes one or more code values (i.e., listed in the .fci file), which indicate the particular classification or classifications to which a file is assigned (if more than one classification is assigned, the code values are separated by spaces). One problem with the .fcl file is that it is necessary to access the '.fci' file in order to determine the description of a classification. Another problem is that role classifications are not effectively separated out in the '.fci' and '.fcl' files – there is no way of telling whether a classification is a *functionality classification* or a *role classification* (the 'function' column is inappropriately named).

University of Cape Town

Table 6.3 provides an excerpt from an '.fcl' file in the form of a table (Figure 4.15 in Chapter 4 shows the contents of the actual CSV file). In order to understand the operation of *fclass* and '.fcl' files, consider the second row of the table (i.e., the line starting with filename 'Basic.c'): this row indicates that the file named 'Basic.c' has been assigned the ACM classification (by looking in the workspace's '.fci' file, this 'ACM' code translates into the description 'application code module').

Table 6.3: A tabular view of a .fcl file in an ESAOA personal workspace.

Filename	Function
Basic.c	ACM
Basic.cm.elf	BE
Basic.cm.map	CMF

The first version of *fclass* (described in Section 4.6.5.2) was unable to send file classifications automatically from a personal workspace to the communal workspace, unless a code version control tool (e.g., CVS [Grune, 2007] or Subversion [Collins-Sussman, 2002]) was used to synchronize the workspaces – in such cases, the relevant files containing metadata would be automatically updated. In cases where a version control tool was not used, team members had to edit the '.fci' and '.fcl' files manually – but often this manual introversion was neglected, leading to artefacts classifications being lost when files were moved between workspaces.

Another drawback of the first version of *fclass* was that the workspace's '.fci' file had to be loaded (and the terminology hierarchy built up in memory) each time the *fclass* tool was executed. As a result, *fclass* had poor scalability, as it would run more slowly as the number of rows in the '.fci' file increased – this behaviour was especially problematic when copying many files because *fclass* would run for each file copied.

6.4.1.2 Version 2 of *fclass* and addition of the PEP service

The synchronization mechanism between personal workspaces and team workspaces (including file classification information) has been remedied in the design of ESAOA KMS version 2. Synchronization of team and personal workspaces is designed around using versioning control tools (such as Subversion [Collins-Sussman, 2002]) or the *esaoa-checkin* and *esaoa-checkout* scripts (Section 6.4.4 details these scripts). These revised synchronization mechanisms avoid the time-consuming manual procedures need to copy and edit .fci and .fcl files maintained by the *fclass* program.

Due to the way in which the communal workspace is used, synchronization tools are not used with the communal workspace (since teams do not necessarily want to send all their project artefacts to the communal workspaces). Consequently, to reduce the manual overhead of editing.fci and .fcl files, the revised version of *fclass* has been made aware of workspace boundaries (this was done by adding ESAOA environments variables and comparing source and destination paths of the esaoa-mv and esaoa-cp commands). The *fclass* program now makes use of the ESAOA_ROOT environment variable, which stores the path of the currently active workspace (usually the user's own personal workspace), and uses the ESAOA_TEAM environment variable, which stores the path of the team workspace. The ESAOA_COMMUNAL environment variable stores the path to the communal

University of Cape Town

workspace (provided it is available on a network drive – otherwise the variable is set to an empty string). Whenever files are copied using *esaoa-cp* (which is called from the *esaoa-checkin* and *esaoa-checkout* scripts), the *fclass* program is invoked by the *esaoa-cp* command to copy file classifications for the file as well as the file specified.

6.4.1.3 Speeding-up the operation of *fclass* using the PEP service

The previous implementation of *fclass* was rather slow. Most of the code used in the *fcs* program (*fcs* is described in Section 4.6.7.1) was duplicated in *fclass* version 1. Each time *fclass* loaded, the program used *fcs* methods to read the entirety of the .fci file for the workspace (even if it was not needed for the operations issued). In addition, the program also read, and if necessary wrote, all the .fcl files related to the files whose metadata was manipulated. In order to make *fclass* version 2 more responsive than its predecessor, the program was redesigned to use inter-process communication (IPC) [Allen, 2000] to make calls to a separate program, called the *Personal Expert Program* (PEP), which runs as a *daemon* or *background program*[3]. This PEP service maintains the .fci file containing the classification indexes for the active workspace. Figure 6.10 uses UML to visually describe these relations between *fclass*, PEP and other files in a workspace.

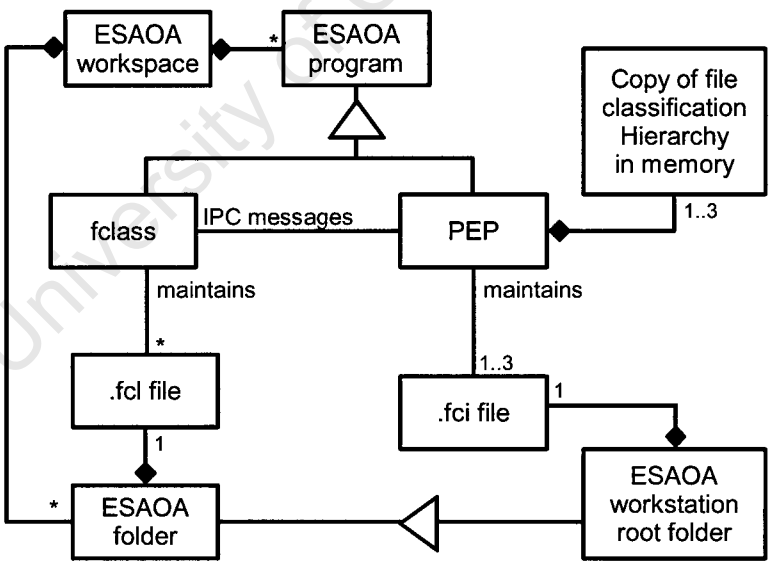


Figure 6.10: UML model of relationships between *fclass*, PEP and related files.

[3] Background programs, or 'daemons' as they are sometimes termed, run without user interaction and provide services to other programs [Enderunix.org, 2008].

University of Cape Town

PEP is started when the user enters the ESAOA environment (i.e., by issuing the *esaoa-enter* command). PEP replaces the functionality classification index manager (i.e., the *fim* tool) mentioned in Section 4.6.7.1.

An overview of the software design for the *fclass* tool is shown in Figure 6.11; the diagram uses standard UML 2.0 *stereotypes* (which are explained by Kroll & Kruchten [2003]). The *fclass* tool was implemented in C++, and its design comprises five main classes. The *fclass command line interface* provides the human/computer interface; most of this class involves parsing the command line arguments sent to the *fclass* program. The FCS (for 'File Classification System') class contains a collection of methods that use the FCL and FCI classes to access classifications associated with files. The FCL class is used to manage .fcl CSV files in an ESAOA workspace. The FCI class is an abstract interface class, and it has two concrete subclasses, namely: the FCI Manager and the FCIClient. The FCI Manager accesses a workspace's .fci file directly, via the KitCSVFile class, to manipulate the file classification index of an ESAOA workspace (FCI files are described in Section 6.4.1.1). The FCI Client implements IPC messaging, which sends requests to the PEP service to access or modify .fci classification records. By default, FCI Client is used to manage FCI information; however, a command line parameter can be set that makes *fclass* use FCM Manager instead (e.g., if IPC messaging fails). KitCSVFile and KitFile are both part of the KIT toolkit (see Section 4.6.7.2). KitCSVFile provides reusable methods for reading and writing CSV files, and KITFile implements methods for listing files in a directory and for accessing file attributes.

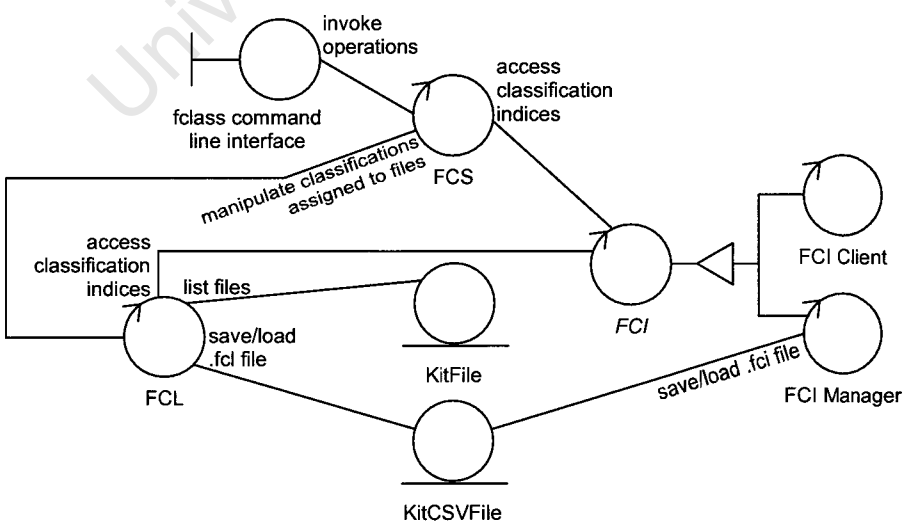


Figure 6.11: UML diagram showing software design of the fclass program.

University of Cape Town

6.4.1.4 Operation of the PEP service

The first thing that the PEP program does, when launched by *enter-esaoa*, is to load the active workspace's .fci file, which it then keeps in memory. Whenever *fclass* needs to perform an operation on the .fci classification index (e.g., to look up a classification code), the PEP program is requested to perform the operation via an IPC message.

The scenario in Figure 6.12 demonstrates the interaction between *fclass* and PEP (the sequence of events starts from the top left of the figure, namely the invocation of *enter-esaoa*). As illustrated in the figure, the PEP service is started when the *enter-esaoa* script is called. The PEP service immediately loads in the .fci file, which stores the classification index for the current workspace. The user then enters a *fclass* command to request the classifications applied to a file named 'Basic.c'. The *fclass* program is executed, and starts by reading in the .fcl file, which links classifications to file names. The program searches through the .fcl file to find all classifications codes linked to the file named 'Basic.c'. Once the list of classification codes applied to 'Basic.c' has been determined, *fclass* then uses IPC messages to request the PEP program to convert the classification codes into their full classicisation names. *Fclass* ends by printing out these classification names. Appendix C.3 lists PEP IPC messages, together with more detail about its operation.

The PEP program can be started in standalone mode, in order to change .fci file settings directly, and to perform other operations, such as testing and debugging. In standalone mode, the user interacts with PEP via a text-based menu structure.

6.4.1.5 Improvements to the CSV files for storing file metadata

The CSV format of the .fci and .fcl files was changed. As mentioned previously, role and functionality classifications were mixed up in these files in the first version of the *fclass* program. In addition, the content of these CSV files was largely unreadable to a human user as each entry in the .fcl file used classification codes that were stored in the workspace's .fci file. At the cost of some data duplication, a 'description' column was added to the .fcl file (i.e., a third column). The description column defaults to a verbatim copy of the description column found in the workspace's .fci file. A fourth column was added to the .fcl file to keep track of role classifications. Furthermore, as a result of having only one description column, and to make it easier to link descriptions to either roles or classifications, each row in a .fcl file has data in either the 'function' or the 'role' column, not both.

University of Cape Town

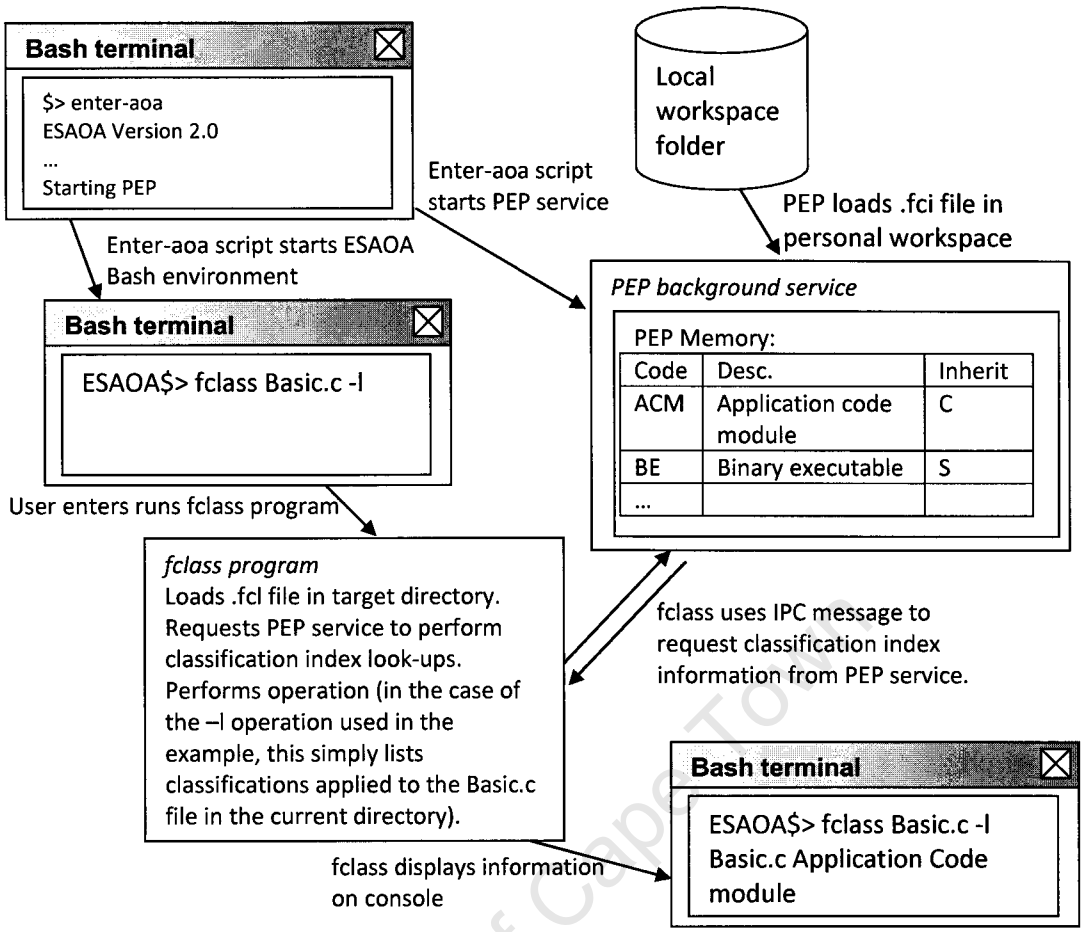


Figure 6.12: Scenario demonstrating interaction between *fclass* and PEP.

Altogether, the changes to the .fci and .fcl files mean that the structure of these files is closer to that of tables in a database. Table 6.4 illustrates this point by using a tabular view of a .fcl file. The table shows how an additional row, with the same file name as another row, is needed to add both a functionality and a role classification to an artefact (in this case to a file named 'Basic.c').

Table 6.4: Tabular view of a second version '.fcl' file.

Filename	Description	Function	Role
Basic.c	Application code module	ACM	
Basic.c	Process engineer		PE
Basic.cm.elf	Binary executable	BE	
Basic.cm.map	Code map file	CMF	

The new version of *fclass* can keep track of URLs related to information used in constructing an artefact and can also record from which workspace the artefact originated. This is done using another CSV named '.fos' (for 'file origin and sources'). The '.fos' file resides in the same directory as the files to which it refers. The *fclass* program provides a set of command line parameters to access and modify .fos files. For instance, the '-src' parameter is used to specify the originating workspace of a

University of Cape Town

file, and the '-url' parameter is used to specific a source in the from of a URL. Table 6.5 presents an example .fos file. The .fos file shown in Table 6.5 can be generated using the following sequence of commands: 1) the command 'fclass Basic.c -src communal' adds the 'communal' entry in the first row; 2) the command 'fclass Basic.c -url http://www.cforum.com' adds the URL entry in the first lineof the file; and 3) finally 'fclass Test.dat -src Team1' adds the second line in the file. Appendix C.3.2 provides more detail concerning the use of these *fclass* parameters and '.fos' files.

Table 6.5: Tabular view of a second version '.fos' file.

Filename	src	url
Basic.c	communal	http://www.cforum.com
Test.dat	Team1	

6.4.1.6 The *fclass* HTML generator mode

Thus far, the *fclass* tool has only been shown to produce unformatted text output; this approach was maintained, as it makes it easier to pipe output from the *fclass* tool to other programs (e.g., to use the GNU *grep* utility to filter the output produced by *fclass*). An option (namely the '–h' parameter) has been added to *fclass* to generate HTML output for classification queries and description queries (i.e., the '–h' option can be used prior to either the –qclass or –qdesc parameters to generate HTML output). By default, when the –h option is used, a file named 'out.html' is created in the root directory of the user's personal workspace to hold the query results. It is intended that the user will keep the out.html file open in a web browser and hit the update button whenever a *fclass* query is issued. The '–hd' option can be used instead of –h to force the HTML output of *fclass* to be output to the console (thus allowing the output to be redirected elsewhere). Figure 6.13 shows a screenshot of the out.html file when viewed in a web browser.

University of Cape Town

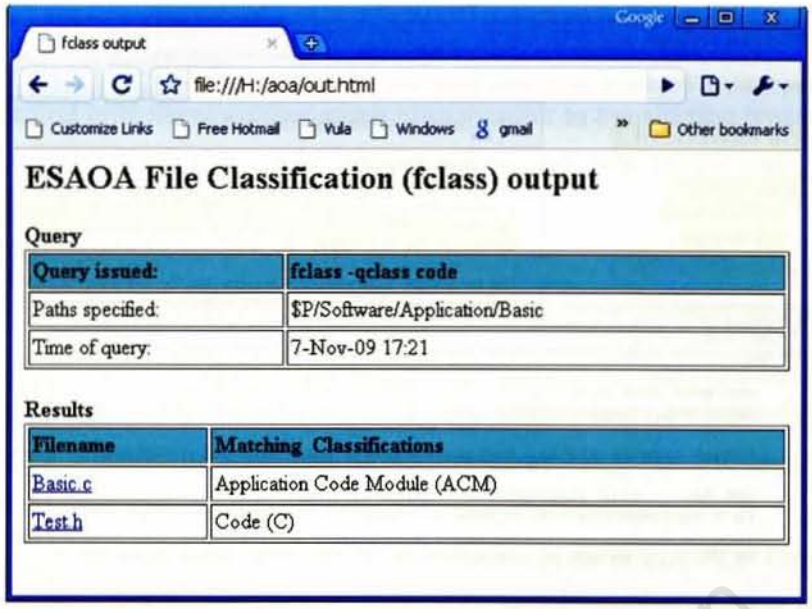


Figure 6.13: Screenshot showing sample HTML output of *fclass*.

6.4.2 Addition of the hotspot logging (*hsl*) tool

The term *knowledge hotspot* (or just *hotspot,* as listed in the ESAOA knowledge ontology in Appendix C.1) demarcates a section of a soft artefact. Illustrations of hotspots taken from Project P2-2 in Experiment 2 are shown in Figure 6.14. As the figure demonstrates, a hotspot may include, among other things, a critical paragraph of a datasheet or a block of code in a code module. These hotspots typically relate to important information that a developers needs to read and understand in order to comprehend and communicate a certain body of technical knowledge that is related to a particular artefact. An additional support tool has been added to ESAOA KMS version 2 to assist in maintaining these hotspots; the program concerned is the *hsl* tool (or HotSpot Logging tool).

The *hsl* tool is designed to operate in a similar way to the *fclass* tool, in that it stores file metadata in CSV files. The tool uses a file named '.fhl' (an acronym for 'file hotspot log') to store metadata. A prototyped *hsl* is added to ESAOA KMS version 2, but it has limited functionality (it has no visualization features and cannot copy hotspot information between workspaces). Although a way of visually highlighting hotspots would make the *hsl* tool more usable (e.g., showing highlighted hotspots in text editors or PDF readers), the prototyped *hsl* has no such facility as yet (such features may be added to the program if it is found useful in future pilot projects).

University of Cape Town

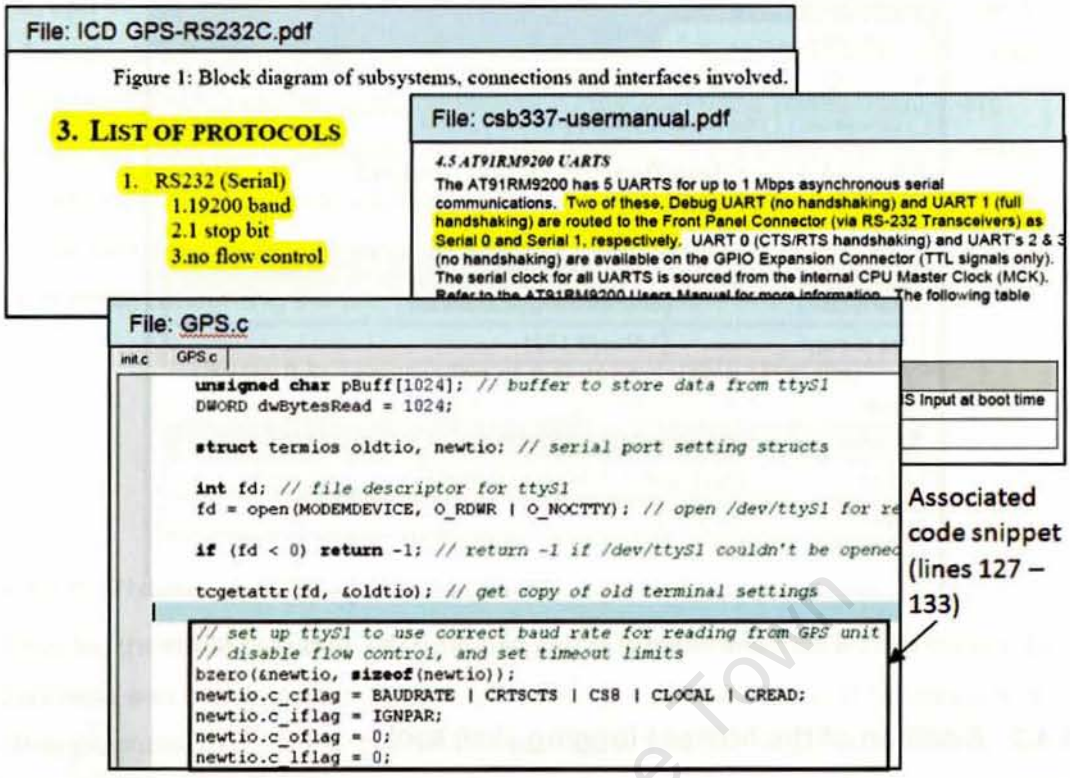


Figure 6.14: Examples of hotspots taken from Project P2-2.

Table 6.6 provides an example showing how the .fhl file would record hotspot information; the entries in Table 6.6 correspond to the highlighting shown graphically in Figure 6.14. It is important to note that all the files referred to in a .fhl file need to be in the same directory as the .fhl file.

Table 6.6: Examples of a .fhl file corresponding to Figure 6.14.

Filename	Postype	Start	End	Link	Description
ICD GPS-RS232C.pdf	2	2 21	2 25		Communications protocol settings for GPS receiver
csb337-usermanual.pdf	2	17 19	17 22		How to configure UART
GPS.c	1	231	237		Configure comms protocol

A .fhl file has five columns, as is shown in Table 6.6. These columns are summarised in sequence below:

- *Filename:* name of the file to which the hotspot relates.
- *Postype:* how the starting and ending location of the hotspot is represented (the title 'postype' is short for 'position type'). This column contains the value 0, 1 or 2, used to specify what the 'start' and 'end' columns represent. The values are interpreted as follows: 0 indicates 'start' and 'end' refers to character offsets in the file; 1 means 'start' and 'end' are the line numbers;

University of Cape Town

and 2 implies that the two columns refer to a page number followed by a line number on that page.

- *Start* and *end:* these columns respectively refer to the starting and ending position in the file that corresponds to the hotspot (see above point concerning 'postype').
- *Link:* used to link to another file that describes the hotspot in detail (this column contains either a URL or the name of a text document that resides in the current directory).
- *Description:* text that briefly describes the hotshot (e.g., a reason for it being marked – this information can be elaborated by using the 'link' column).

Many popular programs used for reading documents (e.g., *Adobe PDF Reader* and Microsoft Word) support annotation features such as highlighting text and adding comments that could be used to identify hotspots. However, the implementation of the *hsl* tool has not been integrated with these popular software tools because there was only limited time available to prototype the *hsl* tool, and the same hotspot recording system needs to be utilised with a variety of file types.

The UML diagram in Figure 6.15 gives an overview of the software design for the HSL tool. The diagram shows that the HSL tool is composed of three parts, each of which is implemented as a C++ class. The *HSL command line interface* class provides the human/computer interface to the user, which invokes methods within the *HSL Manager* to carry out operations requested by the user. The *HSL Manager* implements the main functionality of the HSL tool, such as adding or removing a hotspot description. The *KitCSVFile* is used to maintain a .fhl CSV file, and is the same class as used in the design of *fclass* and *PEP* (see Section 6.4.1).

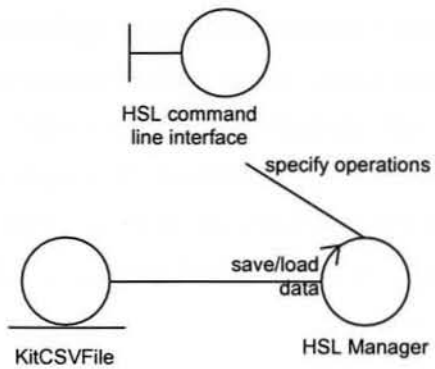


Figure 6.15: UML diagram showing overview of the HSL tool's software design.

University of Cape Town

6.4.3 Improvement to the *esaoa-project* tool

Additional features were added to the *esaoa-project* tool, the tool used to keep track of keywords and general information related to a project folder within an ESAOA workspace. Project information records have been added to the data kept in memory by the PEP service. These records include the full name of the active project, its acronyms and team member names (Appendix C.3 lists the relevant IPC messages).

6.4.4 Tools for synchronizing team and personal workspaces

The *esaoa-checkin* and *esaoa-checkout* scripts are provided in the ESAOA communal distribution to facilitate the process of synchronizing team and personal workspaces. Note that these scripts should not be used if version control software is used to synchronize the workspaces.

The *esaoa-checkin* script overwrites files in the team workspace that have been changed in the personal workspace – if the original version was changed by the same user (i.e., if the *owner attribute* [Von Hagen, 2007] in a Linux file systems is unchanged). If the *esaoa-checkin* script encounters a file that has been changed by a different user, then the user performing the synchronization is prompted to do one of the following operations: 1) overwrite the file; 2) skip the file, or 3) make a backup copy of the file before overwriting the file. Generally, it is expected that these prompts will occur infrequently if teamwork is suitably partitioned, and if the team as a whole has made appropriate file ownership decisions.

The *esaoa-checkout* script operates in the opposite direction to *esaoa-checkin* (i.e., it copies files from the team workspace to the personal workspace). This scripts also prompt the user before overwriting files that have been changed by a different user.

The *esaoa-checkin* and *esaoa-checkout* scripts are designed to work on ext2 or ext3 [Von Hagen, 2007] file systems (including Cygwin simulations of these). This allows the scripts to be portable, without needing additional tools to be compiled or installed (besides the *fclass* tool which is used by both of these scripts). A potential drawback to these scripts is that they can only write whole files at a time; they cannot do patches (e.g., using the GNU patch tool [MacKenzie *et al.*, 2003]) to synchronize changes to a particular files.

University of Cape Town

6.4.5 Workstation-side scripts

Workstation-side scripts are scripts that reside on the user's workstation computer, and are activated remotely from an ESAOA workspace that may reside on a networked server machine. Workstation-side scripts allow debugging software tools, code loaders and other tools that run on the workstation computer to be automatically started from scripts running in the user's personal workspace (note that the personal workspace could reside on a remote machine). The incorporation of workstation-side scripts is a response to an inconvenience noted by developers: all the developers had to manually repeat many of the same sequences of activities on their workstation computers (e.g., downloading a compile executable to the workstation, and then writing the file into flash on the ES platform). Workstation-side scripts allow many aspects of these mechanical procedures to be automated.

Workstation-side scripts are implemented via a custom-built TCP/IP socket program; there are two parts to the design: a WSS service (workstation-side server application), and a WSC (workstation-side client application). Both Microsoft Windows and Linux implementations of the WSS were developed. The WSS can be started either manually by the user, or it can be configured to start when the operating system loads. The WSS invokes either BASH or Windows BAT scripts, which are placed in the WSS directory (by default, this is set to directory '~/WSS' on Linux, or 'C:\WSS' on Windows). The WSC program is called from within the ESAOA environment, and it uses a socket connection to involve one of the scripts in the WSS directory on the workstation computer. By default, the WSC program reads the workstation computer's IP number from a file named ~/.wsc in the user's home directory (alternatively, the –ip command line parameter can be used to override the IP address used).

The prototyped WSS and WSC programs have certain limitations, particularly in terms of security. The main problem is that when a user changes from one computer to another, and has not updated the ~/.wsc file, server-side scripts on the previously used computer (i.e., that the user using before) can be inadvertently activated. Future versions of the WSS and WSC applications will need to be able to synchronize their IP addresses to avoid the problem of invoking scripts on the wrong workstation.

University of Cape Town

6.5 ESAOA roles

The roles of ESAOA KMS version 2 are structured on the basis that they have equivalent levels of importance, without any one role dominating the others. This restructuring of the roles, as mentioned in Section 6.1.1, was also done to facilitate innovation, while avoiding the risks of neglecting other factors of development that could result in poor quality results (see Section 6.1.2.1). In addition, the revised roles improved the correspondence between the described role procedures and the actual role behaviours that were carried out in practice during the ES prototyping projects of Experiment 2 (and as per the research delimitations described in Section 1.6).

ESAOA KMS version 2 thus has seven knowledge worker roles, which are summarised in Table 6.7. The team member (TM) role, which is shown in Table 6.7, is used to abstract, and collectively refer to, the TL, CR, PE and IE roles by a single name. Knowledge is transferred between all roles, however specific flows of knowledge are emphasised in the ESAOA KMS. For instance, a large portion of the knowledge related to components would be exchanged between the CR and other roles; similarly, much of the transfer of process knowledge would occur between the PE and other roles.

University of Cape Town

Table 6.7: Roles of ESAOA KMS version 2.

Role name	Acronym	Description
Chief knowledge officer	CKO	The CKO is responsible for guiding users of the ESAOA KMS, implementing or preparing new communal ESAOA artefacts and procedures for distribution and for maintaining the communal ESAOA workspace.
Communal knowledge steward	CKS	The CKS assists the CKO, helping to maintain the communal ESAOA workspace, and guiding users in procedures and workspace maintenance.
Team leader	TL	The TL is responsible for managerial aspects of the team, such as ensuring that team members are performing. The TL is also responsible for taking minutes during meetings, and ensuring that tasks are performed by one of the other team members.
Component researcher	CR	The CR obtains and produces data artefacts (such as datasheets), reads them, and places them in the ESAOA team workspace. The DS is responsible for maintaining the organisational structure of the ESAOA team workspace and for supporting the PE.
Workspace administrator	WA	The WA is responsible for maintaining the consistency of a team workspace. This task involves ensuring that files are suitably named and placed in a shared directory structure. The WA acts partly as a librarian for the project.
Process engineer	PE	The PE experiments with processes using the data knowledge and support provided by the CR and the WA. The PE is responsible for creating process artefacts and for supporting the IE.
Innovation engineer	IE	The IE applies processes to test ideas and to produce innovations. The IE is supported by the PE and WA.

In the new set of roles, the IE remains focused on testing innovative design ideas (as per the previous version of this role). The objectives of the CR and PE differ from those of the DS and PE of the first version. The CR and the PE have two main objectives: 1) to achieve expertise in producing and managing a specific form of knowledge; and 2) to anticipate and distribute innovative ideas. The role of WA was added to ensure that someone was assigned the responsibility of checking that the team workspace is kept in a functional state. The subsections that follow provide further details about the roles outlined above.

In ESAOA KMS version 2, each artefact within an ESAOA workspace can be assigned a role classification using the *fclass* tool. These role classifications can be used to indicate which role is responsible for maintaining the quality and consistency of a particular artefact[4] (Table 6.8 provides examples of role classifications).

Table 6.8: Examples of artefacts and role classification.

Artefact Description	Role classification
Component datasheets, e.g. datasheet for a microcontroller or analogue to digital converter device.	D artefact
Manuals for a development tools (e.g., GCC user manual)	PE artefact
Concept diagrams or concept sketches of the prototype	IE artefact
Status reports	TL artefact
Sample code, experimental code files used for testing components and interconnections	PE artefact
Data files (e.g., images, saved signal samples)	D artefact

University of Cape Town

6.5.1 The WA and CR roles

As mentioned in Section 6.1.3.1, the WA and CR roles have replaced the DS role, having been added to address limitations of the ESAOA KMS version 1 roles. Activities related to researching components have been assigned to the CR role. The CR role is tasked with maintaining *D artefacts* (see Table 6.8 for examples of D artefacts). The CR performs activities such as searching for datasheets and investigating which components are suited to filling particular project requirements.

The WA is expected to support other team members in terms of organising and finding artefacts, and assisting in the placement of new artefacts into personal or team workspaces. The WA is also assigned the responsibility of inspecting the

[4] A file's owner attribute could be used for this purpose; however, role classifications are used in ESAOA workspaces because different users could fill the same role at different times.

quality of artefacts; this responsibility was added in response to the findings of Section 5.5.2, which showed that teams with successful design reviews and good quality artefacts tended to complete product requirement more successfully. This responsibility of inspecting artefacts is focused on the *presentation* and *structure* of the artefacts, rather than their construction, because it is expected to be impractical for the WA to achieve a detailed understanding of all team artefacts (hence, the intention is to allow roles to offload some of their artefact administration to the WA, so as to focus on building knowledge in their knowledge areas). Figure 6.16 models the processes for which the WA is responsible.

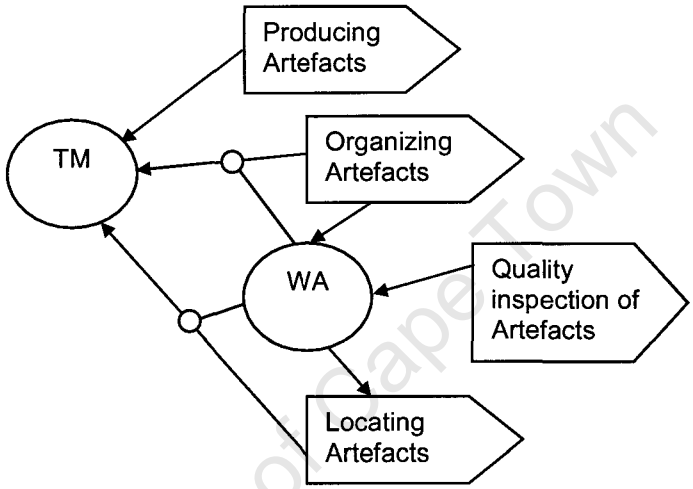


Figure 6.16: Role support provided by the WA.

The WA role is likely to be performed by one of the other roles (e.g., the PE may also be the WA). This needs to be accounted for during development, so that the WA role is assigned appropriately. For example, assigning the WA and IE roles to the same person may be pointless if, as is shown in Experiment 2, production of innovation knowledge is likely to reach its peak towards the end of a project. Such a situation may lead to the artefacts becoming disorganised and inconsistent towards the end of the project, thereby obstructing, instead of facilitating, the final critical phases of a development project.

6.5.2 Chain of command

In terms of team management, the roles are non-hierarchical, with the exception of the TL. All team members are expected to report to the TL as per typical project management practice. The CKO is in charge of the overall operation of the ESAOA KMS, and for making sure not only that it is used, but also that it meets the requirements of its users. The CKO is also responsible for maintaining the

University of Cape Town

consistency and quality of the communal workspace, and to ensure that all team members have access to it. The WA oversees the consistency of the team workspace and is assisted and mentored by the CKO and CKS. An organogram for the chain of command for the roles is provided in Figure 6.17.

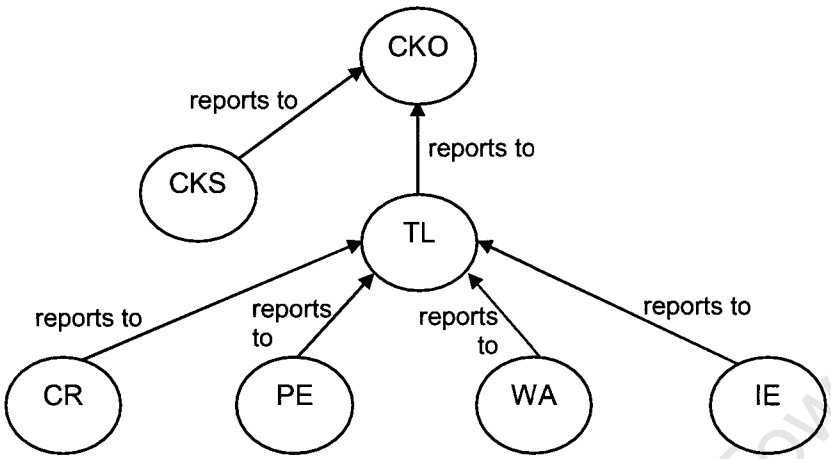


Figure 6.17: Chain of command for ESAOA KMS roles.

6.5.3 Role responsibilities

The ESAOA KMS follows an approach in which most ESAOA knowledge work is performed independently. For example, most of the ESAOA activities performed by the CR, such as finding and reading datasheets for components, are done exclusively by that individual. Knowledge is shared through face-to-face meetings, in which the roles who share knowledge are physically present, and it is accomplished with the use of boundary artefacts. For instance, the CR shows the PE how to configure a hardware device by physically interacting with the device and the soft artefacts related to its configuration.

In the early stage of a project, team members need to be assigned roles, and the responsibilities of these roles need to be made clear. The ESAOA KMS provides guidelines for allocating role responsibilities and corresponding processes and inter-role relations (Section 6.6 describes the processes performed by the various roles and the interactions between these roles; Section 6.7 elaborates on the artefacts used and maintained by the roles). The specific responsibilities of a particular role are dependent on the project concerned. For example, if the product to be built relies on GPS, then the CR could be tasked with reading about GPS modules and selecting an appropriate one, and later the PE would be tasked with establishing procedures to connect to the module and control it from software.

University of Cape Town

At a high level, the CR, PE and IE are each independently responsible for a particular form of knowledge form, and for contributing innovation suggestions during the production and maintenance of this knowledge form. Table 6.9 describes the forms of knowledge that each of these roles focus on, together with general types of innovation ideas that they are likely to discover or suggest while producing and managing this knowledge.

Table 6.9: Team member specialisations.

Role	Knowledge form speciality	Innovative ideas
Component researcher (CR)	• Data knowledge related to components (i.e., hardware components and software components used to develop embedded systems) • Data knowledge regarding interfaces available on the embedded platform used • Process knowledge related to configuring components	• Suggesting well-documented and versatile components for inclusion into the project • Components likely to satisfy product requirements • Components that may prove effective for testing concepts
Process engineer (PE)	• Data knowledge related to development tools (e.g., compilers, IDEs) • Process knowledge relating to the use of tools, components, operating system calls and middleware • Understanding device driver code • Use of hardware interfaces	• Suggesting tools to use • Recommending interface standards (based on expertise and code currently available)
Innovation engineer (IE)	• Innovation knowledge • Design concepts • Methods to test design concepts • The degree of success for particular design concepts	• Concept design ideas • High-level ideas
Workspace administrator (WA)	• Organisation of artefacts • Forms of data knowledge • Team workspace structure	• Improvements to the structure of the team workspace and team artefacts

As described in Table 6.9, the CR produces knowledge relating to the selection and configuration of pre-existing or purchased software and hardware components. The CR focuses in particular on hardware components such as integrated circuits available on the embedded platform (e.g., digital to analogue converter). The PE focuses on learning about development tools and producing development procedures that make use of these tools. The IE is not directly responsible for producing or

University of Cape Town

maintaining data knowledge or process knowledge, but rather focuses on learning and then using highly focused data knowledge, and pre-tested process knowledge, which is prepared and taught by the CR and PE. Figure 6.18 models the way in which these three roles interact. The model also indicates that the CR contributes innovative ideas (or design concepts) related to components, while the PE suggests ideas related to development procedures. The figure shows that the IE is performs the processes 'invent concepts' and 'test concepts', and collaborates with the TL on which concepts should be tested (since many ideas may be had, but there is only limited capacity available for testing the ideas).

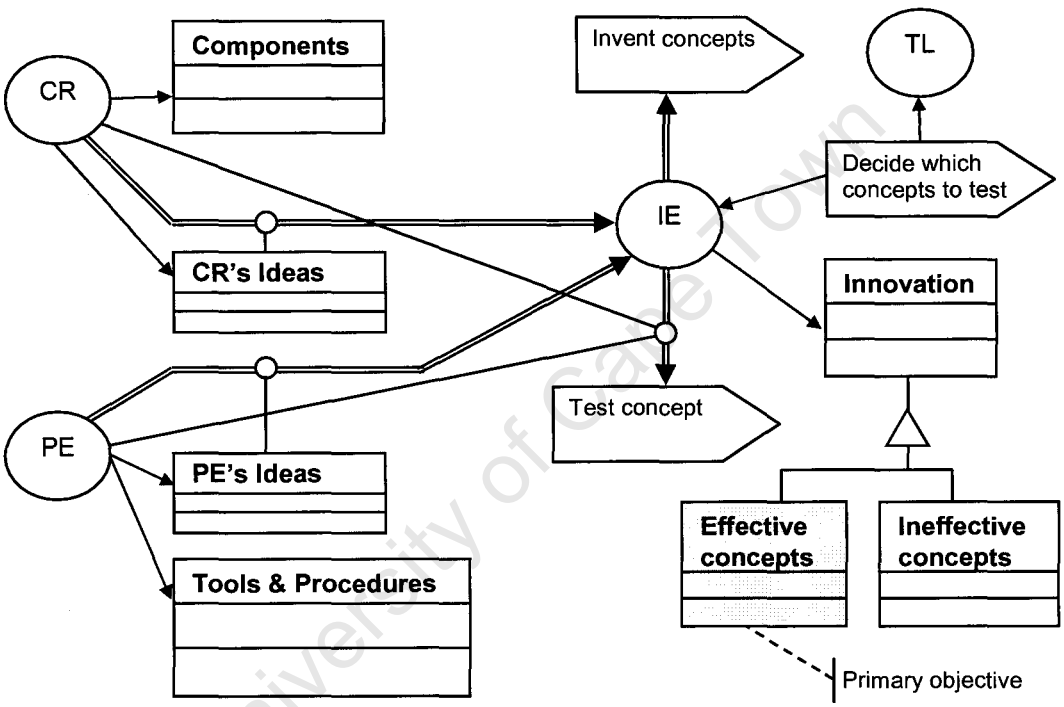


Figure 6.18: Team members and their knowledge specialisations.

In this approach to allocating responsibilities, it may appear that the CR and PE are assigned more routine tasks and fewer problem-solving activities than those of the IE. In practice, however, such a situation is unlikely because all team members are required to solve problems and to think creatively. The CR and PE are both involved in getting elements of the product to work, which is likely to involve a significant amount of problem solving. Moreover, from a project management perspective, the effort in terms of component research and process development becomes more visible, thus allowing the TL and higher-level management to obtain a broader view of the project, instead of seeing progress only on the final product.

University of Cape Town

The IE has a share of routine tasks, similar to those encountered by the PE, in which small changes to design concepts are made, and then tested and retested. The work of the IE may be slower-paced than that of the PE because the PE focuses on developing procedures to use existing products (for which a wide range of support is often available, such as online forums and chat groups). Thus, the PE may produce solutions in a matter of hours. The IE, in contrast, needs to create solutions for a new product, which may take days or weeks, and this is likely to be largely reliant on that individual's ability and perseverance. Many of the projects in Experiment 2 showed instances in which this situation occurred. Project P2-6 provides a clear example in which the IE performed meticulous work over a long time (months) for each innovation occurrence[5]. In the same project, the PE produced process solutions comparatively quickly, spending a few days producing one solution and then moving on to the next problem. In the case of Project P2-6, the PE may well have had a less routine experience than that of the IE.

Furthermore, given that the ESAOA KMS is designed around teams of two to four members (see delimitations in Section 3.2.2.3), the TL and the WA roles are likely to be assigned to at least one of the other team member roles (i.e., CR, PE or IE).

The WA ensures the consistency and quality of artefacts in the team workspace, while assisting other team members in the location and organisation of these artefacts. The WA is likely to be pleased when the other team members find that the organisation of artefacts in the workspace is both effective and accessible.

The TL is in charge of managing the members of a team and for liaising with the CKO to ensure that the ESAOA KMS is useful to the team. The TL role therefore involves mainly administrative aspects, such as provisioning resources, and ensuring that team members are capable of, and fully understand, the role they are assigned.

The CKO is responsible for the smooth operation of the ESAOA KMS as a whole, and this responsibility includes ensuring that aspects of the system, from role level to the communal level, are effective and that they meet the needs of the knowledge workers. The CKO is also in charge of the communal workspace, and for negotiating with the relevant TL for the transferral of artefacts and processes from a team

[5] Each occurrence of productive innovation knowledge is the successful implementation of a significant design concept, which the IE is likely to experience as an accomplishment.

University of Cape Town

workspace to the communal workspace. The CKO needs to perform knowledge audits, to determine the degree to which the KMS is being used, and whether it meets the needs of the users. The CKO also directs and trains the CKS's.

Each CKS is allocated tasks by the CKO. Generally, the CKS assists in maintenance of the communal workspace and incorporating elements from a team workspace into the communal workspace. The CKS and WA have many tasks in common.

6.5.4 Division of labour in development team

The ESAOA KMS focuses on development teams comprising two to four developers (as per the delimitations set out in Section 3.2.2.3). The choice of roles is intended to separate out the ESAOA activities, allowing portions of the project work to be done in parallel, thereby speeding up overall development. This section provides guidelines for the TL to share the workload of a team between its members (note that the CKO is responsible for supervising the workload of the CKS). Specifics of the various role responsibilities are elaborated upon in Section 6.5.

Roles need to be allocated near the start of a project: this approach was followed in all Experiment 2 projects, and it is recommended by the project management literature [Kliem & Ludin, 1998; Humphrey *et al.*, 1999; Addison & Vallabh, 2002]. Assigning the roles (i.e., PE, CR and IE) early on prevents team members becoming confused as to what they are expected to do. This occurred in Project P2-2, where high portions of non-productive knowledge were produced early in the project partly because individuals were unclear of their assigned responsibilities.

Each task (be it workspace administration, team management and so on) may involve different amounts of time to perform, and may differ significantly between projects. For example, one project may involve a small number of complex artefacts that are easy for the WA to keep organised within a team workspace; but a different project may have many artefacts that take longer to organise.

Figure 6.19 presents a recommended strategy for the workload breakdown of a development team of three members. This allocation of tasks is based on Experiment 2 finding, where each team had three members. Each block in the figure indicates a major task performed by each team member, while the height of a block indicates the duration for the task. Reasoning for this allocation of tasks is given below. Note that projects are not all expected to work in precisely the same way, as emphasised

University of Cape Town

previously. Thusly, this task allocation is given as a recommended starting point for new projects using the ESAOA KMS, and can be adapted as the project proceeds.

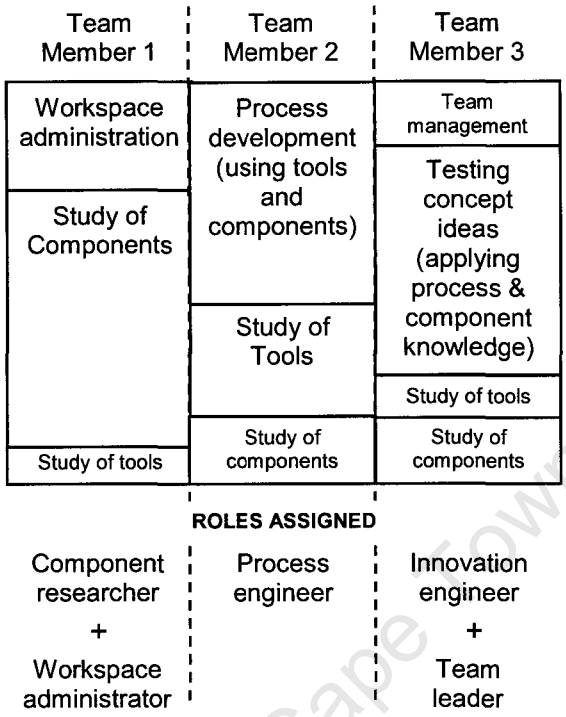


Figure 6.19: A potential scenario providing a fair division of labour (each block represents a task and the height of each block indicates the duration).

The arrangement of blocks in Figure 6.19 indicates an allocation of responsibility, in which the first team member fills both roles of CR and WA. The second team member performs only the role of PE, and the third member is both the IE and the TL. Depending on the specific project involved, it may be more effective to assign the roles differently. Experiment 2 showed that the production of knowledge could be divided between roles to some extent; however, this division of knowledge production could not be done to a high degree of precision (see results in Section 5.7.7). For example, while developing process knowledge, the PE is likely to encounter the need for additional data knowledge related to components and, depending on the problem concerned, it may be faster for the PE to pursue the needed knowledge himself or herself instead of requesting a different role to assist in obtaining the needed knowledge. For this reason, the TL should expect a particular roles is likely to produces much of, but not all, of a particular knowledge type. Correspondingly, Figure 6.19 shows that a particular team member centres on producing a certain form of knowledge, but also accounts for the production of other knowledge forms.

University of Cape Town

In Section 5.5.8, it was shown that 35 percent of knowledge occurrences on average involved ESAOA activities related to tools, while 65 percent of the occurrences related to components. This finding indicates an approximate ratio of 1:2 for the workload of studying tools to researching components, therefore the block representing study of components in Figure 6.19 is twice as high as the block for studying tools. Team management activities, as were discussed in Section 5.5.9, are likely to take considerably less time than ESAOA activities that involve studying components or producing development methods; for this reason, the team management block in Figure 6.19 is rather short. As Experiment 2 demonstrated, the development of process knowledge, and experimentation to produce innovative knowledge, takes significant effort. Consequently, blocks representing these tasks are tall. Workspace administration is expected to be less time-consuming; therefore the workspace administration block is of intermediate height.

6.5.5 Role interrelations and workspaces

In the ESAOA KMS, a role interrelation concerns the exchange of knowledge or the transfer of techniques between roles. ESAOA KMS version 2 has five categories of role interrelations: 1) relations between team members, 2) relations between team members and the TL, 3) relations between the CKS and a team member, 4) relations between the CKO and a team member, and 5) relations between the CKO and CKS. In the ESAOA conceptual modelling language, a role interrelation corresponds to a connection drawn between two role atoms (see Section 3.11.2.1).

Role interrelations are carried out in three main ways using the ESAOA KMS, namely through: 1) face-to-face meetings, 2) online collaboration (e.g., email and forums), and 3) examination and adaptation of workspace artefacts. Role interrelations can occur in other forms too, such as telephone conversations. Role interrelations described in the ESAOA KMS are carried out by modifying, or using, boundary artefacts (boundary artefacts are defined in Section 4.6.3.3). Interrelations between roles involve access or changes to boundary artefacts. Many role interactions associated with the CKO or CKS are likely to involve artefacts in the communal workspace (e.g., updating or adding to the collection of communal artefacts).

Role support structures are realised by using role interrelations. The term role support structure relates to the means by which one role is assisted by another role, or how one role can delegate duties to a second role. In ESAOA KMS version 2, the CKO and CKS support team members. Section 6.1.3 discussed the role support

University of Cape Town

structure of ESAOA KMS version 2, and cited the reasons for changing the role support structures of version 1. Figure 6.20 models the role support structure and outlines the role interactions for the KMS. A role support relation is shown as a line that links two roles, and is terminated by a solid dot on the role that is supported. As shown in the model, the CKO and CKS operate mainly in the communal workspace, while the CR, PE, IE and WA operate chiefly in their own team workspace, although they also use the communal workspace. The model also indicates that all team members report to the TL for supervision of KM strategies, and that the CKO is available to the TL for consultation on KM techniques.

The role support structure shown in Figure 6.20 accommodates a process flow in which the CR starts by researching components, while the PE then builds process knowledge with support of the CR. Finally, the IE carries out experiments using process knowledge and component knowledge produced by the previous two roles. These associations are elaborated in Section 6.6 in terms of ESAOA processes.

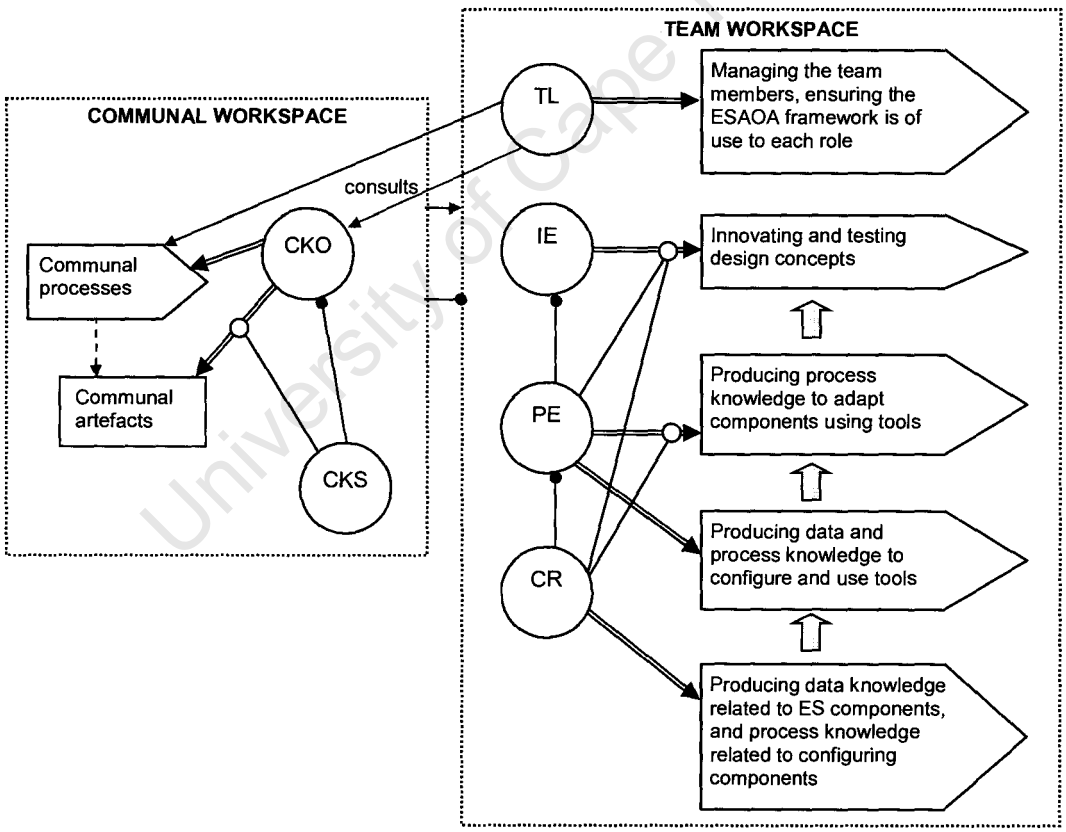


Figure 6.20: Role support structure for ESAOA version 2. The shaded arrows indicate how innovation builds on knowledge and procedures developed by the PE and CR.

University of Cape Town

6.6 ESAOA Processes

This section provides details on the ESAOA processes that occur in ESAOA KMS version 2. The processes are grouped according to the role that predominantly performs them. Role interrelations, which typically involve a transfer of knowledge between two different roles, are shown in the models for both roles.

6.6.1 Processes of the chief knowledge officer (CKO)

The duties of the CKO for ESAOA KMS version 2 are largely the same as those described for version 1 (see Section 4.6.6), but some refinements, based on the results of Experiment 2, have been made. The CKO is responsible for managing the KMS at a high level, ensuring that it addresses the needs of the knowledge workers who use it. The CKO should have computer engineering skills, be a good trainer, have experience in software engineering, and be able to program in ANSI C, C++ and Bash scripts. The CKO also requires an understanding of KM theories. The CKO influences the operation of the KMS in the following ways:

1. Guiding developers in effective KM practices within the specialised area of ES development;
2. Proactively ensuring that knowledge workers using the system are satisfied with the system, and following up requests for changes to the system;
3. Taking note of, and recording, important KM strategies used by team members;
4. Keeping track of the roles, artefacts and processes involved in effective KM strategies (and earmarking elements of team workspaces for incorporation into the communal workspace of the KMS); and
5. Maintaining the overall operation and structure of the ESAOA KMS, ensuring processes are applied consistently and support structures (i.e., CKS, ESAOA workspaces and associated computing infrastructure) are functional (this activity may include updating DAN and RSD models describing the KMS).

The processes performed by the CKO and role interrelations involving the CKO are modelled in Figure 6.21. A principal duty of the CKO is to ensure that users of the ESAOA KMS understand their designated roles and associated processes. Section 5.5.3 showed positive correlations between certain types of knowledge production and effective design reviews; for this reason, it may be beneficial for a development team to have the CKO present during design reviews to provide expert advice on KM

University of Cape Town

strategies in order to enhance productive knowledge production and improve the success of future design reviews. The model also indicates that the CKO needs to be able to manage knowledge related to components and tools, as these activities are necessary for maintaining the communal workspace.

Generally, the CKS supports the CKO in maintaining ESAOA workspaces once team members have been trained in their roles. However, the CKO is responsible for maintaining the integrity and quality of the communal workspace (as is shown by the process maintenance flow from the CKO to the workspace maintenance process atom). The figure also shows that the CKS can act as a CKO proxy, handling some of the requests submitted by team members.

The CKO is responsible for a number of administrative tasks related to the running of the ESAOA KMS, such as managing the CKSs, liaising with team leaders, and ensuring that the computing infrastructure of the KMS is functional. The CKO needs to ensure that communal artefacts and support structures are being properly maintained and extended. Liaison between the CKO and project teams is largely informal, for example, team members email requests to the CKO or inform the CKO of problems related to the use of the KMS. For the sake of efficiency, the CKO should have full access to ESAOA workspaces to make changes remotely (provided this is permitted by the security requirements for the development team concerned).

The CKO is responsible for training the CKS. This training ensures that the CKS is, in turn, able to train team members in performing the roles that have been assigned to them. In Experiment 2, there was one CKS, and he was trained by observing how the CKO assisted the first couple of teams in learning their roles.

University of Cape Town

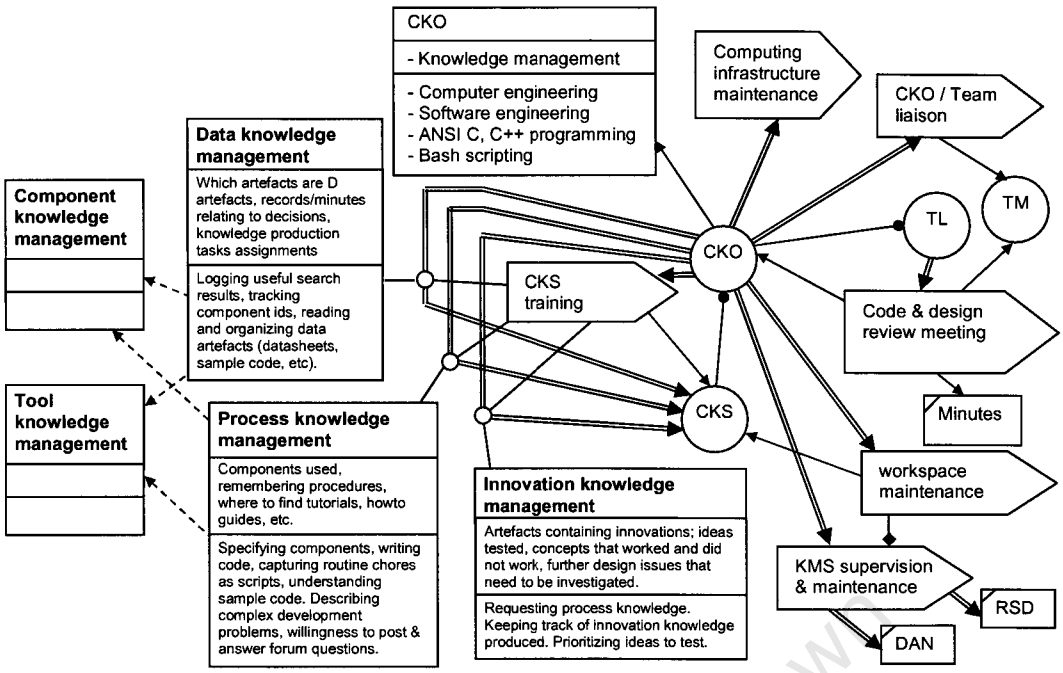


Figure 6.21: Processes performed and maintained by the CKS.

6.6.2 Processes of the communal knowledge steward (CKS)

The CKS performs a number of important duties in transferring knowledge produced in one team to other teams; these duties include maintaining the communal workspace, assisting in the training of knowledge worker roles, and capturing knowledge produced by a team in the form of communal knowledge artefacts. The CKS acts as a proxy and assistant to the CKO, being involved in training team members, helping the CKO to organise artefacts in the communal workspace and transforming team artefacts into communal artefacts (with the permission of the TL concerned).

The responsibility of the CKS starts with training team members to perform the roles they have been assigned. Once the team members have been trained, the CKS changes focus to support the CKO and team leaders in respectively maintaining the communal workspace and the team workspaces. The CKO depends on the CKS to check that teams are making use of the ESAOA KMS; the CKO also relies on the CKS to help with the creation and organisation of communal artefacts. Figure 6.22 models the processes performed by the CKS. In the first version of the KMS, the CKO was responsible for maintaining team to communal artefact conversion, but this responsibility is better allocated to the CKS, with assistance from the CKO.

University of Cape Town

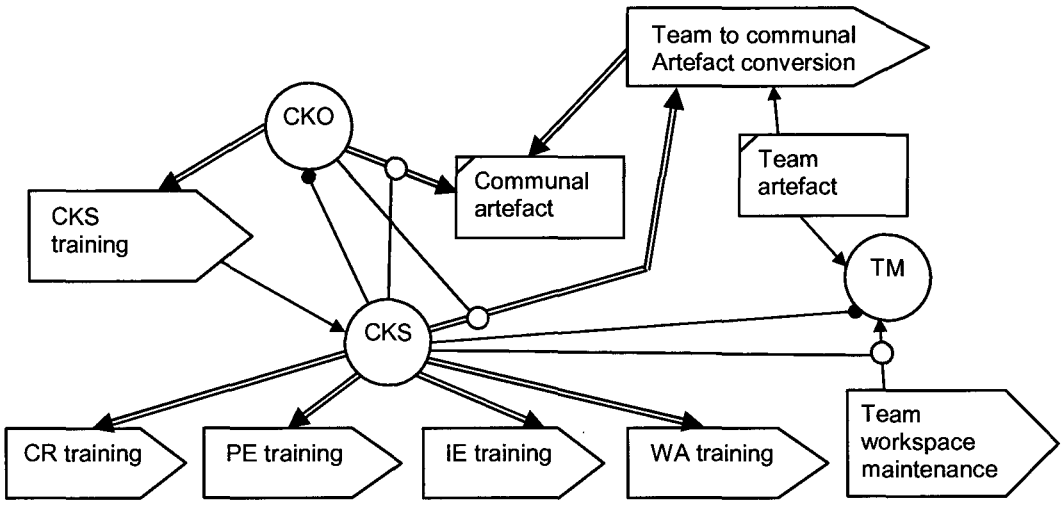


Figure 6.22: Processes maintained and carried out by the CKS.

Using common KM terminology, the CKS can be said to act partly as a "knowledge broker" and partly as a "knowledge steward" [Davenport, 2002] [Brazelton & Gorry, 2003]. The most important skills required by the CKS are tracking and classifying explicit knowledge (i.e., knowledge artefacts such as code modules), in addition to keeping records.

6.6.3 Processes of the team leader (TL)

The main processes of the TL include managing the team, ensuring that each member contributes to the project, and liaising with the CKO (or CKS) with regard to the use of the ESAOA KMS. Each project team should have only one TL, who is expected to fill at least one of the other team member roles. Section 6.5.4 provides guidelines for combining the TL role with another role.

In the first version of the ESAOA KMS, the TL was not assigned any procedures to be maintained as part of the KMS. However, based on the Experiment 2 projects, it was found that the TL was involved in various responsibilities relating to team members performing ESAOA activities. Figure 6.23 shows processes of the TL.

The code and design review procedure and team management procedures are shown as external procedures (drawn using dashed lines) in Figure 6.23 because these procedures are considered to exist at a level outside the ESAOA KMS.

University of Cape Town

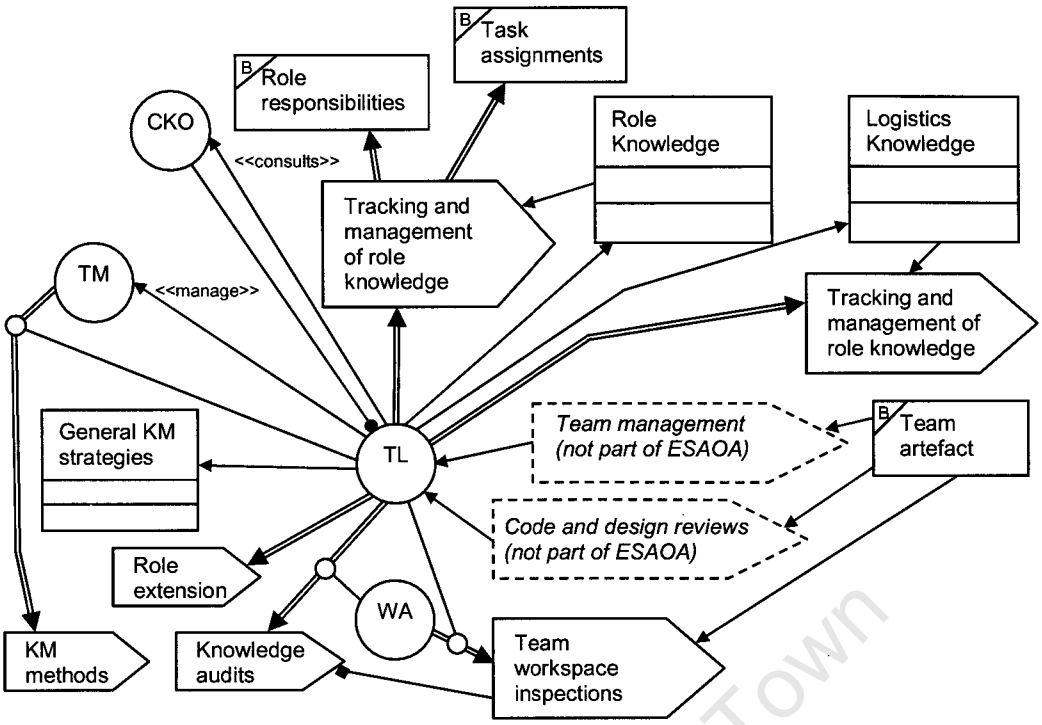


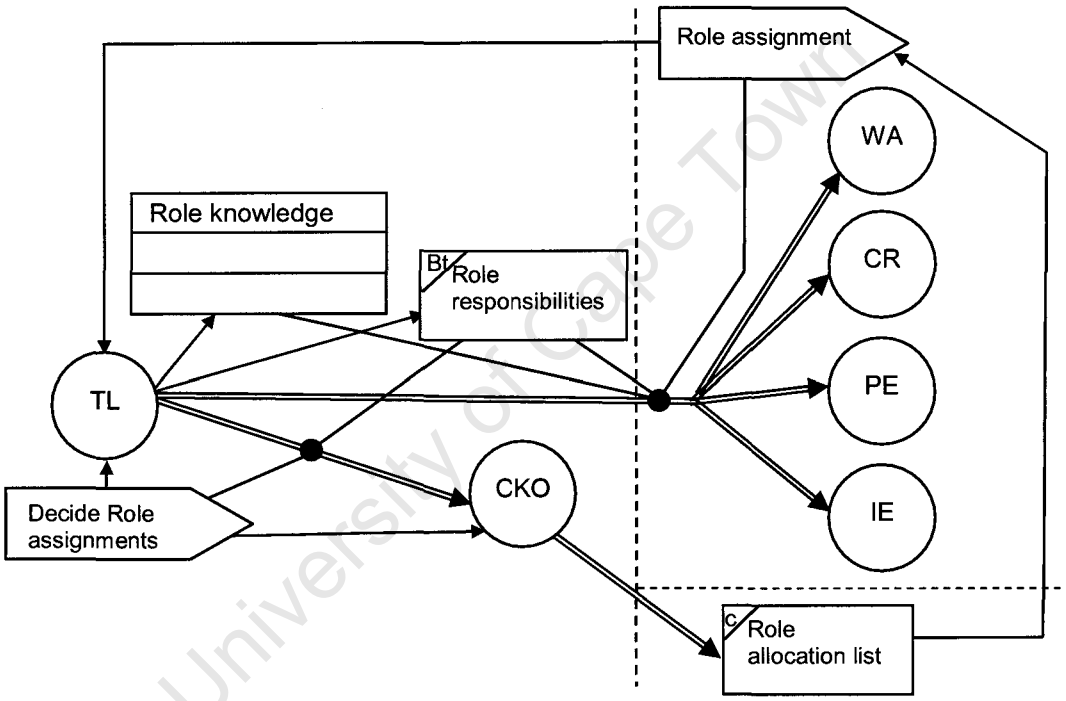
Figure 6.23: Processes performed and maintained by the TL.

The TL role for the ESAOA KMS is not equivalent to that of a TL role defined in a more generalised software development process. Rather, the ESAOA TL role describes duties that are performed by a team member who is leading a team in the use of the ESAOA KMS. The TL's role described in ESAOA KMS version 1 has been expanded to include additional process; these procedures are described below and modelled in Figure 6.24.

The TL is responsible for performing knowledge audits, which includes tasks such as ensuring that knowledge is suitably captured by the team; the WA assists in this task. The TL in turn assists the WA in performing team workspace inspections (which are usually part of a knowledge audit). The TL is also responsible for ensuring that workspace inspections are performed, and he or she is likely to delegate a large portion of this work to the WA. As shown in Figure 6.24, the CKO is available to the TL for consultation concerning the organisation of the team's workspace. The TL also needs to learn general principles of good KM strategies in order to assist members of the team on this issue; similarly, the CKO needs to be available to the TL to assist in recommending KM strategies to team members. The responsibility for tracking and managing logistics and role knowledge related to ESAOA activities are added to the responsibilities of the TL.

University of Cape Town

The TL is accountable for role assignment, for the clarification of role responsibilities, and for ensuring that duties are allocated fairly. Figure 6.24 models these processes. The TL plays a supervisory role in tailoring and expressing ESAOA processes for a team, whereas the CKO advises the TL on these tasks. The TL assesses role extension requests (i.e., additions or reallocations of role responsibilities, as discussed in Section 6.1.3.4), and administers the implementation of these requests. Boundary artefacts used in the capture and dissemination of role knowledge include the responsibilities artefact and the role allocation list. The role responsibilities artefact is in the form of an organogram or simply a textual list that indicates which responsibilities are allocated to each role in the KMS. The role allocation list indicates which ESAOA KMS roles are assigned to the team members.



University of Cape Town

Figure 6.24: Decision and allocation of roles.

As described in Section 6.1.2, roles besides the IE may discover innovations, or strategies to test a particular innovation, which may significantly enhance the production of innovation knowledge. In ESAOA KMS version 2, the IE remains the one responsible for testing innovations, but allowance is made for the other roles to suggest ideas, which are then carried out, if approved by the IE in agreement from the TL. Notice that the IE remains the ultimate decision maker in deciding which ideas to test – the TL's involvement in this issue is to ensure that ideas from other team members, especially suggestions that could be tested quickly due to a team member's insights, are considered. This aspect is modelled in Figure 6.24.

6.6.4 Processes of the component researcher (CR)

The CR is mainly responsible for researching hardware and software components for use in the development project. The duties of the CR include finding, reviewing and organising the documentation and sample code related to components. The CR is generally expected to produce the most data knowledge in a project (since Experiment 2 show that most of the projects produced more data knowledge related to components than related to tools; see Section 5.7.8). The PE and IE will rely on the CR to do a large amount of the groundwork related to knowledge of components, such as investigating the suitability of a specific component[6]. The CR assists the PE by providing component knowledge that is highly focused on the particular product being built. Figure 6.25 models the processes performed by the CR, indicating artefacts that are most commonly used and maintained by this role.

As is the case for all roles, the first process performed by the CR is training – this is modelled in Figure 6.26. During training, the CKS or CKO introduces the CR to the production and management of data knowledge.

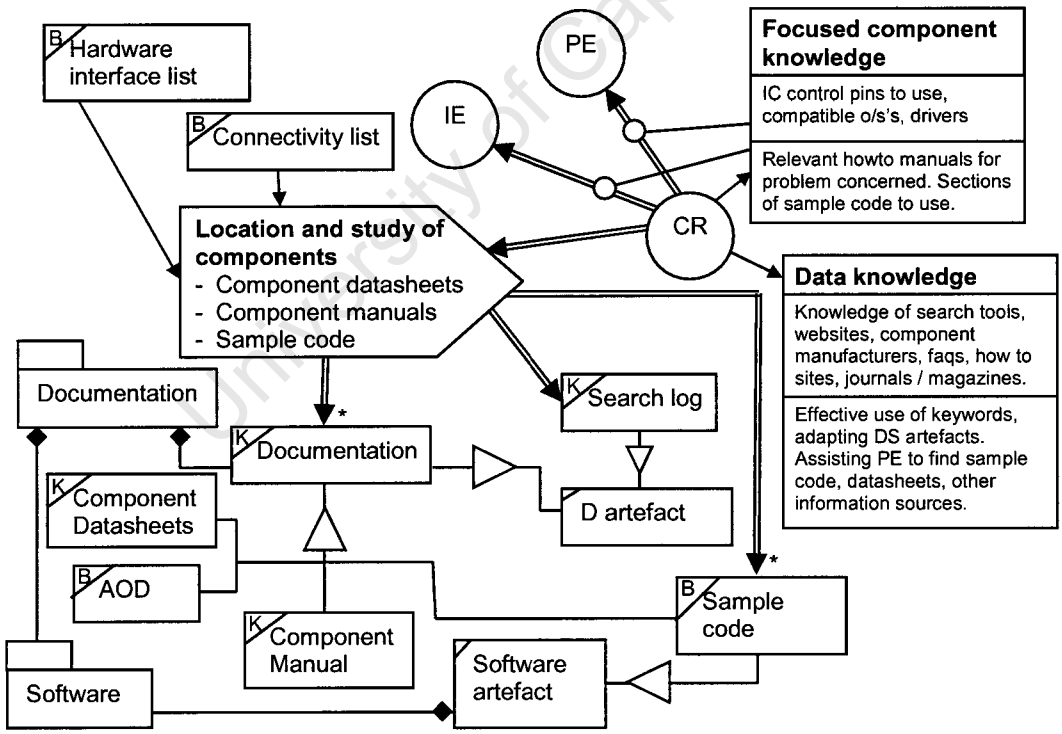


Figure 6.25: Processes of the CR.

[6] This task is often not straightforward, as it often involves checking the availability of the components, finding and downloading their datasheets and determining if the components have operating system support, among other tasks.

University of Cape Town

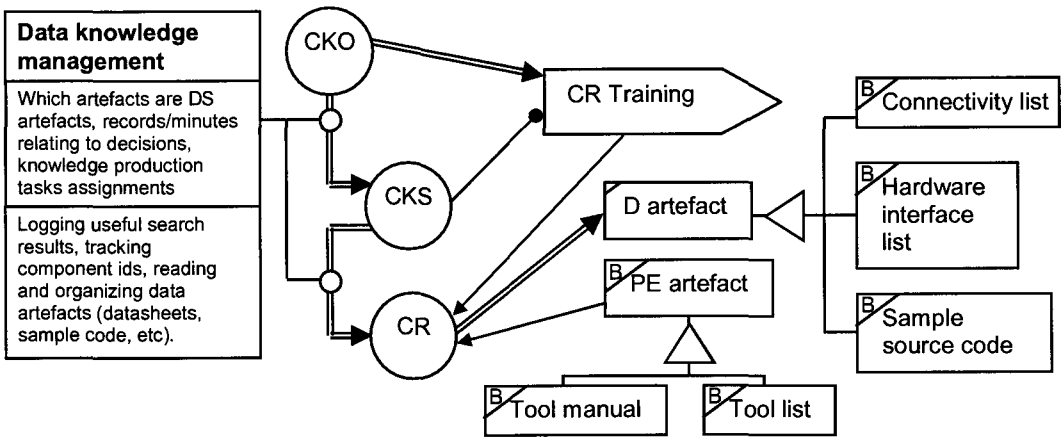


Figure 6.26: CR training process.

As indicated in Figure 6.26, the CKO maintains and documents the CR training process and teaches this process to the CKS. The CKS provides support for the CR for using the processes learned during training. Much of this training is tacit, in which the CKS shows the CR effective strategies for web searches, methods to log search text and URLs, and how and where to save datasheets and other information documents to the team workspace. The CKS also demonstrates the ESAOA tools and types of artefacts that a CR is likely to find and use during a project.

Skills required by the CR include effective search strategies (such as the use of search engine keywords) and knowledge of useful information sources (such as books and online forums) from which data knowledge can be acquired rapidly.

6.6.5 Processes of the process engineer (PE)

The PE constructs development procedures, for example, determining configuration settings for an operating system, or the steps needed to control a specific hardware component from software. ESAOA version 1 intended to shift production and management of data knowledge away from the PE to the DS. However, Experiment 2 results showed that this approach was ineffective because the PE ultimately still had to acquire large portions of a project's data knowledge[7] (see Section 5.7.7). When investigating the knowledge produced by the DS and PE in Experiment 2, it was found that the DS generally acquired data knowledge related to components, whereas the PE typically obtained data knowledge related to tools. For this reason,

[7] It is likely that the PE would find that it takes longer to explain to the DS what data knowledge is needed than simply searching for the necessary information in person.

University of Cape Town

the PE role in ESAOA KMS version 2 includes aspects of producing and managing data knowledge related to tools.

Figure 6.27 models the main processes of the PE, showing related roles and artefacts. As is shown in the model, the PE manages both hardware and software components of an ES, and he or she produces and captures process knowledge using a variety of techniques, including scripts, device drivers, sample code, how-to guides, and key steps, which are then used to transfer process knowledge to the IE.

The PE is responsible for obtaining most of the process knowledge used in a project, and is further responsible for acquiring and managing knowledge related to development tools. The CR provides the PE with highly focused component knowledge. The WA assists the PE and DS in managing data knowledge.

The first process in which the PE is involved is PE training (see Figure 6.28). During this training, the CKO and/or CKS instruct the PE, using mainly a tutorial-based approach, on effective means of creating and managing process knowledge, as well as how to acquire and manage knowledge related to development tools.

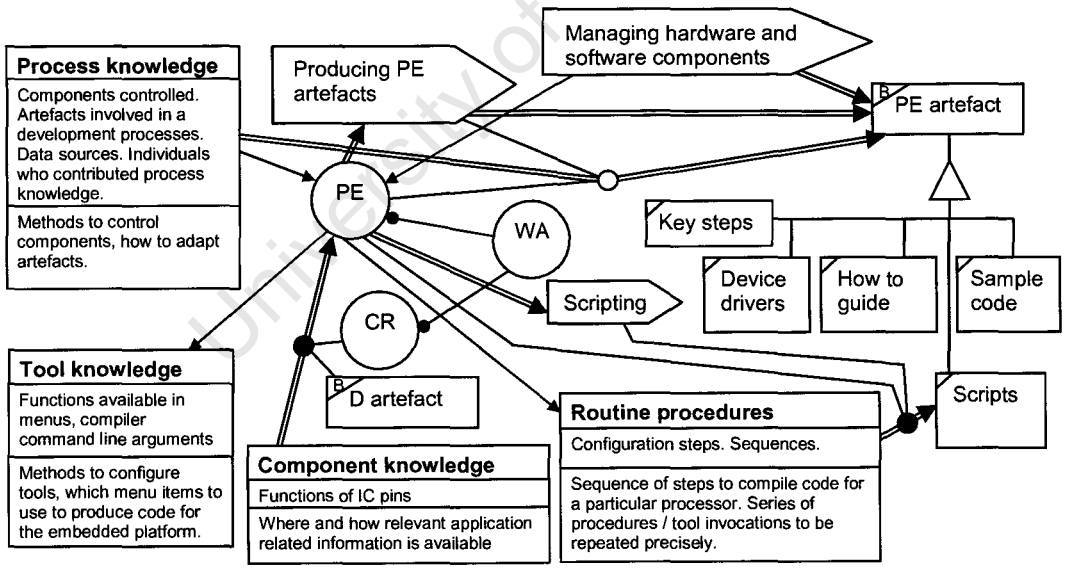


Figure 6.27: Main processes carried out by the PE.

University of Cape Town

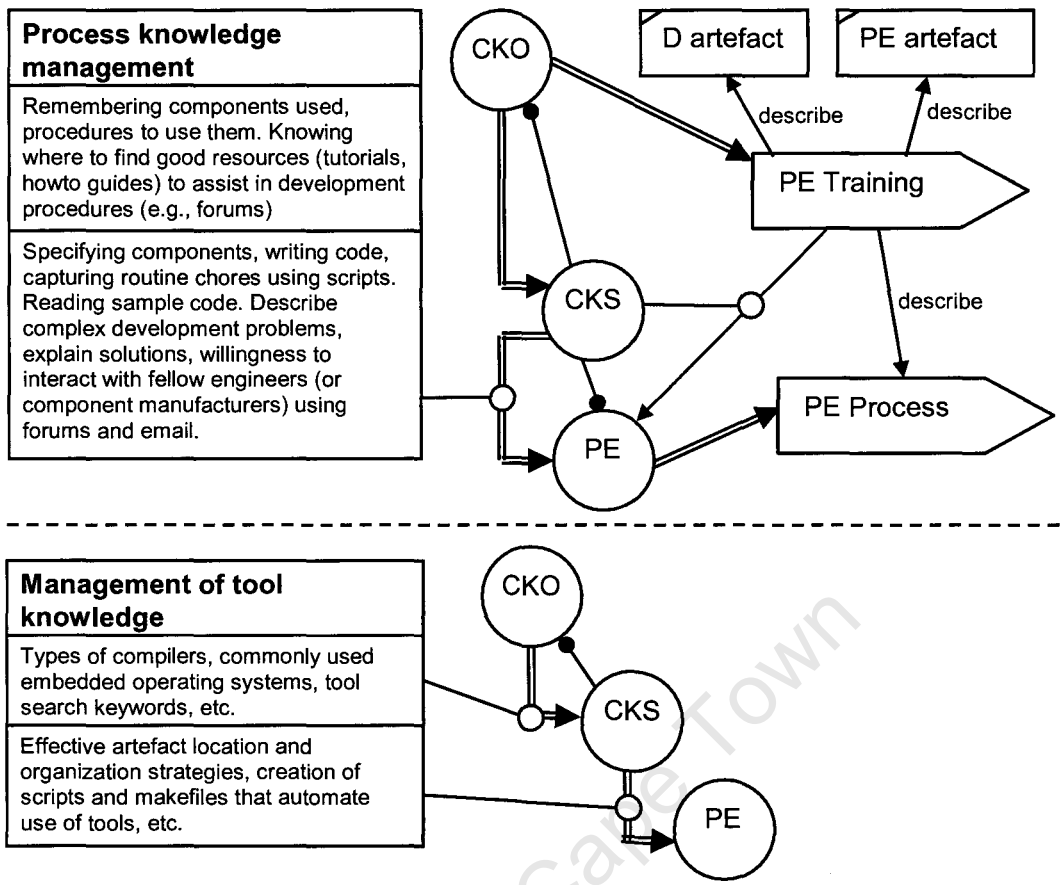


Figure 6.28: The PE training process.

The IE uses procedures created by the PE in order to test design concepts. Both the PE and IE are involved in experimentation. However, the difference between their experiments is that the PE attempts to determine methods to configure and control components and tools that will be used in the system prototype, whereas the IE focuses on testing design ideas by using processes provided by the PE and making little additional effort in terms of developing methods to configure tools or interface components. The PE requires expertise in designing and testing development methods, as well as being able to keep memory joggers and component lists that will help the PE or IE to repeat these processes at a future time. Figure 6.29 models the way in which process knowledge is transferred from the PE to the IE, with the possible inclusion of PE artefacts into the communal workspace.

University of Cape Town

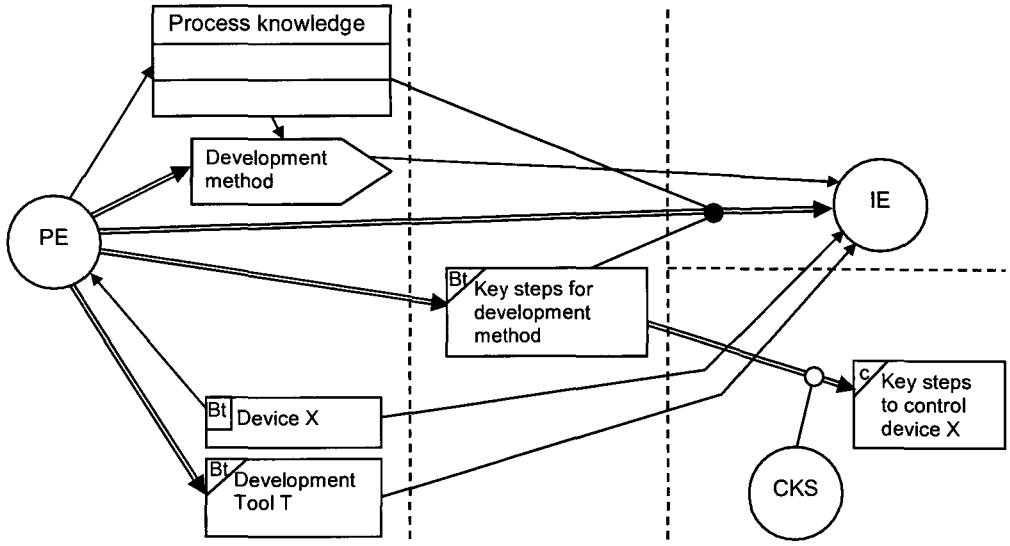


Figure 6.29: Processes involved in the interaction between the PE and IE.

6.6.6 Processes of the workspace administrator (WA)

The WA is responsible for ensuring the overall integrity and consistency of the team workspace. The WA organises and keeps track of (but does not adapt) most files related to data knowledge (e.g., datasheets and data files used for regression testing). Other roles pass D artefacts on to the WA to place into the workspace, and possibly "polish" these (i.e., add metadata, neaten layouts and update the AOD). The WA notifies the CKS of potentially useful artefacts for transfer into the communal workspace, and, if this transfer is approved by the TL, the WA works with the CKS on transferring these artefacts (other team members may need to assist with this process). Figure 6.30 models processes and role interactions concerning the WA.

The WA assists mainly the CR and PE roles in the organisation and location of artefacts in the team workplace. The WA performs quality inspection of artefacts, to ensure that they are sufficiently readable and that metadata related to them (e.g., author details and file dependencies) are suitable. The WA inspects artefacts mainly at a superficial level, as it would be counterproductive for this role to undergo the learning curves of both the CR and PE roles. The WA is responsible for setting up project directories in the team workspace, and for assisting the TL in performing knowledge audits.

University of Cape Town

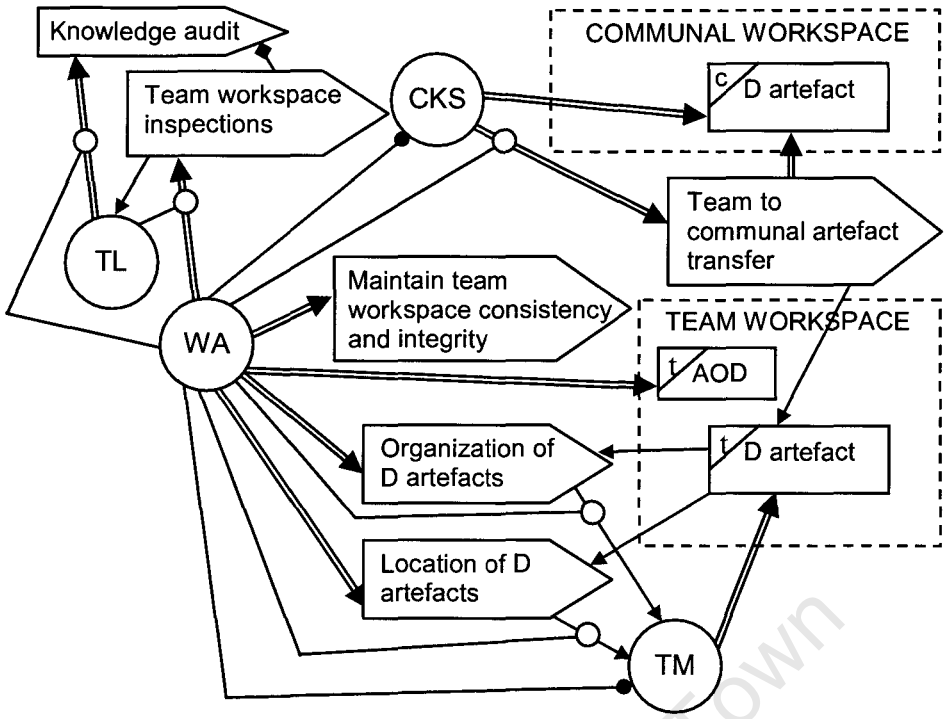


Figure 6.30: Processes and role interactions concerning the WA.

The WA should be competent in keeping records and taking notes at meetings so that important knowledge produced by the team is made explicit and retained. Consequently, the CKS and WA perform similar tasks, but for different workspaces.

6.6.7 Processes of the innovation engineer (IE)

The IE is responsible for performing experiments in which concepts are tested and innovation occurs. The PE and IE are likely to spend a significant amount of time working side-by-side, with the PE explaining development methods to the IE, and the IE identifying additional development methods (e.g., how to control a new type of component) and modifications to existing processes (e.g., optimising the control of a certain component). Most interaction between the PE and IE is likely to be rapid and face-to-face; this form of interchange is difficult to capture and record. For this reason, the PE is involved in capturing solution strategies and memory joggers that will assist in recovering or remembering development strategies. Similarly, the IE records successful solutions and key points regarding strategies used, in addition to maintaining versions of operational designs and prototypes. Figure 6.31 models the main procedures and role interactions of the IE.

University of Cape Town

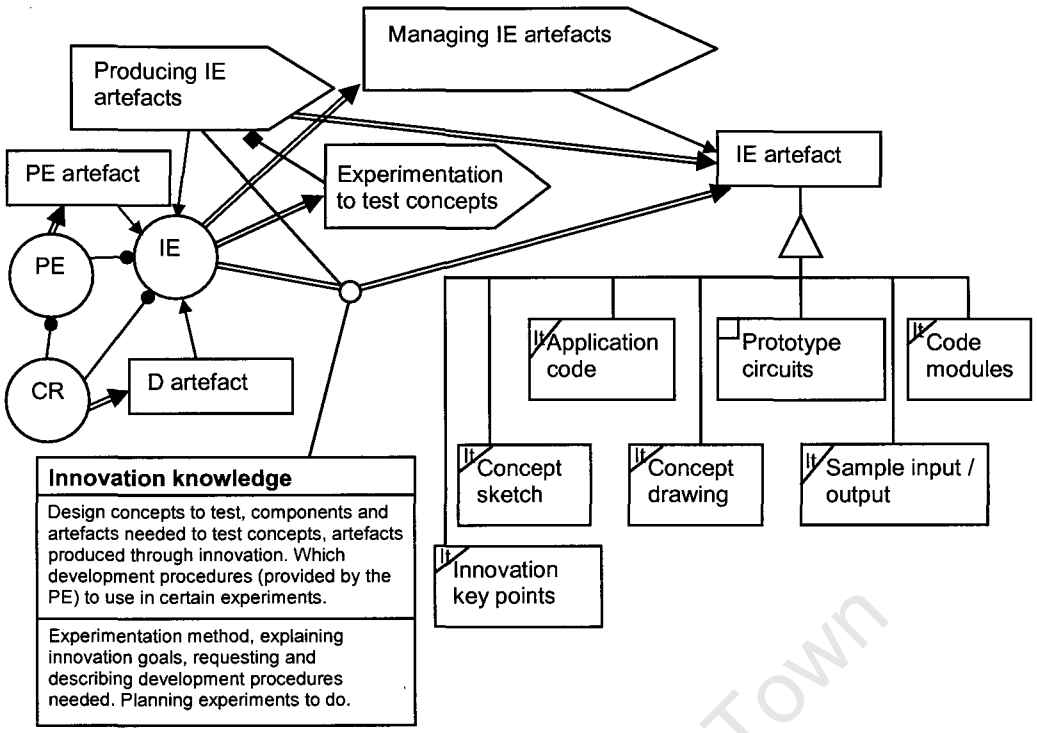


Figure 6.31: Processes and artefacts used and managed by the IE.

Figure 6.31 shows that the IE generally learns development strategies from the PE. However, the IE seldom needs to obtain the detailed process knowledge learned by the PE, such as how exactly a specific development procedure was constructed. The objective of the IE / PE relation is that the IE simply learns from the PE which scripts and configuration files to use, and which tool commands (or keystroke sequences) are relevant in the circumstances, in order to speed up the innovation process.

Innovation depends on many factors, such as an individual's prior experiences, working environment, resources, collaborators, and independence, to name a few. Consequently, the IE is the most challenging of roles to train because innovation (as discussed in Chapter 2 Section 2.5.6) is highly dependent on many complex issues. It is thus difficult to characterise explicitly or structure in the form of processes and interactions. In terms of the ESAOA KMS, the IE is trained on the broad categories, and use, of innovation artefacts (e.g., recording key points to capture design solutions), accessing and maintaining these artefacts in the ESAOA workspace, and the use of concept diagrams and other innovation artefacts to assist in knowledge exchange between the IE and other team members. The CKO is likely to perform most of the IE training (as shown in Figure 6.32); the CKS could assist the CKO and provide additional support to the IE during this training process.

University of Cape Town

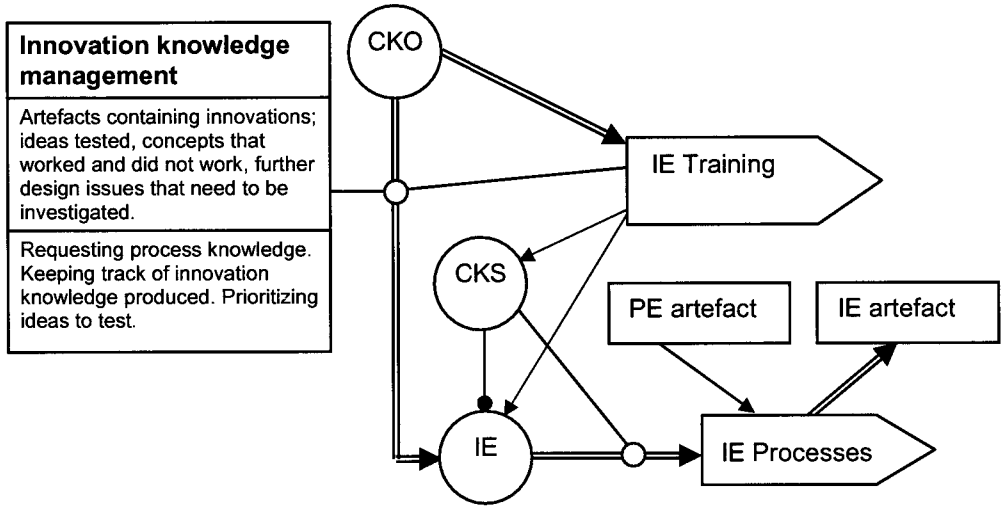


Figure 6.32: Training the IE.

The principal qualities of the IE are creativity, expressiveness and experimentation skills. The IE needs a willingness to test design ideas, coupled with the ability to explain experiment designs to the PE, who will help to establish practical development procedures needed to carry out experiments.

6.7 Artefacts

The types of artefacts used in the second version of the ESAOA KMS are summarised in Table 6.10. The first column of each row names the artefact. The second column indicates the classification of the artefact (e.g., a D artefact or a PE artefact). The third column indicates the role or roles responsible for maintaining the artefact, while the fourth column lists the roles that use it. The fifth column further designates the artefact as a boundary (B) artefact or a knowledge (K) artefact (this column is left blank if the artefact concerned is neither a boundary nor a knowledge artefact). The sixth column provides a brief description of the artefacts and, where applicable, includes a reference to an example of the artefact.

Table 6.10: Artefacts of the EMASO KMS.

Artefact Name	Classification	Main-tainer	Use by roles	B or K artefact	Description
Role responsibility list	T D artefact	TL	ALL	B	General (on-going) responsibilities for a role (e.g., WA does quality inspection of the team workspace each month)
Task assignment list	T D artefact	TL	ALL	B	A list of tasks assigned to a role
CKS responsibilities	C artefact	CKO	CKS, CKO	B	General (on-going) responsibilities of the CKS, e.g., maintaining communal artefacts.
CKS task	C artefact	CKO	CKO,	B	A specific task (or tasks) assigned to

University of Cape Town

assignment			CKS		the CKS.
KMS request form	C D artefact	CKO	ALL	B	Formal request indicating changes to be made to the KMS
KM report	C D artefact	CKS	CKO	B	Recommendations for managing a certain type of knowledge
Artefact list	D artefact	CKS, WA	ALL	K	A list of artefacts contained in a workspace or workstation
Infrastructure control list	C D artefact	CKO	CKO	K	Keeping track of information technology, computers, software tools, and other resources used in general for ESAOA activities by the organisation
Code & design review report	D artefact	CKO, TL	CKO, TL	B	Log of relevant issues observed during a code and design review that affect management of ESAOA knowledge
Components list	D artefact	CR	TM	B	A list of components used by the team (or suggested for use by the CR)
Interface control document	D artefact	PE	TM	K	The PE's (or CR's) own description of a component interface, or a description of the embedded product's interfaces
Source code	T P or I artefact	PE / IE	IE	B	Part of the embedded software for the product under development
Sample code	P artefact	PE	PE,IE	B	Either downloaded sample source code or code developed by the PE and used as part of a development process
Script	P artefact	PE	PE,IE	B	A store of automatable process knowledge
Training script	P artefact	CKO	CKS	B	Steps to carry out training
Tutorial	P artefact	CKO	CKO, CKS	B	Slides or screenshots used in training the CR and PE roles (tutorials are not used in IE training)
Concept brief	I artefact	IE	ALL	B	A short textual (e.g., one paragraph) description of a concept design to test). Usually produced before a concept model or concept design
Concept model	I artefact	IE	ALL	B	A model of an idea to test
Concept design	I artefact	IE	ALL	B	A document or diagram describing a strategy to implement a concept
Concept drawing	I artefact	IE	ALL	B	A diagram that expresses the overall concept of a product being built
Concept sketch	I artefact	IE	ALL	B	A rough or working version of a concept drawing
Concept cartoon	I artefact	IE	ALL	B	Describes the operation of a concept using a sequence of rough drawings
Member contact list	T D artefact	TL	TM	B	A list of all the team members and their contact information
Meeting agenda	T D artefact	TL	TM	B	Agenda for a team meeting
Meeting minutes	T D artefact	TL	TM	B	Minutes from a team meeting
Timetable	T D artefact	TL	TM	B	Timetable to plan when and where ESAOA activities will be performed
Component list	D artefact	CR	ALL	B	List of components used by the team
Component datasheet	D artefact	CR	ALL	K	Datasheet for a component (e.g., a microcontroller datasheet)
Component key points	D artefact	CR	ALL	B	Crucial, application specific, issues related to a component
Hardware interface list	D artefact	PE	ALL	B	List of hardware interfaces supported by a certain embedded platform
Connectivity list	D artefact	PE	ALL	B	List of software interfaces or standards supported by a platform
Tool list	P artefact	PE	ALL	B	List of tools (or equipment) needed for a

University of Cape Town

					development process; or a list of tools used by a team
Tool manual	D artefact	PE	ALL	K	A manual for a tool or lab equipment
Key steps	P artefact	PE	ALL	B	Important steps of a process
Process tutorial	P artefact	PE	IE	B	Used by PE to capture a process and to teach the IE how to perform the process
Howtos	P artefact	PE	ALL	B	Either downloaded step-by-step instructions for solving a problem, or a document developed by the PE and used to explain a process to the IE
Audit report	D artefact	TL	CKO, TL	B	Summarises the result of a knowledge audit
Sample files	D artefact	CR, WA	ALL	K	Files used in regression testing, or sample input/output data file
Workspace plan	D artefact	WA, CKS	ALL	K	A rough layout for an ESAOA workspace
AOD	D artefact	WA, CKS	ALL	B	A schematic describing an ESAOA workspace and directory structure. An example is provided in Figure 4.21
RSD	D artefact	TL, CKO	ALL	B	A role support drawing (RSD) shows how roles in the KMS are supported by other roles (Figure 6.1 is an example)
Day plan	T D artefact	TL	TM	B	Important activities to perform on a single day
Todo list	T D artefact	TL	TM	B	List of important tasks that the team needs to complete

This chapter presented the second version of the ESAOA KMS, and highlighted the changes, as well as the reasons behind these changes, which were made to the first version of the KMS in constructing the second version. The next chapter (Chapter 7) provides the conclusion for this thesis, in which the research question and hypotheses that directed this project are analysed, the contributions made by the thesis are highlighted, and plans for further work are discussed.

University of Cape Town

Chapter 7:

Conclusions and Future Work

This thesis has focused on the construction, evaluation and evolution of an experimental knowledge management system (KMS), namely the ESAOA KMS, with the intention of determining an effective structure for implementing such a KMS. As emphasised in Section 1.3, the ESAOA KMS is intended for use during ESAOA activities within the context of ES product prototyping projects carried out by novice ES engineers.

ESAOA activities (defined in Section 1.1.7) concern a subset of *meso level* KM activities that take place in ES development projects. As explained in Section 1.2.4, organisational activities occur at the macro, meso, and micro level [House *et al.*, 1995]. Rousseau & House [1994] describe meso level tasks as involving the integration of macro and micro processes; in an ES development context, this type of KM involves facilitating the way in which developers experiment with code, organise data and files, and learn how to modify components.

The rationale for this thesis (detailed in Section 1.2) can be expressed in two main reasons. The first reason is that the research literature concerning KM at the meso level is comparatively sparse compared to the literature that covers KM at the macro and micro levels of software development. Thus, this thesis was pursued to contribute to meso level KM research related to ES implementation tasks. The second reason for this thesis is to develop the ESAOA KMS as a possible approach to facilitating knowledge production during ES implementation tasks, thereby promoting the successful completion of ES projects (this can be viewed as a response to the general challenges of ES development discussed in Section 1.2, such as the fact that ES projects are often late and over budget).

The research activities concerning this thesis involved studying KM methods used by novice ES engineers and incrementally refining these methods to develop an increasingly more visible and better refined KMS that can be applied consistently by

University of Cape Town

subsequent teams of novice ES engineers. The research design involved empirical research methods, which followed an incremental approach to developing a testing the ESAOA KMS using two experiments, namely Experiment 1 and Experiment 2.

Experiment 1 involved two teams using their own *ad hoc* KM techniques. Data obtained from this experiment were first used in a preliminary study, while establishing analysis methods that built on theories from the research literature. These analysis methods were used in the KMS Analysis phase of the research design. The KMS Analysis phase was applied to the Experiment 1 data, and this was followed by the first iteration of the Framework Construction phase (during which version 1 of the ESAOA KMS was developed).

Experiment 2 involved 13 development teams, each of which used ESAOA KMS version 1 during ESAOA activities in their projects. The second application of the KMS Analysis phase was applied to data obtained from this experiment. The results were used to identify refinements to ESAOA KMS version 1, which were then incorporated into the design of ESAOA KMS version 2 (and presented in Chapter 6).

The preceding text has provided a brief synopsis of this thesis. The remainder of this chapter proceeds as follows. Section 7.1 reviews the research questions and sub-problems that guided this thesis. Section 7.2 reflects on the research findings in response to the research questions and the sub-problems. Section 7.3 summarises the main contributions of this thesis. Section 7.4 ends the chapter with plans and suggestions for future work.

University of Cape Town

7.1 Response to research questions and sub-problems

The overarching research question (as formulated in Section 1.4), which guided the research for this thesis, is reproduced below:

> **Research question:**
> What is an effective structure for the ESAOA KMS (i.e., the roles, activities, artefacts, etc.) that will contribute to the successful completion of ES implementation tasks?

This research question was broken down into a collection of sub-problems, which were used to investigate the research question from different perspectives. The sub-

problems are summarised below (Section 3.4.2 gives more detail for each one). The sub-sections that follow respond to each of these sub-problems.

1. *Identify different forms of ESAOA knowledge.* Some ESAOA activities may make heavier use of a certain types of knowledge than others do; insights into this problem can be used to create specialised KM approaches for particular kinds of ESAOA activities.
2. *Determine the relative complexity of different ESAOA KM tasks.* Identify the relative difficulty of KM tasks and highlight areas to be optimised.
3. *Establish the relative difficulty of ESAOA knowledge production associated with different ESAOA knowledge forms.* Centring on providing support for forms of knowledge that are more difficult to produce could help in optimising the ESAOA KMS design.
4. *Investigate how the time taken to complete different ESAOA activities may vary, and what factors may contribute to these variations.* Identify activities that are more complex, and that take longer to complete, in order to optimise scheduling of tasks in the KMS design.
5. *Evaluate the relative frequency of different types of problem/solution occurrences in which ESAOA knowledge is used.* Insight into the more commonly occurring KM tasks and objectives may help in planning KM tasks.
6. *Determine the proportion of ESAOA activities that are incomplete or that have been abandoned.* These findings may indicate aspects of the KMS still to be optimised.
7. *Establish how the structure of a general KMS is refined to become an ESAOA KMS.* This will be used in refining the roles, groups, activities and other aspects of the ESAOA KMS design.
8. *Determine implementation tasks that benefit the most from an ESAOA KMS.* General types of implementation tasks can be identified in order to produce higher-level guides to use in building a KMS.

7.1.1 Sub-problem 1 response: Different forms of ESAOA knowledge were identified

Experiment 1 showed that multiple forms of ESAOA knowledge occurred, specifically the broad categories of non-productive and productive knowledge. Each category was divided into the three types, namely: data, process, and innovation knowledge (see Section 4.2.7). Experiment 2 confirmed this finding, and further identified the

University of Cape Town

need for additional knowledge categories (detailed in Section 5.7.2). Data knowledge was separated into component knowledge (produced by the CR) and tool knowledge (produced by the PE). Process knowledge was divided into role knowledge (managed mainly by the TL), logistics knowledge (also maintained by the TL) and other process knowledge (mostly development methods maintained by the PE).

7.1.2 Sub-problem 2 response: The relative complexities of ESAOA KM tasks were found to differ

The KM tasks performed by the project teams were found to vary in complexity. For instance, managing data knowledge generally involved simple routines, such as placing a datasheet in an appropriate location in an ESAOA workspace and annotating the file with suitable metadata. Management of process knowledge was generally more difficult. For example, capturing automatable process knowledge in the form of an executable script involved coding the script, placing the script in a suitable location and perhaps adding classification metadata to assist in recording the purpose of the script.

7.1.3 Sub-problem 3 response: Difficulty of producing different forms of ESAOA knowledge varied

In all the Experiment 2 projects (as discussed in Section 5.2 with regard to the individual projects), the production of innovation knowledge usually involved the IE setting up a complex experiment (such as the experiments of Project P2-1 to test a GPS module, and Project P2-3 to test wireless communication). During these experiments, the IE applied both process knowledge and data knowledge in order to test possible design options and to create innovation knowledge. Process knowledge was also commonly obtained from doing experiments, but these experiments were usually simpler and more clearly focused than the experiments done to obtain innovation knowledge.

Furthermore, innovation experiments tended to integrate a variety of development methods and data knowledge, which drew on prior learning done earlier in the project, whereas experiments to obtain process knowledge tended to depend less on prior learning. Consequently, activities performed by the IE to complete experiments needed to obtain innovation knowledge were among the most challenging and time-consuming activities performed in the projects studied. In comparison to innovation knowledge production, process knowledge was produced more easily and through less complex experiments and related activities. Data knowledge tended to be

University of Cape Town

simpler to obtain than process knowledge, in that it mainly involved reading documents (as per the data knowledge definition given in Section 4.2.7) to obtain specific information needed by the PE or IE.

In most Experiment 2 projects, innovation knowledge occurrences tended to occur more frequently towards the end of the project. This was generally not because it became easier to complete innovation experiments, but rather because the teams had changed focus towards achieving a prototype that was as complete as they could manage by using process and data knowledge already accumulated (and if necessary, sacrificing features and performance due to lacking the required know-how). An example of this situation was seen in Experiment 2, when (at event chain 80) the Project P2-2 team attempted to optimise their development methods (thus building more process knowledge); however this attempt was soon abandoned because much of the prototype functionality was missing and time was running out.

The relative complexity for different types of knowledge production found in Experiment 2 can be generalised as follows:

- Innovation knowledge production activities were generally more complex than those for producing process knowledge (requiring inventive experimentation that drew widely on process and data knowledge obtained earlier in projects).
- Process knowledge production was generally more complex than data knowledge production (involving experiments that drew on both process and data knowledge, but not integrating as much prior knowledge as experiments for innovation knowledge production did).
- Data knowledge production usually involved the easiest types of knowledge production activities (mostly reading text and not requiring any experiments). But *productive* data knowledge was not necessarily always the *easiest* knowledge to obtain; as Section 5.3 indicates, Experiment 2 projects on average produced the highest portion of non-productive data knowledge.

Note that the points above are generalisations based on Experiment 2 results. The results did not demonstrate that each point above was always the case. Moreover, there were numerous situations where these points were invalid. For instance, activities to produce innovation knowledge were sometimes easier than activities to produce process knowledge – one such example occurred in Project P2-1, where the team were engaged in challenging experiments to learn to use uCLinux ramdisks

University of Cape Town

(i.e., process knowledge), but once this had been accomplished, they proceeded with a sequence of comparatively simple experiments to build innovation knowledge.

The response to this third sub-problem can be summarised as follows: there is variation in the difficulty of producing different forms of ESAOA knowledge. Innovation knowledge production frequently involved more complex activities than did the other types of knowledge, and similarly, activities to obtain process knowledge were usually more difficult than those required for producing data knowledge. But this was not consistently the case; there were many counter examples. Generally, more innovation knowledge was produced towards the end of projects, but this did not imply that it had become easier to produce this type of knowledge. Furthermore, no evidence was found to show that activities for data or process knowledge production became easier during the course of a project. It was found that teams often changed tactics towards the end of a project to focus on producing innovation knowledge, using only the process and data knowledge they already had, without starting new activities to obtain more process and data knowledge (this is probably due to teams realising that these activities did not necessarily become easier as the project progressed, and thus deciding they did not have enough time to complete new activities in the last days of a project).

7.1.4 Sub-problem 4 response: the time to complete ESAOA activities depends on their complexity, their dependence on other activities and the provision and understanding of KMS support

Experiments 1 and 2 showed that ESAOA activities typically occur as part of an event chain, in which the completion of one activity may depend on the successful completion of multiple other activities. In the projects studied, it was commonly seen that to complete a particular ESAOA activity, the engineer working on the activity often needs to recall or reconstruct knowledge produced in other, previously completed event chains.

In Section 4.2.4, the terms *trivial solution cycles* and *non-trivial solution cycles* were introduced. For the purpose of this argument, the complexity of an ESAOA activity is expressed in relation to the types of solution cycles involved in the activities concerned. Activities at a low level of complexity (or *simple activities*) involve only trivial solution cycles; these activities are largely mechanical, need little creativity, and do not require the knowledge worker to bring together knowledge obtained from a wide range of prior learning done earlier in the project (examples are reading

University of Cape Town

through documentation to find particular facts). Activities at a high level of complexity (or *complex activities)* are the antithesis of simple activities. *Complex activities* include non-trivial solution cycles, and require some proactive and intuitive work that brings together knowledge from a wide range of prior learning done in the project. This classification of activities builds on Section 4.2.4 and is based on research related to the classification of problem-solving tasks, specifically following techniques of classifying computer interactions in a similar manner to phases ranging from 'phase one' actions based on prior knowledge (closely corresponding to the notion of *simple activities*) to 'phase four' actions, where new knowledge is obtained and existing knowledge is "... reorganised to incorporate compound plans" [Sternberg & Frensch, 1991, pg 323]. Additionally, these classifications were influenced by deconstructing "complex problem solving in electronics" [Sternberg & Frensch, 1991, pg 287] and by Silvestro *et al.*'s [1992] categorisation of service processes.

In Experiment 2, the successful completion of ESAOA activities, and the knowledge resulting from them, was found to be strongly dependent on the level of complexity of the activities and their interrelatedness. Simple activities (involving less knowledge events) were generally completed more quickly than complex activities. Simple activities usually did not directly cause dead-ends – these activities were therefore commonly considered as productive when viewed in isolation (although they could later be reclassified as non-productive in relation to changes in higher-level objectives). Complex activities, in contrast, usually involved many knowledge events (and took longer to complete) than simple activities. Dead-ends, which resulted in event chains being classified as non-productive, were commonly caused by developers being unable to complete complex activities, often due to time limitations. Section 5.2 regularly noted the abandonment of project objectives (causing event chains to become dead-ends), such as Project P2-8 members trying to use a Bluetooth module and then abandoning it, and Project P2-11 developers being unsuccessful in sending messages from a T10 cell phone and later exchanging it for an alternate component.

Developers encountered a variety of difficulties while using the ESAOA KMS (detailed in Section 5.4.2.1), these included: 1) difficulties in learning the directory structure of ESAOA workspaces, 2) problems in understanding how to compile programs in a workspace, 3) uncertainty about adding and placing files in workspaces, and 4) ESAOA support tools taking too long to learn, which in some cases resulted in team members ignoring some of the tools supplied, and either

University of Cape Town

making their own equivalents or resorting to manual solutions. These problems resulted in the KMS being applied ineffectively in some cases, sometimes causing ESAOA activities to be abandoned or taking a long time to complete. In some situations, the ESAOA KMS made the developers' work more difficult, even distracting them from critical work. In Section 5.6.2.1, examples of this kind of distraction were given; one such example was the case when a team member spent valuable time deciphering and responding to cryptic error messages reported by an ESAOA support tool, before being able to proceed with implementation tasks.

In Project P2-2, a difficulty was noted where team members did not understand the ESAOA KMS role responsibilities that they had been assigned. During the first part of this project, the team members were unsure of what to do, which led to poor coordination of teamwork and duplication of ESAOA activities. During this period, the team also decided not to use most of the ESAOA support tools or to adhere to the ESAOA workspace directory structure: instead, they kept all their project files in one folder (although they did later reorganise their files into the requisite directory structure). In effect, this team was unable to use the ESAOA KMS effectively in the first part of their project. Section 5.2.2 reported that this team had a high occurrence of non-productive data knowledge compared to the other teams studied in Experiment 2. The data knowledge occurrence graph for Project P2-2 showed that much of this non-productive data knowledge happened early in the project, which closely corresponds to the period when the team experienced role responsibility problems and were making little use of the ESAOA KMS.

In Section 5.7.5, concerning variables that influenced knowledge production in Experiment 2, it was observed that simple activities are likely to produce little innovation knowledge, and that leadership and understanding of assigned roles improves the proportion of productive knowledge produced by a team.

Based on the preceding argument, the response to sub-problem 4 can be summarised as follows (in respect to the context of Experiment 2): 1) complex ESAOA activities tend to take longer to complete than simple ones; 2) ESAOA activities that are highly dependent on knowledge obtained previously in the project (i.e., during earlier knowledge-producing ESAOA activities) are more likely to take longer to complete, result in higher proportions of non-productive knowledge, and end unsuccessfully, than ESAOA activities that are less dependent on prior knowledge; and 3) ineffective use of the ESAOA KMS and difficulty in using its

University of Cape Town

support tools and underlying workspace structure can lead to higher proportions of non-productive knowledge.

7.1.5 Sub-problem 5 response: developers encounter similar types of ESAOA KM problems and solutions in different projects

In both Experiments 1 and 2, the different teams dealt with similar KM problems, for which comparable solutions were applied. Section 5.4.2.2 presented results from questions concerning knowledge production methods (described in Section 3.8.1) that had been asked of Experiment 2 teams during design review 2.

Almost all Experiment 2 project teams (except Project P2-6, which was the least innovative) started by acquiring data knowledge related to components and tools. This was seen in knowledge occurrence graphs and the discussion of individual projects given in Section 5.2, as well as in the reflections presented in Section 5.7.4 concerning the 'progression towards innovation'. It was found that all teams used similar methods and information sources to produce data knowledge; as listed in Section 5.4.1.2 (and also discussed in Section 5.7.4), common techniques included: web searches, reading datasheets, using manufacturers' and retailers' websites, and online forms. The teams used many of the same methods to organise and adapt artefacts, such as saving downloaded files and web pages into a shared folder, adding comments to downloaded files, using bookmarks, highlighting sections of documents and renaming/relocating files.

In both Experiment 1 (Sections 4.4 and 4.6.4) and Experiment 2 (Sections 5.7.4.3 and 5.5.5), it was found that process knowledge typically built on data knowledge, and that process knowledge decided most data knowledge needs. Different teams used similar approaches to manage process knowledge, such as adding comments and constants values to record settings and parameter options in code files, noting key steps in text files or logbooks, and creating scripts to record settings (Section 5.4.1.2 lists other methods used).

Managing innovation knowledge involved a more varied range of techniques between different teams. The methods listed in Section 5.4.1.2 for innovation knowledge production show that most teams *produced* innovation knowledge through experimental approaches that involved applying previously learned process and data knowledge. Techniques to capture this knowledge varied; for example, some teams established their own methods, such as using concept cartoons, installation diagrams

University of Cape Town

and types of diagrams (examples of which were included in Section 5.2). At a higher level, the innovation knowledge was produced in similar manners, such as: using rough notes or drawings (e.g., concept cartoon) to express a plan; performing experiments (usually trying multiple approaches before deciding on a suitable solution); preparing sample input/output files; and recording key steps of innovation knowledge.

In summary, similar types of techniques were used by the development teams to manage their knowledge. Across the different types of knowledge categories, some of the most common KM methods included: editing code files, making rough sketches, adding comments, adding notes in logbooks and highlighting documents. While ESAOA support tools (e.g. *fclass* and *esaoa-find*) were used to some extent, they were used less frequently than conventional methods (e.g., editing code files). The design of ESAOA KMS version 2 is guided by these similarities observed in KM problems and in the solution methods used by the teams.

7.1.6 Sub-problem 6 response: although dead-ends did occur in ESAOA knowledge production, their number was reduced

Experiment 2 included cases of event chains that resulted in a dead-end (and that were therefore classified as producing non-productive knowledge). In Experiment 2, 75% of knowledge occurrences were on average productive (see Section 5.3 for averaged results). In comparison, and using the same analysis technique, only 36% of knowledge occurrences in Experiment 1 were on average productive (see Section 4.5.3.1). This comparison shows an overall reduction in non-productive knowledge occurrences (i.e., event chains ending in dead-ends) in Experiment 1 compared to Experiment 2.

7.1.7 Sub-problem 7 response: the degree to which the ESAOA KMS is used depends on the complexity of ESAOA activities concerned

Teams that generally found the solution of an ESAOA activity simple, or even trivial, made little or no use of the ESAOA KMS for managing knowledge produced during that activity. For example, the process engineer in Project P2-1 found that sending characters to the embedded Linux console was trivial and therefore did not apply any KM methods to handle this process knowledge. The ESAOA KMS was used more frequently while solving complex ESAOA activities, especially with regard to managing innovation knowledge. For instance, Project P2-1 used the KMS

University of Cape Town

extensively during the final stages of the project to facilitate the transfer of process knowledge related to a GPS device, from the PE to the IE.

Artefacts provided in baseline ESAOA workspaces were used extensively by Experiment 2 teams, and many teams modified or added their own scripts related to process knowledge (this was noted in Sections 5.8.1 and 5.4.1.3 respectively in terms of teams using ESAOA support tools and adding to these). The ESAOA workspaces, ESAOA support tools, and the integrated knowledgebase were used largely implicitly (rather than consciously following step-by-step procedures in the ESAOA manual) while solving problems during ESAOA activities. The ESAOA KMS was also used during meetings between the researcher and team members, and in meetings between the CKO and CKS.

7.1.8 Sub-problem 8 response: benefit of the ESAOA KMS depends on the complexity, difficulty and duration of the activities performed

As discussed in relation to sub-problems 4 and 7, the completion of an ESAOA activity depends on the complexity of the activity and the ease of using the KMS during that activity. It was found that many of the simple ESAOA activities (that were either trivial or completed quickly) made little use of the ESAOA KMS role structures and processes, apart from perfunctory use of ESAOA workspaces (in which all the files were stored) and basic ESAOA support tools (especially the use of *esaoa-mm* to update makefiles automatically, *fclass* to change metadata, and *esaoa-cp* and *esaoa-mv* to copy and move files together with their metadata). Longer and more complex activities often made wider use of the ESAOA KMS design aspects. For example, Project P2-1 team members used the KMS during the development of control code for the GPS device, and during the division of work according to roles, and in particular the transfer of process knowledge (using scripts and logs) and innovation knowledge (using diagrams and sketched models). Generally, the more complex ESAOA activities benefited from more structured approaches (and wider use of the ESAOA KMS) as they helped to reduce the level of confusion and avoid misplacement of knowledge artefacts.

While the sub-problems for this thesis have been addressed, there are a number of issues related to the conditions referred to above, and considerations raised by the experiments, that require further study. These are discussed in the next section.

University of Cape Town

7.2 Reflection of research findings and resolution of research question

The responses to the sub-problems generally highlighted areas in which the ESAOA KMS was found to be beneficial, as well as conditions where the KMS was of little use or where it may even have added to the difficulty of completing ESAOA activities and related implementation tasks. Generally, the ESAOA KMS was of the least use during *simple* activities (i.e., where the needed knowledge was either trivial or was produced following easily remembered or routine procedures). No refinements were therefore implemented in ESAOA KMS version 2 to give additional support for these types of activities. However, teams did make extensive use of the ESAOA KMS when seeking to accomplish *complex* activities that involved complicated knowledge-based work that drew on a wide range of knowledge developed during prior solution strategies done in the project. The ESAOA KMS was found to be beneficial in such contexts, in particular, with regard to the following: improved conformity of artefact location and classification by means of the ESAOA workspaces, use of ESAOA support tools, role assignments to assist with allocating team member responsibilities, division of knowledge work, and guiding knowledge production.

Without the underlying workspace-based framework of artefacts and tools provided with the ESAOA KMS, the set of roles and role responsibilities, and the training programme and documentation, it is unlikely that the ESAOA KMS would be used effectively (and teams would probably revert to establishing their own *ad hoc* techniques due to a lack of understanding how to use the KMS).

Overall, and in response to the main research question that guided this thesis, the study has demonstrated that the ESAOA KMS comprised an effective structure (detailed in Section 4.6) that was shown to facilitate knowledge production performed by novice engineers during ESAOA activities, and thereby aided in the successful completion of ES implementation tasks in Experiment 2. Specifically, an effective structure for the ESAOA KMS was found to comprise the following components:

- An ESAOA workspace (separated into personal, team and communal parts) that presents a consistently organised collection of artefacts, which are accessed and worked on by team members that each use a similar ESAOA workstation that comprises a similar collection of 'external' development tools (i.e., tools that are not stored in an ESAOA workspace). Each workspace

University of Cape Town

incorporates a searchable knowledge base composed of metadata (filenames and file classifications stored in CSV files), code comments, and other CSV files containing recorded keywords and descriptions.

- ESAOA support tools, comprising Bash scripts and C++ programs (built around the KIT API descried in Section 6.3.4) facilitate use of files in the workspaces (these support tools are stored in the ESAOA workspaces).
- A baseline collection of digital artefacts (i.e. files such as documentation and template files) for each ESAOA workspace, which are consistently arranged into the same (documented) directory structure.
- A set of roles with clearly defined responsibilities.
- A description of ESAOA KM processes, which provide strategies that are intended to guide the roles, and to suggest role interactions, to facilitate managing and producing knowledge in ESAOA activities.
- A knowledge ontology that describes specialised terms related to the KMS, and that defines artefact classifications used to classify files in ESAOA workspaces. A team can build a customised version of this ontology, which is stored in the team workspace.
- A conceptual modelling language that visually documents and explains aspects of the KMS and KM strategies.

Findings from the application of the ESAOS KMS in Experiment 2 and conclusions discussed in Sections 5.7 and 5.8 suggest a number of strategies that could be followed and built upon in future research projects. These included the use of proportions of knowledge occurrences to asses the effect of KM in projects, categorisation of productive and non-productive knowledge to asses knowledge work, and the 'progression to process' and 'progression to innovation' (see Section 5.7.4.3) as measurement of the trajectory for the success of a project. Evaluation of projects based on proportions of innovation, process and data knowledge produced, which can show performance in terms of knowledge production, gives a perspective related to 'knowledge deliverables' [Drove, 1999] that could supplement the necessary perspective on 'product deliverables'.

Refinements that are expected to provide an improved design for the ESAOA KMS are detailed in Chapter 6.

University of Cape Town

7.3 Summary of contributions

This thesis focused on integrating a KMS into ES product development projects to manage ESAOA knowledge. In order to accomplish this objective, the knowledge needs of ES engineers in the context of ESAOA activities were investigated [Winberg, 2006d] and an in-depth study of the application and production of ESAOA knowledge was performed. Unique contributions made by this thesis include:

1. Development of a KMS to assist implementation aspects of ES engineering;
2. A methodology for researching and evolving a KMS for ES engineering Winberg & Schach, 2007];
3. Identification of ES engineering tasks that involve significant amounts of knowledge acquisition and that are likely to benefit from KM methods [Winberg *et al.*, 2008];
4. Contribution of an in-depth understanding of the knowledge economy within the particular sector of ES engineering in terms of defining knowledge work processes, knowledge bases, and methods for managing knowledge in ES product development.
5. Identifying the values of evolving a KMS for ES engineering.

7.4 Future work

Implications for future work are discussed in the subsections below. In some cases these 'gaps' were evident in the literature, while in other cases, they were raised by the study itself.

7.4.1 Testing ESAOA KMS version 2

A framework for ESAOA KMS version 2 was developed based on the results of Experiment 2. This revised version of the KMS needs to be tested with another set of ES product development projects. Furthermore, the evaluation strategy used in this future testing could incorporate broader issues related to team member opinions and effects on organisations as a whole. For example, the happiness of knowledge workers [Rahe & Morales, 2005] using the KMS could be measured and contrasted with individuals using only *ad hoc* KM methods and no formalised KMS.

7.4.2 Testing ESAOS KMS on different type of ES engineering

Different types of ES engineering projects are each likely to require a specialised KMS: for example, a KMS for a product development project would probably differ

University of Cape Town

from that of a KMS for a prototyping project. Customer support systems for an ES development company may need a suitably adapted KMS that is not identical to the one used by developers in the company (e.g., KMS that provides a means for supporting staff to access technical engineering knowledge in order to assist clients in solving problems). ES-related services, such as chip manufacturers and embedded operating system vendors, may benefit from another customised KMS.

7.4.3 Phasing in a KMS within existing/ongoing projects

The ESAOA KMS was trialled under laboratory conditions. In a commercial context, a KMS would need to be phased in with minimal disruption to existing work. Further research should be done to understand how a comprehensive KMS could be effectively phased into an ongoing development project.

7.4.4 Broadening the context for the ESAOA KMS

In both experiments, the KMS was studied in the context of the implementation phase of ES product development projects. Additional testing is needed in which the KMS is extended to other phases (e.g., the requirements phase and the maintenance phase) of product development.

7.4.5 A KMS that allows for future software and hardware developments

A KMS for ES needs flexibility and adaptability to cope with expected further advancements in hardware and software technologies (with regard to Moore's law). This aspect could be considered during construction of ESAOA KMS version 3.

7.4.6 Need for further research into KM in ES development

The literature review identified a general lack of knowledge strategies focused on ES development projects. This finding motivates the need for further research in this area of engineering.

7.4.7 Focus on ES innovation knowledge

Where the literature did address KM related to ES development, the focus tended to be on roles, data, information, the identification of needs, and systems to manage processes. Notably missing from the literature was the area of innovation knowledge and the management of innovation systems. Innovation knowledge emerged in this study as particularly important for ES product development, and this is clearly an area, which requires further research.

University of Cape Town

University of Cape Town

References

Aamodt, A. & Plaza, E. [1994]. "Case-based reasoning: foundational issues, methodological variations, and system approaches." AI Communications, 7: 39-59.

Abell, A. & Oxbrow, N. [1999]. "Make knowledge management work: CKO, CKT, or KT?" In Liebowitz, J. (ed.) Knowledge management handbook. New York, NY.: CRC Press: 4.1-4.17.

Abrahamson, P., Warsta, J., Siponen, M., Ronkainen, J. [2003]. "New directions on agile methods: a comparative analysis." Proceedings of the 25th international conference on Software Engineering (ISCE'03). Portland, OR. 3-10 May 2003: 244-254.

ActiveState. [2007]. ActiveTcl: the industry-standard Tcl distribution. Available at http://www.activestate.com/activetcl/ (Accessed 4 Apr 2007).

Addison, T. & Vallabh, S. [2002]. "Controlling software project risks: an empirical study of methods used by experienced project managers". Proceedings of the 2002 annual research conference of the South African institute of computer scientists and information technologists on Enablement through Technology (SAICSIT '02). Port Elizabeth, South Africa 16-18 Sep 2002: 128-140.

Alavi, M. & Leidner, D. [1999]. "Knowledge management systems: issues, challenges, and benefits." Communications of AIS, 1(2): 2-36.

Alavi, M. & Leidner, D. [2001]. "Review: knowledge management and knowledge management systems: conceptual foundations and research issues." MIS Quarterly, 25 (1): 107-136.

Allee, V. [1997]. The knowledge evolution. Boston, MA.: Butterworth-Heinemann.

Allen, M. [2000]. "Linux inter-process communication". In The CTDP Linux programmer's guide version 0.3.0. Available at http://www.comptechdoc.org/os/linux/programming/linux_pgcomm.html (Accessed 5 Mar 2007).

Althoff, K., Birk, A., Hartkopf, S., Muller, W., Nick, M., Surmann, D. & Tautz, C. [1999]. "Managing software engineering experience for comprehensive reuse." 11th international conference on software engineering and knowledge engineering (SEKE'99). Kaiserslautern, Germany 17-19 Jun 1999: 25-50.

Altshuller, G. [2004]. (trans L. Shulyak). And suddenly the inventor appeared: TRIZ, the theory of inventive problem solving (6th edition). Worcester, MA.: Technical Innovation Center, Inc.

Altshuller, G. [1998]. (trans A. Williams). Creativity as an exact science: the theory of the solution of inventive problems (4th edition). New York, NY.: Gordon and Breach Science Publishers.

Aman, H., Okazaki, H. & Yamada, H. [2006] "An effect of comment statements on code corrective maintenance". In Tyugu, E. & Yamaguchi, E. (eds.) Knowledge-based software engineering: proceedings of the 7th Joint Conference on knowledge-based software engineering. Amsterdam, The Netherlands: IOS Pres: 135-138.

Andersen, E. [1996]. "Warning: activity planning is hazardous to your project's health!" International Journal of Project Management, 14: 89-94.

University of Cape Town

Anderson, J. & Bower, G. [1980]. Human associative memory: a brief edition. Hillsdale, NJ.: Lawrence Erlbaum.

Apple. [2009]. Apple iPhone features. Available at http://www.apple.com/ph/iphone/ (Accessed 10 Jan 2009).

Apshvalka, D. & Grundspenkis, J. [2005]. "Personal knowledge management and intelligent agent perspective." In Nilsson, A.G. (ed.) Proceedings of the 14th international conference on information systems development, Pre-Conference-ISD 2005. Karlstad, Sweden 14-17 Aug 2005: 219–230.

Argote, L. & Ingram, P. [2000]. "Knowledge transfer: a basis for competitive advantage in firms." Organizational Behavior and Human Decision Processes, 82(1): 150-169.

Arias, E. & Fischer, G. [2000]. "Boundary objects: their role in articulating the task at hand and making information relevant to it." Proceedings of the International ICSC Symposium on Interactive and Collaborative Computing (ICC'2000). Sydney, Australia 11-15 Dec 2000. Wetaskiwin, ICSC Academic Press: 567-574. Available at http://13d.cs.colorado.edu/gerhard/papers/icsc2000.pdf (Accessed 06 Feb 2010).

Arlow, J. [2005]. UML 2 and the unified process: practical object-oriented analysis and design. Upper Saddle River, NJ.: Addison-Wesley.

ATMEL. [2005]. AT91RM9200 Microcontroller. Available at http://www.atmel.com/dyn/products/product_card.asp?part_id=2983. (Accessed 20 Mar 2005).

Aurum, A., Handzic, M., Wohlin, C. & Jeffery, R. [2003]. Managing software engineering knowledge. Berlin, Germany: Springer.

Bakshi, A., Prasanna, V.K. & Ledeczi, A. [2001]. "MILAN: a model based integrated simulation framework for design of embedded systems." Proceedings of the 2001 ACM SIGPLAN workshop on Optimization of middleware and distributed systems, Snow Bird, Utah 22-23 Jun 2001: 82-93.

Balasubramanian, K., Balasubramanian, J., Parsons, J., Gokhale, A. & Schmidt, D.C. [2005]. "A platform-dependent component modeling language for distributed real-time and embedded systems." Proceedings of the 11th IEEE Real-time on Embedded Technology and Applications Symposium, San Francisco, CA.: 07-10 Mar 2005: 190-199.

Ball, S. [2002]. Embedded microprocessor systems: real world design. Boston, M.A.: Newnes.

Barr, M. [1999]. Programming embedded systems in C and C++. Sebastopol, CA.: O'Reilly.

BASH. [2009]. BASH - GNU Project. Available at http://www.gnu.org/software/bash/. (Accessed 12 Mar 2009).

Basili, V.R., Caldiera, G. & Rombach, H. [1994]. "The experience factory." In Marciniak, J. (ed.) Encyclopedia of software engineering. New York NY.: John Wiley: 469-476.

Baskiyar, S. & Meghanathan, N. [2005]. "A survey of contemporary real-time operating systems." Informatica, 29: 233-240.

Beck, K. [1999]. "Embracing change with extreme programming." Computer, 32(10): 70-77.

Bellinger, G. [2000]. Knowledge management: emerging perspectives. Available at http://www.outsights.com/systems/kmgmt/kmgmt.htm. (Accessed 12 Dec 2007).

Bendix, L. [1993]. Programming-in-the-large versus programming-in-the-many. Informatik Berichte Nr. 93-05, Fachgruppe Informatik, Universität-GH-Siegen, Germany.

University of Cape Town

Available at http://www.cs.lth.se/home/Lars_Bendix/Publications/Papers/siegen.ps.gz (Accessed 06 Feb 2010).

Bennet, A. & Bennet, D. [2004]. Organizational survival in the New World: the intelligent complex adaptive system. Boston, MA.: Butterworth-Heinemann.

Berbers, Y. [1999]. "A component-oriented approach for building complex embedded systems". Proceedings of the 7th International conference on engineering of modern electric systems. Oradea, Romania 27-29 May 1999: 205-210.

Berger, A. [2002]. Embedded systems design: an introduction to processes, tools and techniques. New York, NY: CMP Books.

Bergeron, B. [2003]. Essentials of knowledge management. New York: NY: John Wiley & Sons Inc.

Berghel, H. [1997]. "Cyberspace 2000: dealing with information overload." Communications of the ACM, 40(2): 19-24.

Bergmann, R., Althoff, K-D., Breen, S., Göker, M., Manago, M., Traphöner, R. & Wess, S. [2003]. Developing industrial case-based reasoning applications: the INRECA methodology (2nd edition). Berlin, Germany: Springer.

Bhave, R. & Narendra, N. [2000]. "An innovative strategy for organizational learning." Proceedings of the world congress on total quality (WCTQ 2000). Mumbai, India. Available at http://resources.bnet.com/topic/total+quality+management+concept.html (Accessed 12 Mar 2008).

Binney, D. [2001]. "The knowledge management spectrum: understanding the KM landscape." Journal of Knowledge Management, 5(1): 33-42.

Birk, A. & Tautz, C. [1998]. "Knowledge management of software engineering lessons Learned." Proceedings of the 10th international conference on software engineering and knowledge engineering (SEKE'98). San Francisco, CA. 18-20 Jun 1998. http://publica.fraunhofer.de/documents/PX-51617.html (Accessed 10 Jul 2010).

Birk, A., Surmann, D. & Althoff, K. [1999]. Applications of Knowledge Acquisition in Experimental Software Engineering. Knowledge Acquisition, Modeling and Management: Proceedings of the 11th European Workshop (EKAW'99). Dagstuhl Castle, Germany May 26-29 1999: 67-84.

Blackburn, J., Scudder, G. & Van Wassenhove, L. [1996]. "Improving speed and productivity of software development: a global survey of software developers." IEEE Transactions on Software Engineering, 22(12): 875-885.

Blessing, L. [1993]. A process-based approach to computer supported engineering design. Unpublished Ph.D. Thesis, University of Twente, The Netherlands.

Bloom, B., Mesia, B. & Krathwohl, D. [1964]. Taxonomy of educational objectives. New York, NY.: David McKay.

Blosiu, J. [1999]. "Use of synectics as an idea seeding technique to enhance design creativity." IEEE International Conference on Systems, Man, and Cybernetics 1999, Toyko, Japan 12-15 Oct 1999, 3: 1001-1006.

Boehm, B. [1988]. "A spiral model of software development and enhancement." Computer, 21(5): 61-72.

Bonner, D. [2000]. "The knowledge management challenge: new roles and responsibilities for chief knowledge officers and chief learning officers". In Phillips, J. & Bonner, D. (eds.). Leading knowledge management and learning. Alexandria, VA.: American Society for Training & Development: 3-19.

Borghoff, U. & Pareschi, R. [1998]. Information technology for knowledge management.

Berlin: Springer.

Bouthillier, F. & Shearer, K. [2002]. "Understanding knowledge management and information management: the need for an empirical perspective." Information Research, 8 (1): 141.

Brazelton, J. & Gorry, G. [2003]. "On site: creating a knowledge-sharing community: if you build it, will they come?" Communications of the ACM, 46(2): 23-25.

Brennan, L. & Gupta, S.M. [1993]. "A structured analysis of material requirements planning systems under combined demand and supply uncertainty." International Journal of Production Research, 31: 1689-1707.

Briggs, J. [1992]. "Group Projects in Software Engineering at York." SIGCSE Bulletin, 23: 48-50.

Brinkkemper, S., Lyytinen, K. & Welke, R. [1996]. Method Engineering. London: Chapman & Hall.

Broomé, M. & Runeson, P. [1999]. "Technical requirements for the implementation of an experience base." Proceedings of the international conference on Software Engineering and Knowledge Engineering (SEKE'99), Kaiserslautern, Germany 17-19 Jun 1999: 119-123.

Brown, C. [1999]. Human-computer interface design guidelines. Exeter: Intellect Books.

Brown, J. & Duguid, P. [1991]. "Organizational Learning and Communities-of-Practise: Toward a Unified View of Working, Learning and Innovation." Organizational Science, 2(1): 40-57.

Browning, T. [1999]. Modeling and analyzing complex system development: cost, schedule, and performance. Unpublished PhD Thesis, Massachusetts Institute of Technology, A&A Department.

Bűchel, B. & Raub, S. [2002]. "Building knowledge-creating value networks." European Management Journal, 20 (6): 587-596.

Capra, R., Marchionini, G., Oh, J., Stutzman, F. & Zhang, Y. [2007]. "Effects of structure and interaction style on distinct search tasks." Proceedings of the 2007 conference on Digital libraries: Vancouver, Canada 17-23 Jun 2007. New York, NY.: ACM Press: 442-451.

Capshaw, S. [1999]. Whaddya know: Find out with a knowledge audit the first step towards knowledge management. Available at http://www.aiim.org/inform/july99/p16.html. (Accessed 15 Jan 2006).

Carlsen, S., Johnsen, S., Jørgensen, H., Coll, G, Mæhle, Å., Carlsen, A. & Hatling, M. [1999]. "Knowledge re-activation mediated through knowledge carriers". Proceedings of the international conference on Management of Information and Communication Technology 1999 (MICT 99). Copenhagen, Denmark: 15-16 Sep 1999. Available at http://www.kunne.no/upload/Gamle%20publikasjoner/Særtrykk/S0299%20Knowledge%20Re-Activation%20Mediated%20through%20Knowledge%20Carriers%20%20.pdf (Accessed 6 Feb 2010).

Carneiro, A. [2000]. "How does knowledge management influence innovation and competitiveness?" Journal of Knowledge Management, 4(2): 87-98.

Caspi, P., Sangiovanni-Vincentelli, A., Almeida, I., Benveniste, A., Bouyssounouse, B., Buttazzo, G., Crnkovic, I., Damm, W., Engblom, J., Folher, G., Garcia-Valls, M., Kopetz, H., Lakhnech, Y., Laroussinie, F., Lavagno, L., Lipari, G., Maraninchi, F., Peti, P. H., de la Puente, J., Scaife, N., Sifakis, J., de Simone, R., Torngren, M., Ver´issimo, P., Wellings, A. J., Wilhelm, R., Willemse, T. & Yi, W. [2005]. "Guidelines

University of Cape Town

for a Graduate Curriculum on Embedded Software and Systems." ACM Transactions on Embedded Computing Systems, 4(3): 587-611.

Castells, M. [1996]. The information age: economy, society and culture. Oxford: Blackwell.

Catsoulis, J. [2002]. Designing embedded hardware. Sebastopol, CA.: O'Reilly.

Chan, J. & Yu, K. [2004]. "TRIZ-aided technology mapping for information system implementation." Proceedings of the IEEE international conference on engineering management: innovation and entrepreneurship for sustainable development. Singapore 18-21 Oct 2004,1: 239-243.

Charette, R. [2005]. "Why software fails." IEEE Spectrum, 42(9): 36.

Cherry, S. [2004]. "Telecom: Edholm's law of bandwidth." IEEE Spectrum, 41(7): 58-60.

Chikofsky, E. & Cross, J. [1990]. "Reverse engineering and design recovery: a taxonomy." IEEE Software, 7(1): 13-17.

Child, J. [2001]. "Survey finds embedded efforts lagging, lacking." EE Times, 250(18): 967-989.

Chiodo, M., Giusto, P., Jurecska, A., Hsieh, H., Sangiovanni-Vincentelli, A., Lavagno, L. & Marelli, M. [1994]. "Hardware-software co-design of embedded systems." Micro IEEE, 14(4): 26-36.

Coates, C., Arden, B., Bartee, T., Bell, C., Kuo, F., McCluskey, E. & Surber, W. [1971]. "An undergraduate computer engineering option for electrical engineering." Proceedings of the IEEE, 59(6): 854-860.

Cogent Computers. [2005]. CSB337 Atmel AT91RM9200 based SBC. Available at http://www.cogcomp.com/csb_csb337.htm. (Accessed 14 Feb 2005).

Collins-Sussman, B. [2002]. "The subversion project: building a better CVS." Linux Journal, 94: 3.

Conradi, R. & Dingsøyr, T. [2000]. "Software experience bases: a consolidated evaluation and status report." Proceedings of the 2nd international conference on product focused software process improvement (PROFES, 2000). Oulu, Finland 20-22 Jun 2000: 391-406.

Conradi, H. & Fuggetta, A. [2002]. "Improving software process improvement." IEEE Software, 19(4): 92-99.

Coombs, R. & Hull, R. [1998]. "Knowledge management practices and path-dependency in innovation." Research Policy, 27(3): 237-253.

Cordeiro, L., Barreto, R., Barcelos, R., Oliveira, M., Lucena, V. & Maciel, P. [2007]. "Agile development methodology for embedded systems: a platform-based design approach." 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07). Tucson, AZ. 26-29 Mar 2007: 195-202.

Cortada, J. & Woods, J. [1999]. The knowledge management yearbook: 1999-2000. Woburn: MA: Butterworth-Heinemann.

Counsell, S., Phalp, K., Mendes, E. & Geddes, S. [2005]. "What formal models cannot show us: people issues people issues during the prototyping process." In Bomarius, F & Komi-Sirviö, S. (eds.) Proceedings of the 6th International: product-focused software process improvement conference (PROFES 2005). Oulu, Finland 13-15 Jun 2005: 3-15.

University of Cape Town

Counsell, J., Porter, I. Dawson, D. & Duffy, M. [1999]. "Schemebuilder: computer aided knowledge based design of mechatronic systems." Assembly Automation, 19(2): 129-38.

Covin, J. & Slevin, D. [1989]. "Strategic management of small firms in hostile and benign environments." Strategic Management Journal,10(1): 75-87.

Cross, N. [1990]. "The nature and nurture of design ability." Design Studies, 11(3): 127-140.

Cross, N. [1994]. Engineering design methods: strategies for product design (2nd edition). Chichester: John Wiley.

Cross, N. [2000]. Engineering design methods: strategies for product design (3rd edition). Chichester: John Wiley.

Cross, N. [2004]. "Expertise in design: an overview." Design Studies, 25(5): 427-441.

Cross, N., Christiaans, H. & Dorst, K. (eds.) [1996]. Analysing Design Activity. Chichester: John Wiley.

Cross, N., Naughton, J. & Walker, D. [1981]. "Design method and scientific method." Design Studies, 2(4): 195-201.

Crowston, K. & Howison, J. [2005]. "The social structure of free and open source software development." First Monday,10 (2): 2-7.

Cummings J. & Teng B. [2003]. "Transferring R&D knowledge: The Key Factors Affecting Knowledge Transfer Success." Journal of Engineering and Technology Management, 20: 39-68.

Curtis, B, Krasner, H. & Iscoe, N. [1988]. "A field study of the software design process for large systems". Communications of the ACM, 31(11): 1268-1287.

Cushman, A., Fleming, M., Rosser, B., Hunter, R. & Harris, K. [1999]. The knowledge management scenario: trends and directions for 1998-2003. Available at http://gartner6.gartnerweb.com/ggbin/ggtoc (Accessed 12 Dec 2007).

Cygwin. [2008]. Cygwin Information and Installation. Available at http://www.cygwin.com/ (Accessed 18 Nov 2008).

Dalkir, K. [2005]. Knowledge management in theory and practice. Burlington, Canada: Elsevier Butterworth-Heinemann.

Davenport, T. [2002]. "Knowledge management case study." Knowledge Management at Microsoft. Available at ftp://ftp.unibocconi.it/pub/corsi/strut738/wop/EY.htm. (Accessed 14 Feb 2007).

Davenport, T. & Klahr, P. [1998]. "Managing customer support knowledge". California Management Review, 40(3): 195-208.

Davenport, T. & Prusak, L. [1998]. Working knowledge: how organizations manage what they know. Cambridge, MA: Harvard Business School Press.

Davenport, T. & Prusak, L. [2000]. "Working knowledge: How organizations manage what they know." Ubiquity, 1(24): 2.

Davenport, T., Long, D. & Beers, M. [1998]. "Successful knowledge management projects." Massachusetts Institute of Technology Sloan Management Review, 40 (2): 43-57.

Davies, R. [1998]. "An evolutionary approach to organisational learning: an experiment by an NGO in Bangladesh". In Mosse, D., Farrington, J. & Rew, A. (eds.) Development as Process: Concepts and Methods for Working with Complexity. London: Routledge: 68-83.

University of Cape Town

Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P. & Sitaram, P. [1993]. "Identifying and measuring quality in a software requirements specification." Proceedings of the First International Software Metrics Symposium. Baltimore, MD. 21-22 May1993:141-152.

Davis, L. [2008]. SPI integrated circuits bus. Available at http://www.interfacebus.com/SPI_Bus.html (Accessed 23 Nov 2008).

De Meyer, A., Loch, C. & Pich, M. [2002]. "Managing project uncertainty: from variation to chaos." Massachusetts Institute of Technology Sloan Management Review, 44 (2): 60-67.

Desouza, K. & Evaristo, R. [2003]. "Global knowledge management strategies." European Management Journal, 21(1): 62-67.

Dhanaraj, C., Lyles, M., Steensma, H. & Tihanyi, L. [2004]. "Managing tacit and explicit knowledge transfer in IJVs: the role of relational embeddedness and the impact on performance." Journal of International Business Studies, 35(5): 428-442.

Dhont, S. [2003]. Knowledge management, innovation and creativity. TNO report 018.30201.50. Available at http://www.ukwon.net/files/kdb/bb41c78da9bb0c4f4888fd47beb6e799.pdf (Accessed 10 Feb 2010).

Dignum, V. [2006]. "An overview of agents in knowledge management." Lecture Notes in Computer Science, 4369: 175.

Dingsøyr, T. [2003]. "Knowledge management in medium-sized software consulting companies." Empirical Software Engineering, 7(4): 383-386.

Dingsøyr, T. & Conradi, R. [2002]. "A survey of case studies of the use of knowledge management in software engineering." International Journal of Software Engineering and Knowledge Engineering, 12(4): 391-414.

Dionne, S., Yammarino, F., Atwater, L. & Spangler, W [2004]. "Transformational leadership and team performance". Journal of Organizational Change Management, 17(2): 177-193.

Doolittle, L. & Nelson, J. [2006]. Boa Webserver. Available at http://www.boa.org/ (Accessed 20 May 2006).

Douglass, B. [1999]. Doing hard time: developing real-time systems with UML, objects, frameworks, and patterns. Reading, MA.: Addison Wesley Longman.

Douglass, B. [2000]. Real-Time UML (2nd edition). Reading, MA.: Addison Wesley Longman.

Dove, R. [1999]. "Knowledge management, response ability, and the agile enterprise." Journal of Knowledge Management. 3(1): 18-35.

Drucker, P. [1998]. "The coming of the new organization." In Drucker, P. F., Garvin, D., Leonard, D., Straus, S. & Brown, J. (Eds.) Harvard Business Review on Knowledge Management. Boston, MA: Harvard Business School Press: 1-20.

Drucker, P. [2000]. "Managing knowledge means managing oneself." Leader to Leader, 16: 1-4.

Drucker, P., Garvin, D., Leonard, D., Straus, S. & Brown, J. [1998]. Harvard Business Review on Knowledge Management. Boston, MA.: Harvard Business School Press.

Earl, M. [2001]. "Knowledge Management Strategies; Towards a Taxonomy". Journal of Management Information Systems, 18(1): 215-233.

Easterby-Smith, M. & Lyles, M. [2005]. The Blackwell handbook of organizational learning and knowledge management. Oxford: Blackwell.

Edvinsson, L. & Malone, M. [1997]. Intellectual Capital: Realizing Your Company's True

University of Cape Town

Value by Finding Its Hidden Brainpower. New York, NY: HarperCollins.

Edwards, S., Lavagno, L. & Lee, E. [1997]. "Design of embedded systems: formal models, validation, and synthesis." Proceedings of the IEEE, 85(3): 366-390.

Elliott, J. [1999]. "Making evidence-based practice educational." British Educational Research Journal, 27: 555-574.

Ellis C. & Wainer J. [2002]. "Groupware and computer supported cooperative work." In: Waiss G. (Ed.) Multiagent systems: a modern approach to distributed artificial intelligence. Boston, MA: Massachusetts Institute of Technology Press: 425-458.

emDebian. [2007]. The embedded Debian project. Available at http://www.emdebian.org/. (Accessed 12 Aug 2007).

Enderunix.org [2008]. Unix Daemon server programming. Available at http://www.enderunix.org/docs/eng/daemon.php (Accessed 18 Mar 2008).

English, L. [1999]. Improving data warehouse and business information quality. New York, NY: Wiley Computer Publishing.

Eppinger, S., Nukala, M. & Whitney, D. [1997]. "Generalized models of design iteration using signal flow graphs." Research in Engineering Design, 9(2): 112-123.

Eppinger, S., Whitney, D., Smith, R. & Gebala, D. [1994]. "A model-based method for organizing tasks in product development." Research in Engineering Design, 6(1): 1-13.

Eraut, M. [2000]. "Non-formal learning, implicit learning and tacit knowledge in professional work." In Coffield, F. (ed.) The necessity of informal learning. Bristol: Policy Press: 12-31.

Erdogmus, H. [2002]. Advances in software engineering: comprehension, evaluation, and evolution. Berlin, Germany: Springer.

Ernst, D. & Kim, L. [2002]. "Global production networks, knowledge diffusion, and local capability formation." Research Policy, 31(8-9): 1417-1429.

Espinosa, A., Harnden, R. & Walker, J. [2007]. "Beyond hierarchy: a complexity management perspective." Kybernetes, 36(3/4): 333-347.

Fayad, M., Schmidt, D. & Johnson, R. [1999]. Building application frameworks: object-oriented foundations of framework design. Chichester, NY.: John Wiley.

Feldmann, R. [1999]. "Developing a tailored reuse repository structure: experience and first results". Proceedings of the Workshop on Learning Software Organisations (SEKE'99). Kaiserslautern, Germany, 17-19 Jun 1999: 119-123.

Feltus, A. [1995]. "Exploding the myths of benchmarking". Continuous Journey, 19 (4). Available at http://www.apqc.org/topics/articles/bench02.htm (Accessed 1 Jul 2007).

Finger, R. [1992]. "AES3-1992: the revised two-channel digital audio interface." Journal of the Audio Engineering Society, 40(3): 107-116.

Fischer, M. [2001]. Knowledge, complexity and innovation systems. Berlin, Germany: Springer.

Fischer, G. & Schneider, M. [1984]. "Knowledge-based communication processes in software engineering". IEEE Software, 34(5): 34-49.

Floyd, C. [1984]. "A systematic look at prototyping." In Budde, R. (ed.) Approaches to prototyping. Berlin, Germany: Springer: 1-18.

University of Cape Town

Ford, D. [1995]. The dynamics of project management: an investigation of the impacts of project process and coordination on performance. Boston, MA.: Massachusetts Institute of Technology Press.

Forsberg, K., Mooz, H. & Cotterman, H. [2000]. Visualizing project management: a model for business and technical success. New York, NY.: John Wiley.

Fowler, K. [2007]. What every engineer should know about developing real-time embedded products. New York, NY.: CRC Press.

Freeman, G. & Schach, S. [2004]. "The task-dependent nature of the maintenance of object-oriented programs." The Journal of Systems and Software, 76: 195-206.

French, W. & Bell, C. [1990]. Organisational development. New York, NY.: Prentice-Hall.

Fuggetta, A. [2000]. "Software process: a roadmap." Proceedings of the 22nd international conference on software engineering: future of software engineering track (ICSE 2000), Limerick, Ireland 4-11Jun 2000: 25-34.

Fuld, L. [1994]. The new competitor intelligence: the complete resource for finding, analyzing, and using information about your competitors. New York, NY.: John Wiley.

Galliers, R. & Newell, S. [2001]. "Back to the future: from knowledge management to data management." In Smithson, S. (ed.) Proceedings of the 9th european conference on information systems. Bled, Slovenia 27-29 Jun 2001: 609-615.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. [1997]. Design Patterns. Boston, MA.: Addison-Wesley.

Ganssle, J. [1999] The art of designing embedded systems. Boston, MA.: Newnes.

Ganssle, J. [2007]. Embedded systems. Boston, MA.: Newnes.

Gao, S., Zhang, Z. & Hawryszkiewycz, I. [2005]. "Managing collaborative business processes in knowledge-intensive environments." Proceedings of the IEEE international conference on services computing (SCC'05), Orlando, FL.: 11-15 Jul 2005, 2: 107-114.

Gehrmann, B., Tennis, C. & Pol, B. [2004]. KDevelop User Manual. Available at http://docs.kde.org/kde3/en/kdevelop/kdevelop/ (Accessed 8 Mar 2004).

German, D. & Hindle, A. [2006]. "Visualizing the evolution of software using softchange." International Journal of Software Engineering, 16(1): 5-21.

Gibbons, M., Nowotny, H., Limoges, C., Trow, M., Schwartzman, S. & Scott, P. [1994]. The new production of knowledge: the dynamics of science and research in contemporary societies. London: Sage.

Glass, R. [1999]. "The realities of software technology pay-offs." Communications of the ACM, 42(2): 74-79.

GNU [2005]. GNU Binutils. Available at http://www.gnu.org/software/binutils/ (Accessed 12 Nov 2005).

GNU. [2008a]. The GNU C library. Available at http://www.gnu.org/software/libc/manual/ (Accessed 24 Apr 2008).

GNU. [2008b]. Findutils - GNU project. Available at http://www.gnu.org/software/findutils. (Accessed 12 Apr 2008).

GNU [2009]. Grep. Available at http://www.gnu.org/software/grep/ (Accessed 23 Nov 2009).

Gold, A., Malhotra, A. & Segarset, A. [2001]. "Knowledge management: an organizational capabilities perspective." Journal of Management Information Systems 18(1): 185-214.

University of Cape Town

Gopalswamy, S., Kinikar, J. & Sivashankar, S. [2004]. "Practical considerations for the implementation of model based control system development processes." Proceedings of the IEEE Conference on control applications (CCA 2004). Taipei, Taiwan 1-4 Sep 2004, 2: 1425-1430.

Goossens, G., Van Praet, J., Lanneer, D., Geurts, W., Kifli, A., Liem, C. & Paulin, P. G. [1997]. "Embedded software in real-time signal processing systems: design technologies." Proceedings of the IEEE, 85 (3): 436-454.

Graaf, B., Lormans, M. & Toetenel, H. [2003]. "Embedded software engineering: the state of the practice". IEEE Software, 20: 61-69.

Graham, A. [2008]. Statistics. New York, NY: McGraw-Hill.

Grant, R. [1996]. "Prospering in dynamically competitive environments: organizational capabilities as knowledge integration." Organizational Science, 7(4): 375-387.

Grenning, J., Peeters, J. & Behring, C. [2004]. "Agile Development for Embedded Software." Lecture Notes in Computer Science, 3134: 194-195.

Greenfield, J., Short, K. Cook, S., Kent, S. & Crupi, J. [2007]. Software factories: assembling applications with patterns, models, frameworks, and tools. New York, NY.: John Wiley.

Grimheden, M. & Törngren, M. [2005]. "What is embedded systems and how should it be taught? – results from a didactic analysis." ACM Transactions on Embedded Computing Systems (TECS), 4(3): 651.

Groff, T. & Jones, T. [2003]. Introduction to knowledge management: KM in business. Amsterdam, The Netherlands: Butterworth-Heinemann.

Gruber, T. [1993]. "A translation approach to portable ontologies." Knowledge Acquisition, 5(2): 199-220.

Grundspenkis, J. [2007]. "Agent-based approach for organization and personal knowledge modelling: knowledge management perspective." Journal of Intelligent Manufacturing, 18(4): 451-457.

Grundspenkis, J. & Kirikova, M. [2005]. "Impact of the intelligent agent paradigm on knowledge management." In Leondes, C. (ed.) Intelligent knowledge-based systems, Volume 1: Knowledge-based systems. Boston, MA: Kluwer.

Grune, D. [2007]. Concurrent versions system CVS. Available at http://www.nongnu.org/cvs (Accessed 5 Mar 2007).

Guthrie, J. & Petty, R. [1999]. "Knowledge management: the information revolution has created the need for a codified system of gathering and controlling knowledge." Company Secretary, 9(1): 38-41.

Hahn, J. & Subramani, M. [2000]. "A framework of knowledge management systems: issues and challenges for theory and practice." Proceedings of the 21st international conference on Information systems. Atlanta, GA.: 10-14 Dec 2000: 302-312.

Hall, T., Rainer, A. & Jagielska, D. [2005]. "Using software development progress data to understand threats to project outcomes". Proceedings of the 11th IEEE international symposium on software metrics (METRICS 2005). Como, Italy 19-22 Sep 2005: 10-18.

Hallows, J. [2002]. The project management office toolkit: a step-by-step guide to setting up a project management office. New York, NY.: AMACOM/American Management Association.

Hansen, M. [1999]. "The search-transfer problem: the role of weak ties in sharing knowledge across organizational subunits." Administrative Science Quarterly, 44: 82-111.

University of Cape Town

Hansen, M., Nohria, N., & Tiemey, T. [2001]. "What's Your Strategy for Managing Knowledge?" Harvard Business Review on Organizational Learning. Boston: Harvard Business School Press: 61-86.

Harvey, M. & Speier, C. [2000]. "Developing an inter-organization relational management perspective." Journal of Marketing Channels, 7(4): 23-44.

Healey, P., Garrod, S., Fay, N., Lee, J. & Oberlander, J. [2002]. "Interactional context in graphical communication." Proceedings of the 24th annual conference of the Cognitive Science Society, Fairfax VA. 7-10 Aug 2002: 441-446.

Heikkinen, M. [1997]. A concurrent engineering process for embedded systems development. Licentiate thesis, University of Oulu, Finland, 1997 (Technical Research Centre of Finland, VTT Publications 313).

Henkel, J. [2006]. Selective revealing in open innovation processes: the case of embedded linux. Working paper. Available at http://www.tim.wi.tum.de/paper/Henkel_Selective_revealing_2005.pdf (Accessed 12 Dec 2007).

Henniger, S. [1997a]. "Case-based knowledge management tools in software development." Automated Software Engineering, 4: 319-339.

Henniger, S. [1997b] "Capturing and formalizing best practices in a software development organization." Proceedings of the 9th international conference on software engineering and knowledge engineering (SEKE'97, 1997). Madrid, Spain 17-20 Jun 1997: 24-31.

Henninger, S. & Schlabach, J. [2001]. "A tool for managing software development knowledge." Proceedings of the 3rd international conference on product focused software process improvement (PROFES 01). Kaiserslautern, Germany 10-13 September 2001: 182-195.

Herbsleb, J. , Mockus, A., Finholt, T., Grinter, R. [2001]. "An empirical study of global software development: distance and speed" Proceedings of the 23rd international conference on software engineering. Toronto, Canada 19-21 May 2001: 81-90.

Herrmann, T., Kunau, G., Loser, K. & Menold, N. [2004]. "Socio-technical walkthrough: Designing technology along work processes." Proceedings of the participatory design conference. Toronto, Canada 27-31 Jul 2004, 3: 132-141.

Hill, M. [1999]. The Impact of Information on Society. London: Bowker Saur.

Ho, J. & Tang, R. [2001]. Towards an optimal resolution to information overload: an Infomediary approach. Proceedings of the international ACM SIGGROUP conference on supporting group work (GROUP '01). Boulder, CO. 30 Sep-03 Oct 2001: 91-96.

Hoffmann, M., Loser, K., Walter, T. & Herrmann, T. [1999]. "A design process for embedding knowledge management in everyday work." Proceedings of the international ACM SIGGROUP conference on supporting group work (GROUP '99). Phoenix, AZ. 14-17 Nov 1999: 296-305.

Holsapple, C. [2003]. Handbook of knowledge management. New York, NY.: Springer.

Houdek, F., Schneider, K. & Wieser, E. [1998]. "Establishing experience factories at Daimler-Benz: An experience report." Proceedings of the 20th IEEE international conference on software engineering (ICSE 20). Kyoto, Japan 19-25 Apr 1998: 443-447.

House, R., Rousseau, D. & Thomas-Hunt, M. [1995]. "The Meso paradigm: a framework for the integration of micro and macro organizational behavior." Research in organizational behavior, 17: 71-71.

Hu, C. & Chen, P. [2002]. "Design and evaluation of a knowledge management system." IEEE Software, 19: 56–59.

University of Cape Town

Hughes, B. & Cotterell, M. [2005]. Software project management (4th edition). London: McGraw-Hill.

Humphrey, W., Lovelace, M., & Hoppes, R. [1999]. Introduction to the Team Software Process. Reading, MA.: Addison-Wesley.

IBM. [2009]. Rational Rose Technical Developer. Available at http://www-01.ibm.com/software/awdtools/developer/technical/?S_TACT=105AGY59&S_CMP=13&ca=dtl-13. (Accessed 10 Jun 2009).

Jackson, D. & Caspi, P. [2005]. "Embedded systems education: Future directions, initiatives, and cooperation." ACM SIGBED Review, 2(4): 1-4.

Jacobson, I., Booch, G. & Rumbaugh, J. [1999]. The unified software development process. Reading, MA: Addison Wesley.

Jantsch, A. & Tenhunen, H. [2003]. Networks on chip. Boston, MA.: Kluwer Academic Publishers.

Jepsen, H.P, Dall, J. & Beuche, D. [2007]. "Minimally invasive migration to software product lines." Proceedings of the 11th international software product line conference (SPLC 2007). Kyoto, Japan 10-14 Sep 2007: 203-211.

Jerraya, A. [2004]. "Long term trends for embedded system design", Proceedings of the digital system design, EUROMICRO systems. Rennes, France 31 Aug-3 Sep 2004: 20-26.

Jerraya, A. & Wolf, W. [2005]. "Hardware/Software interface codesign for embedded systems." Computer, 38 (2): 63-69.

Johnson, B., Lorenz, E. & Lundvall, B. [2002]. "Why all this fuss about codified and tacit knowledge?" Industrial and Corporate Change, 11(2): 245-262.

Jørgensen, M. & Sjøberg, D. [2000]. "The importance of NOT learning from experience." Proceedings of European software process improvement conference. Copenhagen, Denmark 7-9 Nov 2000. Available at http://www.informatik.uni-bremen.de/gdpa/conferences/eurospi2000.htm (Accessed 6 Feb 2010).

Joy, B. & Kennedy, K. [1999]. Information technology research: investing in our future: report from the President's Information Technology Advisory Committee (Pitac). Available at http://ww.itrd.gov. (Accessed 14 Dec 2007).

Jurison, J. [1999]. "Software project management: the manager's view." Communications of the AIS. 2(3es): 2.

Kamsties, E. & Rombach, H. [1997]. "A framework for evaluating system and software requirements specification approaches." In Broy, M. & Rumpe, B. (eds.) Requirements targeting software and systems engineering. Berlin, Germany: Springer: 203-222.

Kan, S. [2002]. Metrics and models in software quality engineering. Boston, MA.: Addison-Wesley Longman.

Karadsheh, L., Mansour, E., Alhawari, S., Azar, G. & El-Bathy, N. [2009]. "A theoretical framework for knowledge management process: towards improving knowledge performance." Communications of the IBIMA, 7(7): 67-79.

Kass, R. & Stadnyk, I. [1992]. "Using user models to improve organizational communication". Proceedings of 3rd international workshop on user modeling (UM'92). Dagstuhl, Germany 10-13 August 1992: 135-147.

Karsai, G., Sztipanovits, J., Ledeczi, A. & Bapty, T. [2003]. "Model-integrated development of embedded software." Proceedings of the IEEE, 91(1): 145-164.

University of Cape Town

Kessels, J. [2001]. "Learning in organisations: a corporate curriculum for the knowledge economy." Futures, 33(6): 497-506.

Kegel, D. [2007]. Building and testing gcc/glibc cross toolchains. Available at http://www.kegel.com/crosstool. (Accessed 10 Sep 2007).

Kettinger, W., Teng, J. & Guha, S. [1997]. "Business process change: a study of methodologies, techniques, and tools." Management Information Systems Quarterly, 21: 55-80.

Kettunen, P [2001]. Towards rapid development of embedded telecommunications system software products. Working Paper (unpublished), Helsinki University of Technology, Finland.

Kettunen, P. [2003]. "Managing embedded software project team knowledge." IEEE Proceedings: Software, 150(6): 359-366.

Kettunen, P. & Laanti, M. [2005]. "How to steer an embedded software project: tactics for selecting the software process model." Information and software technology, 47(9): 587-608.

Keutzer, K. [2002]. "System-level design: Orthogonalization of concerns and platform-based design". IEEE Transactions on Computer Aided Design, 19 (12): 1523-1542.

Kirikova M. & Grundspenkis J. [2000]. "Using knowledge distribution in requirements engineering." In Leondes C.T. (ed.) Knowledge based systems: techniques and applications (1). San Diego, CA.: Academic Press: 149-184.

Kitamura, Y., Koji, Y. & Mizoguchi, R. [2005]. "An ontological model of device function and its deployment for engineering knowledge sharing." Proceedings of the 1st workshop – formal ontologies meet industry (FOMI). Castelvouva del Guarda, Italy 9-10 Jun 2005. Available at http://www.ei.sanken.osaka-u.ac.jp/pub/kita/documents/kita-fomi05.pdf (Accessed 6 Feb 2010).

Kitamura, Y., Washio, N., Koji, Y. & Mizoguchi, R. [2005]. "Functional metadata schema for engineering knowledge management". Proceedings of the workshop on activities on semantic web technologies in Japan, The 14th international World Wide Web Conference (WWW2005). Chiba, Japan 10 May 2005. Available at http://www.ei.sanken.osaka-u.ac.jp/pub/documents/kita-www05.pdf (Accessed 6 Feb 2010).

Kitamura, Y., Washio, N., Koji, Y. & Mizoguchi, R. [2006]. "Towards ontologies of functionality and semantic annotation for technical knowledge management." In Hirata, K., Nishida, T., Sumi, Y., Izumi, K., Yamaguchi, T. & Harao, M. (eds.) New frontiers in Artificial Intelligence. Berlin/Heidelberg, Germany: Springer.

Kitchenham, B. [1998]. "The certainty of uncertainty". Proceedings of the 4th European conference on software measurement and ICT (FESMA 98). Antwerp, Belgium 6-8 May 1998: 33-38.

Kitchenham, B., Pickard, L. & Pfleeger, S. [1995]. "Case studies for method and tool evaluation." IEEE Software, 12(4): 52-62.

Kliem, R. & Ludin, I. [1998]. Project management practitioner's handbook. Amacom Books.

Knapik M. & Johnson J. [1998]. Developing intelligent agents for distributed systems. New York: McGraw Hill.

Knoppix. [2009]. Knoppix Linux live CD. Available at http://www.knoppix.org/. (Accessed 23 Apr 2009).

Knorr-Cetina, K. [1997]. "Sociality with objects: social relations in postsocial knowledge societies." Theory, culture & society, 14(4): 1-30.

University of Cape Town

Knorr-Cetina, K. [2001]. "Objectual practice", in T.R. Schatzki, K. Knorr Cetina, & E. von Savigny (eds.) The practice turn in contemporary theory. London: Routledge: 175-188.

Knorr-Cetina, K. & Brugger, U. [2002]. "Traders' engagement with markets: a postsocial relationship." Theory Culture Society, 9(5): 161-185.

Ko, A., DeLine, R. & Venolia, G. [2007a]. Information needs in collocated software development teams. Proceedings of the 29th international conference on software engineering (ICSE 2007), Minneapolis, MN. 20-26 May 2007: 344-353.

Ko, H., Chang, G. & Kangtae, K. [2007b]. "A reengineering approach of the legacy system in the digital media domain." Procceedings of the international conference on software engineering advances (ICSEA 2007). Cap Esterel, French Riviera, France 25-31 August 2007: 76.

Kogut, B. & Zander, U. [1992]. "Knowledge of the firm, combinative capabilities, and the replication of technology." Organization Science, 3 (3): 383-397.

Komi-Sirviö, S., Mäntyniemi, A. & Seppänen, V. [2002]. "Toward a Practical Solution for Capturing Knowledge for Software Projects." IEEE Software, 19 (3): 60-62.

Kommeren, R. & Parviainen, P. [2007]. "Philips experiences in global distributed software development." Empirical Software Engineering, 12(6): 647-660.

Koopman, P. [1996]. "Embedded system design issues (the rest of the story)". Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'96) Austin, TX 7-9 Oct 1996: 310–317.

Kopetz, H. [1997]. Real-time systems: design principles for distributed embedded applications. London: Springer Publishers.

Krishna, S. [1992]. Introduction to database and knowledge-base systems. Singapore: World Scientific Publishing.

Kroll, P. & Kruchten, P. [2003]. The rational unified process made easy. Boston, MA.: Addison-Wesley.

Kurzweil, R. [2001]. The law of accelerating returns. Available at http://www.kurzweilai.net/articles. (Accessed 01 Dec 2007).

Labrosse, J., Ganssle, J., Noergaard, T., Oshana, R., Walls, C., Curtis, H., Andrews, J., Katz, F., Gentile, R., Hyder, K. & Perrin, B. [2008]. Embedded Software. Amsterdam, The Netherlands: Newnes.

Langer, H., Gehrke, J., Hammer, J., Lorenz, M., Ingo, J. & Herzog, O. [2006]. "A framework for distributed knowledge management in autonomous logistic process". International Journal of Knowledge-based and Intelligent Engineering Systems, 10 (4): 277-290.

Larsson, S., Wall, A. & Wallin, P. [2007]. "Assessing the influence on processes when evolving the software architecture – foundations of software engineering." Proceedings of the 9th international workshop on principles of software evolution (in conjunction with the 6th ESEC/FSE joint meeting). Dubrovnik, Croatia 3-7 Sep 2007: 59-66.

Leonard, D. [1999]. Innovation and knowledge management. Williamsburg, VA.: Institute for Knowledge Management Press.

Leonard-Barton, B. [1995]. Wellsprings of knowledge: building and sustaining the source of innovation. Boston. MA.: Harvard Business School Press.

Lewis, J. [2006]. The project manager's desk reference. London: McGraw-Hill.

University of Cape Town

Lindeman, D. & Moore, D. [1994]. "PDM: An enabling technology for integrated product development." Proceedings of the IEEE annual symposium on reliability and maintainability (RAMS). Anaheim, CA. 24-27 Jan 1994: 320-326.

Lindvall, M., Rus, I., Jammalamadaka, R. & Thakker, R. [2001]. Software tools for knowledge management. Fraunhofer Center for Experimental Software Engineering. Maryland, USA. Available at http://www.dacs.dtic.mil/ techs/kmse/swtools4km.pdf. (Accessed 16 Feb 2009).

Lindvall, M., Komi-Sirviö, S., Costa, P. & Seaman, C. [2003]. Embedded Software Maintenance. Available at https://www.thedacs.com/get_pdf/DACS-347003.pdf. (Accessed 10 Nov 2008).

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M. Kiefer, D. & May, J. [2004]. "Agile software development in large organizations". Computer, 37(12): 26-34.

Linn, J. [2001]. "Embedded Software Development Challenges in the Digital Signal Processing Era". 5th International symposium on autonomous decentralized systems. Dallas, TX. 26-28 March 2001: 299-302.

Liu, J. [2000]. Real-time systems. London: Prentice Hall.

Louridas, P. [2006]. "Using wikis in software development." IEEE Software, March/April 2006: 88-91.

Luqi, L., Berzins, Y. & Qiao, Y. [2004]. "Documentation driven development for complex real-time systems". IEEE Transactions on Software Engineering, 30(12): 936-952.

Lutz, M. [2006]. Programming python. Sebastopol, CA.: O'Reilly Media Inc.

Lyles, M., Steensma, H. & Tihanyi, L. [2004]. "Managing tacit and explicit knowledge transfer in IJVs: the role of relational embeddedness and the impact on performance." Journal of International Business Studies, 35(5): 428-442.

Lynn, G., Reilly, R. & Akgűn, A. [2000]. "Knowledge management in new product teams: practices and outcomes". IEEE Transactions on Engineering Management, 47(2): 221-231.

Lyytinen, K. & Robey, D. [1999]. "Learning failure in information systems development." Information Systems Journal, 9(2): 85-101.

MacKenzie, D., Eggert, P. & Stallman, R. [2003]. Comparing and Merging Files with GNU diff and patch. Network Theory Ltd. Available at http://ufpr.dl.sourceforge.net/project/souptonuts/GNU%20Documentation/Original%20Files/GNU_diff.pdf (Accessed 12 Apr 2008).

Madhavan, R. & Grover, R. [1998]. "From embedded knowledge to embodied knowledge: new product development as knowledge management." The Journal of Marketing, 64: 1-12.

Maier, R. [2004]. Knowledge management systems: information and communication technologies for knowledge management. New York: Springer.

Maier, R. & Hadrich, T. [2006]. "Knowledge management systems". Encyclopedia of knowledge management. Schwartz, D. & Edwards, J. Idea Group: 442-450.

Maier, R. & Remus, U. [2003]. "Implementing process-oriented knowledge management strategies." Journal of Knowledge Management, 7(4): 62-74.

Majchrzak, A., Rice, R., Malhotra, A., King, N. & Ba, S. [2000]. "Technology adaptation: the case of a computer supported inter-organizational virtual team". MIS Quarterly, 24 (4): 569-600.

University of Cape Town

Majchrzak, A., Cooper, L. & Neece, O. [2003]. "Knowledge reuse in the radical innovation process at the jet propulsion laboratory." In Leibowitz, J. & Holm, J. (eds.), Knowledge management technologies and applications in NASA. Washington, DC: Government Printing Office.

Mäkäräinen, M. [2000]. Software change management processes in the development of embedded software. Unpublished Dissertation, University of Oulu, Finland, 2000 (Technical Research Centre of Finland, VTT Publications 416).

Malhotra, Y. [2007]. KMBook.com. BRINT Institute. Available at http://www.kmbook.com/. (Accessed 02 Mar 2007).

Mano, M. & Ciletti, M. [2006]. Digital Design (4th edition). London: Prentice Hall.

Mäntyniemi, A., Pikkarainen, M. & Taulavuori, A. [2004]. A Framework for Off-The-Shelf Software Component Development and Maintenance Processes. Espoo, Finland: VTT Publications. Available at http://www.vtt.fi/inf/pdf/publications/2004/P525.pdf (Accessed 06 Feb 2010).

Martensson, M. [2000]. "A critical review of knowledge management as a management tool." Journal of Knowledge Management, 4(3): 204-216.

Martland, D., Holloway, S. & Bhabuta, L. [1986]. Fourth generation languages and applications generators. Brookfield, WI: The Technical Press.

Marwedel, P. [2003]. Embedded System Design. Berlin, Germany: Springer.

McDermott, R. [1999a]. "Nurturing three-dimensional communities of practice." Knowledge Management Review, 2 (5): 26-29.

McDermott, R. [1999b]. "Why Information Technology Inspired but Cannot Deliver Knowledge Management." California Management Review, 41 (4): 103-17.

McKeown, M. [2008]. The truth about innovation. London: Pearson Education.

Medvidovic, N. & Mikic-Rakic, M. [2003]. Architectural support for programming-in-the-many. Proceedings of the ACM/IFIP/USENIX 2003 international conference on middleware. Rio de Janeiro, Brazil 16-20 June 2003: 162-181.

Mellis, W. [1998]. "Software quality management in turbulent times: are there alternatives to process oriented software quality management?" Software Quality Journal, 7(3): 277-295.

Meyers, L., Guarino, A. & Gamst, G. [2005]. Applied multivariate research: design and interpretation. Thousand Oaks, CA.: Sage.

Microchip. [2008]. MPLAB Integrated Development Environment. Available at http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002. (Accessed 20 Nov 2008).

Mikkonen, T. & Pruuden, P. [2001]. "Flexibility as a design driver." Computer, 34 (11): 52-56.

Milton, N. [2005]. Knowledge management for teams and projects. Oxford: Chandos.

Molnar, W & Nandhakumar, J. [2007]. "Managing a new computer device development in a creative ISO 9001 certified company: a case study." Proceedings of the 40th annual Hawaii international conference on system sciences (HICSS'07) Waikoloa, Big Island, HI. 3-6 January 2007. Available at http://www.informatik.uni-trier.de/~ley/db/conf/hicss/hicss2007.html (Accessed 06 Feb 2010).

Muller, H., Jahnke, J., Smith, D., Storey, M., Tilley, S. & Wong, K. [2000]. "Reverse engineering: a roadmap." Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000): Future of Software Engineering Track. Limerick, Ireland 4-11 Jun 2000: 47-60.

University of Cape Town

MySQL. [2009]. MySQL. Available at http://www.mysql.com/ (Accessed 25 Aug 2009).

Nakagawa, T., Kosha, H. & Mihara, Y. [2002]. "Reorganizing TRIZ solution generation methods into simple five in USIT." ETRIA World Conference, Strasbourg, France 6-8 Nov: 333-345.

Nelson, V., Nagle, H. Carroll, B. & Irwin, J. [1995]. Digital logic circuit analysis and design. London: Prentice Hall.

Newell, S., Swan, J. & Robertson, M. [1998]. "A cross-national comparison of the adoption of business process reengineering: Fashion-setting networks". Journal of Strategic Information Systems, 7(4): 299-317.

Nonaka, I. [1994]. "A dynamic theory of organizational knowledge creation." Organization Science, 5: 14-37.

Nonaka, I. & Konno, N. [1999]. "The Concept of "Ba": Building a Foundation for Knowledge Creation." California Management Review, 40(3): 40-54.

Nonaka, I., Konno, N. & Toyama, R. [2001a]. "Emergence of BA." In Nonaka I. & Nishiguchi, T. (eds.) Knowledge Emergence: Social, Technical, and Evolutionary Dimensions of Knowledge Creation. New York, NY.: Oxford University Press: 13-29.

Nonaka, I. & Takeuchi, H. [1995]. The knowledge-creating company: How Japanese companies create the dynamics of innovation. New York, NY.: Oxford University Press.

Nonaka, I., Toyama, R. & Boysiere, P. [2001b] "A theory of organizational knowledge creation: understanding the dynamic process of creating knowledge". In Dierkes, M, Berthoin, A, Antal, A, Child, J. & Nonaka, I. (eds.) [2001]. Handbook of organizational learning and knowledge. Oxford: Oxford University Press.

Nonaka, I., Toyama, R. and Konno, N. [2000]. "SECI, *Ba*, and leadership: A unified model of dynamic knowledge creation". Long Range Planning, 33: 5-34.

NXP Semiconductors. [2007]. The I2C-bus specification. Available at http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf (Accessed 12 Feb 2007).

Nyerges, T., Jankowski, P. & Drew, C. [2002]. "Data-gathering strategies for social-behavioural research about participatory geographical information system use." International Journal of Geographical Information Science,16(1): 1-22.

O'Brien, J. [2006]. Introduction to Information Systems (13th edition). London: McGraw-Hill.

O'Dell, C. & Grayson, C. [1999]. "If only we knew what we know: identification and transfer of internal best practices." California Management Review, 40(3):154-174.

O'Dell, C., Elliott, S. & Hubert, C. [2003]. "Achieving knowledge management outcomes." In Holsapple, C. (ed.) Handbook on knowledge management: knowledge directions 2. Berlin, Germany: Springer: 253-288.

Object Management Group. [2002]. "Object constraint language (OCL)." Specification, 1(4) Available at http://www.omg.org/cgi-bin/doc?ad/03-01-07.pdf (Accessed 15 Feb 2010).

Odell, L., Goswami, D. & Herrington, A. [1983]. In Mosenthal, P. & Walmsley, L. (eds.) Research on writing: principles and methods. New York, NY.: Longman: 221-36.

Oluic-Vukovic, V. [2001]. "From information to knowledge: some reflections on the origin of the current shifting towards knowledge processing and further perspective." Journal of the American Society for Information Science and Technology, 52: 54-61.

OMG. [2005]. UML 2.0. Available at http://www.omg.org/spec/UML/2.0/ (Accessed 12 Nov

University of Cape Town

2005).

Ordanini, A. & Pol, A. [2001]. "Infomediation and competitive advantage in B2B digital marketplaces." European Management Journal, 19(3): 276-285.

Ortenblad, A. [2007]. "The evolution of popular management ideas: an exploration and extension of the old wine in new bottles metaphor." International Journal of Management Concepts and Philosophy 2(4): 365-388.

Osiov, I., Smirnov, I., Tikhomirov, I., Vybornova, O. & Zavjlrva, O. [2006]. "Linguistic knowledge for search relevance improvement." In Tyugu, E. & Yamaguchi, E. (Eds.) Proceedings of the 7th joint conference on knowledge-based software engineering. Amsterdam, The Netherlands: IOS Press: 294-302.

Pan, S. Scarbrough, H. [1999]. "Knowledge management in practice: an exploratory case study." Technology analysis and strategic management 11(3): 359-374.

Patil, L., Dutta, D. & Sriram, R. [2005]. "Ontology-based exchange of product data semantics." IEEE Transactions on Automation Science and Engineering 2(3): 13-225.

Patrick, A. [2008]. "Formalising the 'No Information without Data-representation' Principle." Frontiers in Artificial Intelligence and Application: proceedings of the 2008 conference on current issues in computing and philosophy. Amsterdam, The Netherlands: IOS Press: 79-90.

Patriotta, G. [2004]. "On studying organizational knowledge." Knowledge Management Research & Practice 2: 3-12.

Patterson, D. & Hennessy, J. [2005]. Computer organization and design: the hardware/software interface. San Francisco: CA.: Morgan Kaufmann.

Pena-Mora, F., Sriram, R. & Logcher, R. [1993]. "SHARED DRIMS: SHARED design recommendation and intent management system". Enabling technologies: infrastructure for collaborative enterprises. IEEE Press: 213-221.

Pena-Mora, F., Sriram R. & Logcher, R. [1995]. "Conflict mitigation system for collaborative engineering". AI EDAM: special issue on concurrent engineering 9 (2): 101-123.

Pérez Pérez, M. & Sanchez, A. [2003]. "The development of university spin-offs: early dynamics of technology transfer and networking." Technovation, 23(10): 823-831.

Piaseki, D. [2005]. RFID Update: The basics, the Wal-Mart mandate, EPC primary concerns, and more. Available at http:// www.rfidjournal.com/article/articleview/781/1/82. (Accessed 14 Dec 2007).

Polanyi, M. [1958]. Personal knowledge: towards a post-critical philosophy. London, Routledge.

Project Management Institute [2004]. A guide to the project management body of knowledge (PMBOK Guide) (3rd edition). Newtown Square, PA.: Project Management Institute.

Prowell, S., Trammell, C. Linger, R. & Poore, J. [1999]. Cleanroom software engineering: technology and process. Lebanon, IN.: Addison-Wesley Publishers.

Radding, A. [1998]. Knowledge management: succeeding in the information-based global economy (1st edition). Charleston, SC.: Computer Technology Research.

Rahe, M. & Morales, C. [2005]. "Reducing resistance to change through knowledge management: a conceptual approach." Research and Practice in Human Resource Management, 13(2): 49-64.

Rennolls, K. & Al-Shawabkeh, A. [2008]. "Formal structures for data mining, knowledge discovery and communication in a knowledge management environment." Intelligent

University of Cape Town

Data Analysis 12(2): 147-163.

Richter, H. & Abowd, G. [2004]. "Tagging knowledge acquisition sessions to facilitate knowledge traceability". International Journal of Software Engineering and Knowledge Engineering,14: 3-19.

Riesgo, T., Torroja, Y. & de la Torre, E. [1999]. "Design methodologies based on hardware description languages." IEEE Transactions on Industrial Electronics, 46(1): 3-12.

Rodgers, P., Huxor, A. & Caldwell, N. [1999]. "Design support using distributed web-based AI tools." Research in Engineering Design, 11: 31-44.

Romano, N., Chen, F. & Nunamaker, J. [2002]. "Collaborative project management software". Proceedings of the 35th Hawaii international conference on system sciences (HICSS-35.02) (1). Maui, Hawaii, 7-10 January 2002: 233-242.

Rousseau, D. & House, R. [1994]. "Meso organizational behavior: avoiding three fundamental biases". In Cooper, C. & Rousseau, D. (eds.) Trends in organizational behavior (Volume 1). New York, NY.: John Wiley & Sons: 13-30.

Rowen, C. & Leibson, S. [2004]. Engineering the complex SoC: fast, flexible design with configurable processors. Upper Saddle River, NJ.: Prentice Hall.

Royce, W. [1970]. "Managing the development of large software systems." (Reprinted from Proceedings of IEEE WECON, August 1970). Available at http://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf (Accessed 10 Jul 2010).

Rumbaugh, J., Jacobson, I. & Booch, G. [2005]. The unified modeling language reference manual. (2nd Edition). Boston. MA.: Addison-Wesley.

Rumizen, M. [2002]. The complete idiot's guide to knowledge management. Madison, WI.: CWL Publishing Enterprises.

Runco, M. & Pritzker, S. [1999]. Encyclopedia of creativity (Volumes 1 & 2). San Diego, CA.: Academic Press.

Rus, I. & Lindvall, M. [2002]. "Knowledge management in software engineering." IEEE Software, 19(3): 26–38.

Rus, I., Lindvall, M. & Sinha, S. [2001]. Knowledge management in software engineering. The Data and Analysis Centre for Software (DACS) state-of-the-art report. Available at https://www.thedacs.com/techs/DACS345468.pdf Accessed 11 Jul 2010.

Sabat, H. [2007]. "Successful paradigms, execution issues and best practices to build learning organisations." International Journal of Knowledge Management Studies, 1(3): 306-329.

Sangiovanni-Vincentelli, A. & Pinto, A. [2005]. "An overview of embedded system design education at Berkeley." ACM Transactions on Embedded Computing Systems (TECS), 4(3): 472-499.

Schach, S. [2005]. Object-oriented and classical software engineering. Boston, MA.: McGraw-Hill.

Scheifler, R. & Gettys, J. [1986]. "The X window system." ACM Transactions on Graphics, 5(2): 79-109.

Sefton-Green, J. & Sinker, R. [2000]. Evaluating creativity: making and learning by young people. Milton Park: Routledge Falmer.

Senge, P. [1990]. The fifth discipline: the art and science of the learning organization. New York, NY.: Currency Doubleday.

University of Cape Town

Seth, D. [2007]. A platform based approach for embedded systems software development. Unpublished doctoral thesis, Massachusetts Institute of Technology. Available at http://dspace.mit.edu/handle/1721.1/35092 (Accessed 20 Nov 2007).

Seviora, R. [2005]. "A curriculum for embedded system engineering." ACM Transactions on Embedded Computing Systems (TECS), 4(3): 569-586.

Sgroi, M., Sheets, M., Mihal, A., Keutzer, K. & Malik, S. [2001]. "Addressing the system-on-a-chip interconnect woes through communication-based design." Proceedings of the 38th conference on design automation. Las Vegas, NV.18-22 June 2001: 667-672.

Shetler, J. [1996]. "Teaming in the microprocessor laboratory." Proceedings of the Frontiers in Education conference (Vol 3). Salt Lake City, UT. 6-9 Nov, 1996: 1437-1440.

Silvestro, R., Fitzgerald, L., Johnston, R. & Voss, C. [1992]. "Towards a classification of service processes." International Journal of Service Industry Management, 3(3): 62-75.

Simard, A., Broome, J., Drury, M., Haddon, B., O'Neil, B. & Pasho, D. [2007]. Understanding knowledge services at natural resources Canada. Ottawa, Canada: Office of the Chief Scientist, Natural Resources Canada. Available at http://warehouse.pfc.forestry.ca/HQ/27338.pdf (Accessed 11 Jul 2010).

SimIt-ARM. [2008]. SimIt-ARM. Available at http://simit-arm.sourceforge.net (Accessed 14 Feb 2008).

Simonin, B. [1999]. "Ambiguity and the process of knowledge transfer in strategic alliances." Strategic Management Journal, 20(7): 595-623.

Skuce, D. [1995]. "Knowledge management in software design: a tool and a trial." Software Engineering Journal, 10(5): 183-193.

Smith, H. & McKeen, J. [2003]. "Creating and facilitating communities of practice." In Holsapple, C.W. (ed.) Handbook on Knowledge Management 1: Knowledge Matters. Berlin/Heidelberg/New York: Springer: 393-407.

Smith, P. & Reinertsen, D. [1998]. Developing products in half the time (2nd edition). New York, NY.: John Wiley and Sons.

SnapGear. [2007]. SnapGear embedded Linux distribution home page. Available at http://www.snapgear.org/ (Accessed 21 Aug 2007).

Snowden, D. [1998]. "A framework for creating a sustainable knowledge management programme." The knowledge management yearbook 1999-2000. Oxford: Butterworth Heinemann: 52-64.

Sommerville, I. [2005]. "Integrated requirements engineering: a tutorial." IEEE Software, 22(1): 16-23.

Sommerville, I. [2006]. Software engineering. Reading, MA.: Addison-Wesley.

Sophia, A. [2006]. Creative process vs product. Available at http://ezinearticles.com/?Creative-Process-Vs-Product&id=2367713 (Accessed 15 Mar 2006).

Souchkov, V. [2005]. "Skills and tools to support productivity in creative work." Paper presented at the European summit for the future. Amsterdam, The Netherlands 27 Jan 2005. Available at http://www.xtriz.com/publications/SummitForFuture2005Souchkov.pdf (Accessed 11 Jul 2010).

Speer, S. & Hutchby, I. [2003]. "From ethics to analytics: aspects of participants' orientations to the presence and relevance of recording devices." Sociology, 37(2): 315-337.

University of Cape Town

Spinuzzi, C. [2002]. "Modeling genre ecologies." Proceedings of the ACM 20th annual international conference on computer documentation (SIGDOC).Toronto, Ontario, Canada 20-23 Oct 2002: 200-207

Spinuzzi, C. [2003]. "Compound mediation in software development: using genre ecologies to study textual artifacts." In Bazerman, C. & Russell, D. (eds.) Writing selves/writing societies. Mahwah, NJ.: Laurence Erlbaum Associates: 97-124.

Staab, S., Studer, R., Schnurr, H. & Sure, Y. [2001]. "Knowledge processes and ontologies." IEEE Intelligent systems 16(1): 26-34.

Ståhle, P. & Grönroos, M. [2000]. Dynamic intellectual capital: knowledge management in theory and practice. Porvoo/Helsinki/Juva, Finland: Werner Söderström Osakeyhtiö (WSOY) Publishers.

Stamelos, I., Angelis, L., Oikonomou, A. & Bleris, G. [2002]. "Code quality analysis in open source software development." Information Systems Journal, 12(1): 43–60.

Sternberg, R. & Frensch, P. [1991]. Complex problem solving: principles and mechanisms. Hillsdale, NJ.: Lawrence Erlbaum.

Sutter, E. [2002]. Embedded systems firmware demystified. Lawrence, KA.: CMP Books.

Stojanovic, L., Maedche, A., Motik, B. & Stojanovic, N. [2002]. "User-driven ontology evolution management." Proceedings of the 13th international conference on knowledge engineering and knowledge management: ontologies and the semantic web (EKAW02). Sigüenza, Spain 1-4 October 2002: 285-300.

Sveiby, K. [1997]. The invisible balance sheet: key indicators for accounting, control and evaluation of know-how companies. Stockholm, Sweden: The Konrad Group. Available at http://www.sveiby.com/articles/IntangAss/denosynl.htm (Accessed 10 Jul 2010).

Sveiby, K. [1998]. Measuring intangibles and intellectual capital: an emerging first standard. Available at http://www.sveiby.com/Portals/0/articles/ EmergingStandard.html (Accessed 02 Mar 2007).

Sveiby, K. [2000]. What is knowledge management? Available at http://www.sveiby.com.au/KnowledgeManagement.html (Accessed 20 Nov 2007).

Sveiby, K. [2001]. "A knowledge-based theory of the firm to guide in strategy formulation." Journal of Intellectual Capital, 2: 344-358.

Tatham, S. [2009]. PuTTY: A free Telnet/SSH client. Available at http://www.chiark.greenend.org.uk/~sgtatham/putty/ (Accessed 23 Aug 2009).

Tautz, C. & Althoff, K. [1997]. "Using case-based reasoning for reusing software knowledge." Case-based reasoning research and development: proceedings of the 2nd international conference on case-based reasoning (ICCBR-97). Providence, RI. 25-27 Jul 1997:156-165.

Tautz, C., Althoff, K. & Nick, M. [2000]. "A case-based reasoning approach for managing qualitative experience". AAAI-00 workshop on intelligent lessons learned systems. American Association for Artificial Intelligence. Papers from the AAAI Workshop. Menlo Park, Edison NJ.: AAAI Press: 54-58.

Tcl Developer Site. [2007]. Available at http://www.tcl.tk/ (Accessed 21 Mar 2007).

Teece, D. [2001]. "Strategies for managing knowledge assets: the role of firm structure and industrial context". In Nonaka, I. & Teece, D.J. (eds.), Managing industrial knowledge-creation, transfer and utilization. London: Sage Publications: 125-144.

Thamhain, H. [1990]. "Managing technologically innovative team efforts toward new product success." An international publication of the product development & management

University of Cape Town

association 7(1): 5–18.

Tian, J., Nakamori, Y. & Wierzbicki, A. [2009]. "Knowledge management and knowledge creation in academia: a study based on surveys in a Japanese research university." Journal of Knowledge Management, 13(2): 76-92.

Tierney, P. & Farmer, S. [2002]. "Creative self-efficacy: Its potential antecedents and relationship to creative performance." Academy of Management Journal, 38: 1137-1148.

Törngren, M., Grimheden, M. & Adamsson, N. [2007]. "Experiences from large embedded systems development projects in education, involving industry and research." ACM SIGBED Review, 4(1): 55-63.

Tsui, E. [2002]. "Technologies for personal and peer-to-peer (P2P) knowledge management." CSC Leading Edge Forum (LEF) Technology Grant report, 2002. Available at http://www2.csc.com/lef/programs/completed_02.html (Accessed 11 Feb 2010).

Turner, J. [2009]. The handbook of project-based management (3rd edition). New York, NY.: McGraw-Hill Professional.

Turner, R., Huemann, M. & Keegan, A. [2008]. "Human resource management in the project-oriented organization: employee well-being and ethical treatment." International Journal of Project Management, 26(5): 577-585.

Ulrich, K. & Eppinger, S. [1995]. Product design and development. New York, NY.: McGraw-Hill.

Underhill, L. & Bradfield, D. [1998]. Introstat. Cape Town, South Africa: Juta Academic.

Van der Spek, R. & Spijkervet, A. [1997]. "Knowledge management: dealing intelligently with knowledge." In Wilcox, L.C. (ed.), Knowledge management and its integrative elements. Boca Raton, FL.: CRC Press: 31-60.

Van Zolingen, S., Streumer, J. & Stooker, M. [2001]. "Problems in knowledge management: A case study of a knowledge-intensive company." International Journal of Training and Development, 5(3): 168-184.

Vaswani, V. [2004]. MySQL: the complete reference. Emeryville, CA.: McGraw-Hill Osborne.

Vinter, O. [2005]. "A framework for classification of change approaches of process improvement models." In Bomarius, F. & Komi-Sirviö, S. (eds.) Proceedings of the 6th international product-focused software process improvement conference. Oulu, Finland, 13-15 Jun 2005: 29-38.

Von Hagen, W. [2007]. Ubuntu Linux Bible. Indianapolis, IN.: Wiley.

Von Krogh, G., Spaeth, S. & Lakhani, K. [2003]. "Community, joining, and specialization in open source software innovation: a case study." Research Policy, 32(7): 1217-1241.

Wagner, I., Tellioglu, H., Balka, E., Simone, C., Ciolfi, L. [2009]. "On the effects of refactoring in the coordination of software development activities". Proceedings of the 11th European Conference on Computer Supported Cooperative Work (ECSCW 2009). Vienna, Austria 9-11 Sep 2009: 215-222.

Wake, W. [2002]. Extreme programming explored. New York, NY.: Addison-Wesley.

Wandeler, E., Thiele, L., Verhoef, M. & Lieverse, P. [2006]. "System architecture evaluation using modular performance analysis: a case study." International Journal on Software tools for technology transfer, 8(6): 649-667.

University of Cape Town

Wangenheim, C., Weber, S., Hauck, J. & Trentin, G. [2006]. "Experiences on establishing software processes in small companies." Information and Software Technology, 48(9): 890-900.

Warden, R. [1992]. "Re-engineering – a practical methodology with commercial applications". In Hall, P.A.V. (ed.) Software reuse and reverse engineering in practice. London: Chapman & Hall: 283-305.

Weinberg, G. [1975]. An Introduction to general systems thinking. New York: NY.: Wiley-Interscience.

Wenger, E. [1998]. Communities of practice: learning, meaning and identity. Cambridge: Cambridge University Press.

Wenger, E., McDermott, R. & Snyder, W. [2002]. Cultivating communities of practice. Boston, MA.: Harvard Business School Press.

White, S. [2005]. "Improving the system/software engineering interface for complex system development." Proceedings of the12th IEEE International Conference and Workshop on the Engineering of computer-based systems (ECBS '05) Greenbelt, ML. 4-7 Apr 2005: 281-288.

Wiig, K. [1995]. Knowledge management methods. New York, NY.: Schema Press.

Wiig, K. [1997]. "Knowledge management: where did it come from and where will it go?" Expert Systems with Applications, 13: 1-14.

Wikipedia. [2008]. Correlation. Available at http://en.wikipedia.org/w/index.php?title=Correlation&oldid=324601995. (Accessed 10 Mar 2008).

Wikipedia. [2009]. Virtual machine. Available at http://en.wikipedia.org/w/index.php?title=Virtual_machine&oldid=320601238. (Accessed 21 Aug 2009).

Wiktionary. [2008]. Tool chain. Available at http://en.wiktionary.org/wiki/toolchain. (Accessed 14 Oct 2008).

Wilson, T.D. [2002] "The nonsense of 'knowledge management" Information Research, 8(1), paper no. 144. Available at http://InformationR.net/ir/8-1/paper144.html (Accessed 30 Jan 2010).

Winberg, S. [2005a]. "A study of learning tasks used in implementation procedures during embedded software development." Radar Remote Sensing Group, University of Cape Town. Report no. 4527, July 2005, Available at https://rrsg.ee.uct.ac.za/refs/refview.php?id=4527 (Accessed 14 Mar 2007).

Winberg, S. [2005b]. "Getting Started With GCC and ESAOA on Linux." Unpublished laboratory tutorial. University of Cape Town.

Winberg, S. [2005c]. "CSB337 & MicroMonitor." Unpublished laboratory tutorial. University of Cape Town.

Winberg, S. [2006a]. "Embedded system artefact organization and adaptation (ESAOA) framework setup procedure." Radar Remote Sensing Group. Report no. 4648, Feburary 2006, Available at https://rrsg.ee.uct.ac.za/refs/refview.php?id=4648. (Accessed 15 Mar 2007).

Winberg, S. [2006b]. "Development of a knowledge management methodology with application to embedded system development." Radar Remote Sensing Group. Report no. 4658, March 2006, Available at https://rrsg.ee.uct.ac.za/refs/refview.php?id=4658. (Accessed 15 Mar 2007).

Winberg, S. [2006c]. "An innovative approach to embedded system education and

University of Cape Town

development." Radar Remote Sensing Group. Report no. 4649, November 2006, Available at https://rrsg.ee.uct.ac.za/refs/refview.php?id=4649. (Accessed 20 Apr 2007).

Winberg, S. [2006d]. "Knowledge management needs in embedded software development." Radar Remote Sensing Group. Report no. 4650, December 2006, Available at https://rrsg.ee.uct.ac.za/refs/ refview.php?id=4650. (Accessed 20 Apr 2007).

Winberg, S. [2006e]. "A pilot study of knowledge acquisition of embedded system methods: self-directed learning of engineering methods to improve laboratory practice." Radar Remote Sensing Group. Report no. 4657, December 2006, Available at https://rrsg.ee.uct.ac.za/refs/refview.php?id=4657. (Accessed 03 May 2007).

Winberg, S. & Schach, S. [2007]. "A pilot study of productive versus nonproductive knowledge acquisition in embedded software development." International Journal of Software Engineering and Knowledge Engineering, 11(4): 539-556.

Winberg, S., Schach, S. & Inggs, M. [2008]. "Bringing knowledge management into an engineering curriculum." South African Journal of Higher Education, 21(7): 969-983.

Winberg, S. [2009]. "A pilot study of the ESAOA Ontology Manager." Radar Remote Sensing Group. Report no. 6820, March 2009, Available at https://rrsg.ee.uct.ac.za/aigaion/publications/show/6820 (Accessed 17 Mar 2009).

Wood, W. & Agogino, A. [1996]. "Case based conceptual design information server for concurrent engineering". Computer-Aided Design, 8(5): 361-369.

Yahoo! Messenger. [2004]. Available at http://messenger.yahoo.com/. (Accessed 12 Aug 2004).

Yoon, S., Henschen, L., Park, E. K. & Makki, S. [1999]. "Using domain knowledge in knowledge discovery". Proceedings of the eighth international conference on Information and knowledge management (CIKM '99). Kansas City, MO. 2-6 Nov 1999: 243-250.

Zander, U. & Kogut, B. [1995]. "Knowledge and the speed of the transfer and imitation of organizational capabilities: an empirical test." Organization Science, 6 (1) 76-92.

Zha, X. & Du, H. [2005]. "Knowledge-intensive collaborative design modeling and support, Part 1: review, distributed models and frameworks". Computers in Industry, 57(1): 39-55.

Zha, X. & Du, H. [2006]. "Knowledge-intensive collaborative design modeling and support, Part 2: system implementation and application". Computers in Industry, 57(1): 56-71.

Zha, X., Fu, M., Lu, W., Ma, S. & Zhu, C. [2002]. Knowledge modeling in design process-knowledge modeling in product family design for mass customization, SIMTech Technical Report, MIT/02/031/PDD, Singapore, 2002.

Zhou, J., Cooper, K., Ma, H. & Yen, I. [2007]. "On the customization of components: a rule-based approach". IEEE Transactions on Knowledge and Data Engineering, 19(9): 1262-1275.

Zingheim, P. & Schuster, J. [1995]. "Supporting teams with multi-rater performance reviews." Compensation and Benefits Management, 11: 41-41.

Van Zolingen, S., Streumer, J. & Stooker, M. [2001]. "Problems in knowledge management: a case study of a knowledge-intensive company." International Journal of Training and Development, 5(3): 168-184.

University of Cape Town

Appendix A: Experiment 1 appendices

A.1 Knowledge register for first case study (P1-1)

The table below presents the list of ESAOA events for the first case study (P1-1). The thirteen columns of this table are described below:

1. *Event No.:* event number;
2. *Event description:* brief description of the event;
3. *Event chain:* the event chain number for this event;
4. *Pred:* predecessor(s) of the event;
5. *Dead-end:* number of the event causing this event to become a dead-end (events that are not dead-ends have a blank in this column);
6. *Event comments:* additional comments and solution strategies related to the event;
7. *K:* knowledge type (one of D for data, P for Process, and I for Innovation);
8. *Src:* data source used to determine this event (L for developer log, M for meeting, E for email archive);
9. *PT:* hours of productive time for this event;
10. *NT:* hours of non-productive time for this event;
11. *TT:* total number of hours spent on the event;
12. *T/C:* Indicates if the event concerned mainly a tool (indicated by T) or a component (indicated by C);
13. *Tool Time:* amount of hours involved in learning a tool;
14. *Cmp Time:* amount of hours spent learning a component.

University of Cape Town

Table A.1: Knowledge events for P1-1.

Event No.	Event description	Event Chain	Pred	Dead end	Event comments / solution description	K	Src	PT	NT	TT	T/C	Tool Time	Cmp Time
0	**<Root>**				This is the starting point of development							0	0
1	What computers are we to use?	1	0		Windows lab computers	P	M	0.10	0.00	0.10	T	0.10	0.00
2	What embedded platform should we use?	1	0		Cogent CSB337	P	M	0.10	0.00	0.10	C	0.00	0.10
3	Cannot connect to network	2	1		Supervisor helped to fix proxy configuration for lab PC	P	M	0.10	0.20	0.30	C	0.00	0.30
4	Cannot seem to access Linux server	2	3		Supervisor reset passwords for developer.	P	E	0.05	0.20	0.25	T	0.25	0.00
5	Physically connect CSB337 to PC	3	2, 4		Use either a RS232C null link (NL) modem connection. Alternatively, use an Ethernet cross-over cable	P	M	0.20	0.90	1.10	C	0.00	1.10
6	Where can we get cables?	3	5		Technical officer	P	M	0.10	0.00	0.10	C	0.00	0.10
7	PC software to talk to CSB337 over RS232C?	3	6		Various free and shareware options available. Must run on Windows. Hyperterm may provide all the needed features.	D	L	0.20	0.40	0.60	T	0.60	0.00
8	Decided on comms software to use	3	7		Use HyperTerminal. Must create config file.	P	L	0.15	0.05	0.20	T	0.20	0.00
9	RS232C protocol settings	3	8		Look through manual for prot settings. 38400bps, 8 data bits, no parity, one stop bit, no flow control.	D	L	0.60	0.10	0.70	C	0.00	0.70
10	Info about testing board?	3	8		Found ref to Ed Stutter's book and website with umon source. Use MicroMonitor (boot loader).	D	L	0.30	0.60	0.90	T	0.90	0.00
11	Inversitage umon source	4	10		Download umon source from web. Has nice manual for commands. But it's a lot of reading to get through -- might use it later.	D	L	1.30	0.00	1.30	T	1.30	0.00

University of Cape Town

12	How to use MicroMonitor to test the CSB337?	3	9, 10		Stat, arp, etc. Most of the time (say 70%) was spent playing about with commands which were not used.	P	L	0.30	0.80	1.10	T	1.10	0.00
13	Decided to ignore umon source for now.	4	11	13	Commands seem simpler enough. Will put the umon source aside and tidy directory.	P	L	0.00	0.10	0.10	T	0.10	0.00
14	What software tools should be used?	5	2		Supervisor suggests Open Office or MS Word for documentation. Requirements C or ASM.	P	E	0.20	0.15	0.35	T	0.35	0.00
15	Software development tools for CSB337?	5	14		Browsed through CSB337 manual - much of it did not mention s/w - mentioned commercial tools: do we have it? - Mentioned MicroMonitor	D	L	0.30	0.60	0.90	T	0.90	0.00
16	Commercial development tools for CSB337?	5	15	17	Asked supervisor if we have commercial tool mentioned. -- reply: no	D	M	0.00	0.50	0.50	T	0.50	0.00
17	Decided to use MicroMonitor for dev tools	5	12, 15		Conclusion: use MicroMonitor. Considered pros and cons of using other bootloader or operating systems, but decided to keep it simple. Need to select toolchain for compiling.	P	L	0.80	0.50	1.30	T	1.30	0.00
18	Toolchain	6	17		Looking for toolchain. Search web, found arm-linux. Perhaps supervisor has a toolchain?	D	L	0.20	0.30	0.50	T	0.50	0.00
19	Toolchain: arm-linux	6	18		Found sources for Linux and compiler sources. Don't want Linux source, just compiler executable for windows.	D	L	1.20	0.00	1.20	T	1.20	0.00
20	Windows arm-linux compilers for ARM?	6	19		Seems to come with MicroMonitor disk, but we don't have it. Looked some more on web.	D	L	0.90	0.00	0.90	T	0.90	0.00

University of Cape Town

21	Method to develop software	7	18		Supervisor made toolchain available on Linux server. Work on code (edit, compile, etc) using ssh, then download to lab PC using winscp. Gave up looking for a Windows compiler.	P	E	0.40	0.20	0.60	T	0.60	0.00
22	Example C program for CSB337	8	21		Looking for example C code, preferably written for for MicroMonitor and that can send/receive chars over RS232. Asked supervisor, and he provided an example.	D	L	0.30	0.70	1.00	C	0.00	1.00
23	Decide to archive arm-linux	6	20	23	Example C and supervisor's toolchain looks OK. Will zip arm-linux incase needed later.	P	L	0.15	0.07	0.22	T	0.22	0.00
24	Makefile: how to use?	9	22		Cannot remember how to do makefiles, will look at examples from umon and on web.	D	L	0.00	0.10	0.10	T	0.10	0.00
25	Makefile from umon source?	9	24	26	Spent some time looking at makefiles but cannot find for CSB337 platform. Not provided in this generic version. Gave up.	D	L	0.00	0.60	0.60	T	0.60	0.00
26	Makefile	9	24		Found tutorial on makefiles. Read a few things not really needed. Should manage now.	D	L	1.20	0.40	1.60	T	1.60	0.00
27	Compiling example program	9	26		Writing make file to compile program. And tested compiling.	I	L	1.00	0.00	1.00	T	1.00	0.00
28	Install program on CSB337	10	24		Want to get executable onto the CSB337.	P	E	0.20	0.00	0.20	T	0.20	0.00
29	Considering install options	10	28		Supervisor says to upload with umon xmodem command, or it's faster to upload using tftp.	D	E	0.15	0.00	0.15	T	0.15	0.00
30	Install program using TFTP	10	29	30	Could not get an IP number. Don't have second eth card because sys admin not available to install one. Will just use hyperterm.	P	E	0.00	0.80	0.80	T	0.80	0.00

University of Canterbury

31	Installing program using xmodem	10	29		Took a while to figure out how exactly to do it. Need the right flags set in umon, and Xmodem settings configured in HyperTerm.	P	L	0.30	0.90	1.20	T	1.20	0.00
32	Why does program crash?	11	31		Program won't load. Wrong format? Turns out that the 'e' flag needs to be set when downloading using Xmodem.	P	L	0.20	1.00	1.20	T	1.20	0.00
33	Looking through files for cause of crash.	11	32	34	Program crashes. Tried recompiling, re-uploading, etc. No luck. Will ask supervisor.	P	E	0.10	0.50	0.60	T	0.60	0.00
34	Compiling example program	9	31		Supervisor looked at code. Wrote a new makefile because it was a mess and fixed linking problems.	I	M	0.80	0.10	0.90	T	0.90	0.00
35	Toss earlier makefile	9	27	34	Tossed earlier makefile because new one works better.	P	M	0.00	0.10	0.10	T	0.10	0.00
36	Running program	12	34		Example application now runs. Testing.	P	L	0.25	0.00	0.25	T	0.25	0.00
37	Reading user interface spec	13	36		Writing user interface code. Evaluating UI design to see how to do this. Need to check some terms.	D	L	0.80	0.60	1.40	C	0.00	1.40
38	User interface	14	37		Writing UI	I	L	1.20	0.40	1.60	C	0.00	1.60
39	How to send bytes	15	37		Sending bytes was quite easy; the solution was in the example code.	D	L	0.20	0.00	0.20	C	0.00	0.20
40	How to read bytes	16	39		Took a bit more time to figure out. Had to investigate monitor.h file.	D	L	0.10	0.50	0.60	C	0.00	0.60
41	Command processor decisions	17	40		Will need to develop a state machine, build up command strings using command buffer. Wasted some time trying alternate ways.	P	M	0.40	0.20	0.60	C	0.00	0.60
42	Command processor implementation	17	41		Display command prompt. Parse input characters and output replies.	I	L	1.10	0.20	1.30	C	0.00	1.30
43	Startup	18	42		Display welcome message when program starts. Displays prompt if in interactive mode.	I	L	1.00	0.15	1.15	C	0.00	1.15

University of Time Tech

44	Handle input (nb_readLine function)	17	43		State machine triggered on character input. Enter returns 1 to tell main loop to call command parser. Wasted some time figuring out how to prevent blocking calls -- simply use mon_getchar().	I	L	1.50	0.30	1.80	C	0.00	1.80
45	Command parser (Parse_String function)	17	44		Just used a set of string compares, called corresponding functions. Wrote stub functions for handlers.	I	L	1.30	0.25	1.55	C	0.00	1.55
46	Command parser	17	45		Optimizations	I	M	0.60	0.20	0.80	C	0.00	0.80
47	Jump table optimization	19	46		Experimenting with a jump table.	I	L	1.60	0.00	1.60	C	0.00	1.60
48	Looking at LED control	20	46	48	Could be good to use LED port for doing timing. No easy sample code. Will leave for now.	P	L	0.00	0.50	0.50	C	0.00	0.50
49	Removed jump table	19	47	49	Could not get jump table working properly, reverted to old version of code.	I	L	0.00	1.00	1.00	C	0.00	1.00
50	Command parser	17	45		Need to handle parameters for commands (e.g. frequency for freq command)	P	L	0.30	0.00	0.30	C	0.00	0.30
51	String to integer conversion	21	50	52	Trying to find sscanf equivelent to convert the parameter to an integer. Does not want to link as function is not declared. Trying to find implementation on web; not much help.	D	L	0.00	0.50	0.50	C	0.00	0.50
52	String to integer conversion	21	50		The itoa is working instead.	P	L	0.10	0.00	0.10	C	0.00	0.10
53	Integer to string conversion	21	52		Using atoi	P	L	0.10	0.00	0.10	C	0.00	0.10
54	I2C Interface for actuator board containing DAC	22	2		Need to determine how to connect the CSB337 to the DAC. Need to find datasheets.	P	L	0.20	0.00	0.20	C	0.00	0.20
55	DAC interface	22	54		Searching web for DAC datasheet. Found easily enough. Now reading through. Most of it was rather irrelevant.	D	L	0.10	0.40	0.50	C	0.00	0.50

University of Cape Town

56	Pins to use on DAC	22	55		Need to determine what pins to use.	P	L	0.20	0.20	0.40	C	0.00	0.40
57	CSB337 I2C port	22	56		Read CSB337/AT91RM9200 datasheet to find info about I2C port. Lot of stuff.	D	L	0.20	0.60	0.80	C	0.00	0.80
58	Connecting up the actuator board	23	57		Got jumper cable from supervisor, stripped other end. Installed DAC on breadboard.	I	L	0.60	0.20	0.80	C	0.00	0.80
59	Powering actuator board	24	58		Try power supply directly	P	L	0.50	0.00	0.50	C	0.00	0.50
60	Powering actuator board	24	59	60	Decide to utilize regulator in design	P	M	0.00	0.50	0.50	C	0.00	0.50
61	I2C software interface	25	58		Searching for example code for this.	D	L	0.20	0.60	0.80	C	0.00	0.80
62	I2C polling sw interface	25	61		After some effort, got polling I2C interface to send and receive. Implemented own buffer.	I	L	1.20	0.70	1.90	C	0.00	1.90
63	I2C alternatives?	25	62		Looking for sample code for I2C interrupt	D	L	0.40	0.10	0.50	C	0.00	0.50
64	Command processor help cmd	17	53		Need help command to list commands and how to use each one.	I	L	1.00	0.25	1.25	C	0.00	1.25
65	Command processor	17	64		Need to load custom signal and have additional command to do so. Put in code instead of dowloading.	I	L	1.30	0.60	1.90	C	0.00	1.90
66	I2C interrupt sw interface	25	63		Want to improve the I2C interface so that an ISR handles the input. Much time spent.	I	L	4.50	0.00	4.50	C	0.00	4.50
67	Gave up on I2C interrupt	25	66	67	Decided to give up on the I2C interrupt, but improved the polling implementation.	I	L	0.00	0.20	0.20	C	0.00	0.20
68	Timing	26	62		Can CPU clock be used as timer?	P	M	0.30	0.10	0.40	C	0.00	0.40
69	Make improvements to communications	27	65		Made various improvements to the communications. More robust.	I	L	0.50	0.30	0.80	C	0.00	0.80
70	Minor improvement to actuator board	23	62		Improved power supply to reduce noise on outputs.	I	L	0.40	0.50	0.90	C	0.00	0.90
71	Timing improvements	26	68		Finding info on onboard time.	D	L	1.00	0.00	1.00	C	0.00	1.00
72	Test timing improvements	26	71	72	Experimented with using timer based on datasheet. Too little time; gave up.	P	L	0.00	0.70	0.70	C	0.00	0.70
73	Tidied up final prototype	28	72, 70		Readied system for demo and tidied up code	I	L	1.20	0.30	1.50	C	0.00	1.50
74	**<Final>**	**28**	73		This is the ending point of the project								

University of Colorado

A.2 Knowledge register for the second case study (P1-2)

The table below shows each event for the second case study (P1-2). Please refer to A.1 for a description of the columns.

Table A.2: Knowledge events for P1-2.

Event No.	Event description	Event Chain	Pred	Dead end	Event comments / solution description	K	Src	PT	NT	TT	T/C	Tool Time	Cmp Time
0	**<Root>**				This is the starting point of development							0	0
1	What components should be used for interface board?	1	0		Searched internet, requested prices from supplier	D	L	2.00	0.00	2.00	C	0.00	2.00
2	How should the files be managed?	2	0		Supervisor says to keep work files on local workstation, use samba to copy to server	P	M	0.10	0.00	0.10	T	0.10	0.00
3	Problem mounting samba	3	2		Had to recompile kernel on developer PC	P	L	2.00	0.00	2.00	T	2.00	0.00
4	How to connect up CSB337	4	0		Had to figure out how to connect up CSB337	P	L	0.20	0.00	0.20	C	0.00	0.20
5	How to communicate with CSB337 over RS232?	4	4		Need to find a terminal program for Linux. Had short look, but no luck yet. Try again later.	D	L	0.00	0.50	0.50	T	0.50	0.00
6	Power supply problems	5	4	7	Power supply seems faulty. Testing it out.	P	L	1.40	0.00	1.40	C	0.00	1.40
7	Replace power supply	5	4		Replaced power supply, some time to find.	P	L	0.20	0.30	0.50	C	0.00	0.50
8	Software for CSB337?	6	4		MicroMonitor is a bootloader provided with it	D	M	0.20	0.00	0.20	T	0.20	0.00
9	Learn about MicroMonitor	7	8		Find umon site. Download umon source	D	L	0.25	0.30	0.55	T	0.55	0.00
10	Software development options	8	8		Need to decide if an O/S is needed, if we will use e.g. Embedded Linux or only umon	P	M	0.20	0.00	0.20	T	0.20	0.00
11	Linux options	9	10		Emailed cogent requesting info re Linux. Goodled for "linux toolchain csb337" (found various options, nothing of use)	D	L	0.10	0.50	0.60	T	0.60	0.00

University of Cape Town

12	RTEMS as O/S?	10	10	12	Visited RTEMS, but decided not to use it	D	L	0.00	0.30	0.30	T	0.30	0.00
13	RS232 comms	4	5		Investigate ckermit	D	L	0.10	0.30	0.40	T	0.40	0.00
14	Use ckermit for RS232 comms	4	13		Decided to use ckermit for comms	P	L	0.10	0.00	0.10	T	0.10	0.00
15	Using MicroMonitor (1)	11	9	15	Googled for MicroMonitor howto, found only index of training guide.	D	L	0.00	0.50	0.50	T	0.50	0.00
16	Using MicroMonitor (2)	12	9		Read general documentation supplied with umon source, most not relevent to what I'm wanting to do.	D	L	0.20	0.80	1.00	T	1.00	0.00
17	How to communicate with CSB337 over Network?	13	14		Want to use network for faster transfer. Need to organize an IP.	P	M	0.15	0.00	0.15	C	0.00	0.15
18	IP for CSB337	13	17		Requested IP address from UCT ICTS	P	L	1.00	0.00	1.00	C	0.00	1.00
19	Assign IP to CSB337	13	18		Doesn't seem to get the IP. Tried various things. Worked using: set IPADD DHCP	P	L	0.50	0.00	0.50	C	0.00	0.50
20	Linux options	9	11		Googled for "linux on arm", found kernel for Cogent on www.linux.arm.org.uk	D	L	0.50	0.00	0.50	T	0.50	0.00
21	Program for doing diagrams	14	1		Tried various options, decided on dia	D	L	0.50	0.00	0.50	T	0.50	0.00
22	Interface board	15	21		Working on schematic for HW/HW interface	I	L	2.00	0.00	2.00	C	0.00	2.00
23	Arm-Linux	16	20		Downloaded installation guide and source for ARM-LINUX	D	L	2.00	0.00	2.00	T	2.00	0.00
24	TFTP Problems	13	19		Problems using TFTP on UCT network. May need to install second network card and set up a local net.	P	L	0.10	0.20	0.30	C	0.00	0.30
25	TFTP	13	24		Looking on web for solutions. Trying to recompile developer PC kernel for tftp.	D	L	0.50	1.70	2.20	C	0.00	2.20
26	Linux options	9	20		Looking for option options. Found emdebian toolchain, started download.	D	L	0.10	0.20	0.30	T	0.30	0.00
27	EmDebian	17	26	38	Download didn't work, so installed each package one at a time.	D	L	2.70	0.00	2.70	T	2.70	0.00

University of Cape Town

28	TFTP	13	25		TFTP still not working. Seems like wrong version of kernel installed	D	L	0.00	0.50	0.50	T	0.50	0.00
29	How to install kernel	9	28		Reading-up on how to install kernel	D	L	1.00	0.00	1.00	T	1.00	0.00
30	Parts for interface board	18	22		Searching web for info on parts. Reading datasheets for ideas. Requesting quotes.	D	L	0.25	0.90	1.15	C	0.00	1.15
31	Installing kernel	9	29		Dowloading, fixing: lilo, mouse, xserver. Much time wasted on network -- could not fix.	D	L	1.30	4.50	5.80	C	0.00	5.80
32	Fixing network	19	31	33	Search google, etc for help with network.	D	L	0.00	1.33	1.33	C	0.00	1.33
33	Fixing network	19	31		New net card installed. Looked for some settings. Got it to work.	P	L	0.50	0.20	0.70	C	0.00	0.70
34	TFTP	13	28	35	Now, with new kernel and net network, trying again to get TFTP working. No luck.	P	L	1.00	0.00	1.00	C	0.00	1.00
35	ATFTP	20	19		Set up atftp, it works fine. Able to upload executable to cogent board.	P	L	0.35	0.00	0.35	C	0.00	0.35
36	Executing example application on MicroMonitor	21	35		Attempting to run hello.elf on MicroMonitor. But no luck.	P	L	0.00	0.90	0.90	T	0.90	0.00
37	Fixing linker options	22	36		Simon found linker options were wrong.	P	M	0.20	0.00	0.20	T	0.20	0.00
38	Compiling Snapgear	23	20		Found snapgear linux. Seems better. Decided to toss emdebian. Attempting to compile snapgear. Gives errors.	P	L	0.10	0.40	0.50	T	0.50	0.00
39	Installing uCliux rpms	24	38		Installing uClinux tools	P	L	0.15	0.00	0.15	T	0.15	0.00
40	Compiling Snapgear	23	39		Attempting to compile snapgear using the uClinux tools. Get different errors now.	P	L	0.70	0.00	0.70	T	0.70	0.00
41	Compiling Snapgear	23	40	42	Searching for solution to errors on web	D	L	0.20	0.00	0.20	C	0.00	0.20
42	Compiling Snapgear	23	38		Trying yet another toolchain (m68k-elf-tools)	P	L	1.20	0.00	1.20	T	1.20	0.00
43	Compiling Snapgear	23	42		Attempting to compile snapgear. After various fixes, got it to compile. When run on csb337, caused crc error.	P	L	1.40	0.00	1.40	T	1.40	0.00

University of Maine

44	Compiling Snapgear	23	43	45	Reading howtos, web, man, etc to solve crc problem.	D	L	1.10	0.00	1.10	C	0.00	1.10
45	Gave up on Snapgear	23	20	45	Searched for fixes for Snapgear; decided to give up on it after not finding anything useful.	D	L	1.00	0.00	1.00	C	0.00	1.00
46	Arm-linux new option found	25	20		Attempting to get arm-linux working. Cannot get it to compile.	P	L	0.20	0.60	0.80	T	0.80	0.00
47	Compiling arm-linux	26	46		Trying to find information to get arm-linux to compile.	D	L	0.10	0.65	0.75	T	0.75	0.00
48	Compiling arm-linux	26	47	49	Experimenting with settings to get arm-linux to compile.	P	L	1.00	0.90	1.90	T	1.90	0.00
49	Compiling arm-linux	26	47		Tried different toolchain for compiling	P	L	0.50	0.00	0.50	T	0.50	0.00
50	Installing arm-linux	27	49		Compile succeeded, but when executes, cannot mount root fs.	P	L	0.50	0.00	0.50	T	0.50	0.00
51	Booting arm-linux	28	50		Found that ramdisk needs to be installed.	D	L	0.30	1.20	1.50	C	0.00	1.50
52	Ramdisk	29	51		Found and downloaded ramdisk	D	L	0.30	0.20	0.50	C	0.00	0.50
53	Installing ramdisk	30	52		Find info about configuring ramdisk	D	L	0.40	0.10	0.50	C	0.00	0.50
54	Installing ramdisk	31	53		Trying to configure and install ramdrive	P	L	2.33	2.50	4.83	C	0.00	4.83
55	DC motor interface	32	30		Researching DC motor drive	D	L	0.40	0.60	1.00	C	0.00	1.00
56	AC-DC circuit	33	30	57	Constructed AC-DC circuit	I	L	1.00	0.00	1.00	C	0.00	1.00
57	AC-DC circuit	33	55		Problem with AC-DC circuit; rebuilt it	I	L	1.00	0.00	1.00	C	0.00	1.00
58	Examine busybox	34	20		Considering busybox for use on csb337	D	L	0.50	0.00	0.50	T	0.50	0.00
59	Installing arm-linux	27	54		Determined procedure to mount ramdrive on PC and modify the settings, by editing inittab. Got linux running on the csb337.	P	L	1.30	0.00	1.30	T	1.30	0.00
60	csb337 linux networking	35	59		Trying to get csb337 network going	P	L	0.00	0.30	0.30	C	0.00	0.30
61	Revisiting snapgear	23	45	62	Compiling and trynig to install snapgear. No luck will ask cogent.	P	L	1.00	0.30	1.30	T	1.30	0.00
62	New snapgear toolchain	36	45		Found and downloaded new snapgear	D	L	0.30	0.60	0.90	T	0.90	0.00
63	Imeplemented envolope detector method	37	0		Implemeted and tested method for envolope detector on PC	I	L	2.00	1.00	3.00	C	0.00	3.00

University of Cape Town

64	ADCS research	38	57		Studying ADCS datasheets	D	L	0.35	0.00	0.35	C	0.00	0.35
65	New snapgear toolchain	36	62	66	Compiing snapgear. Uninstall emdebian.			1.10	0.00	1.10	T	1.10	0.00
66	New snapgear toolchain	36	62		Failed to compile snapgear; searching for answers. Some success.	D	L	0.30	0.30	0.60	T	0.60	0.00
67	Busybox	39	58		Got busybox working on csb337	P	L	0.30	0.70	1.00	T	1.00	0.00
68	Device drivers	40	66		Researching device drivers for emb linux	D	L	0.30	0.70	1.00	C	0.00	1.00
69	Device drivers	40	68		Test sample device driver on csb337	P	L	0.70	0.60	1.30	C	0.00	1.30
70	Device drivers	40	69		Developing experimental device driver	I	L	0.50	0.00	0.50	C	0.00	0.50
71	Device drivers	40	70		Read-up on device drivers	D	L	0.20	0.80	1.00	C	0.00	1.00
72	LED driver	41	71		Managed to get simple LED driver to work	I	L	0.70	1.30	2.00	C	0.00	2.00
73	AT91RM9200	42	72		Read-up on AT91RM9200	D	L	0.50	0.50	1.00	C	0.00	1.00
74	i2c driver	43	73		Developed i2c driver running on snapgear	I	L	2.00	3.00	5.00	C	0.00	5.00
75	i2c driver works	43	74		Managed to get i2c driver to work	I	M	1.00	2.00	3.00	C	0.00	3.00
76	Antenna control code	44	75, 63		Making antenna rotate via s/w control	I	M	3.00	1.00	4.00	C	0.00	4.00
77	User interface	45	76		Code to start/stop/change direction of rotation via RS232	I	M	3.00	0.33	3.33	C	0.00	3.33
78	**<Final>**	**45**	77		This is the ending point of the project					0.00		0.00	0.00

University of Cape Town

Appendix B: Experiment 2 appendices

In order to save space, only the knowledge registers for the first project in Experiment 2 is provided (the knowledge registers of the other projects followed the same structure).

B.1 Knowledge register for Project P2-1

As explained in Chapter 4, the knowledge register structure for the second experiment was refined. The table below presents the list of ESAOA events for the first project in the second experiment (P2-1). The 20 columns of this table are described below:

1. *No.*: indicates the number of the data entry.
2. *Type*: indicates the data source, e.g., email, project discussion forum, developer log, etc.
3. *Project*: identifies the team number.
4. *Event chain*: links the knowledge event to other knowledge events in the chain.
5. *Phase*: the duration of the project was divided into eight phases – the phase number indicates in which of the eight phases the knowledge occurrence took place
6. *KD*: indicates data knowledge events
7. *KP*: indicates process knowledge events
8. *R*: indicates process knowledge that related to roles.
9. *L*: indicates process knowledge that related to logistical issues.
10. *KI*: indicates innovation knowledge events
11. *PK*: indicates a productive knowledge event
12. *NPK*: indicates a non-productive knowledge event
13. *Artefacts*: indicates the components used
14. *Comments*: researcher's comments
15. *DS con*: data steward (DS) contributed to this occurrence (*)
16. *PE con*: process engineer (PE) contributed to this occurrence (*)
17. *IE con*: innovation engineer (IE) contributed to this occurrence (*)
18. *Delta*: 1 if either (a) DS contributed process or innovation knowledge or (b) PE contributed data or innovation knowledge or (c) IE contributed data or process knowledge (*)
19. *Tool*: the engineer/team learned to use a tool in this occurrence (*)
20. *Component*: the engineer/team learned to use a component in this occurrence (*)

University of Cape Town

(*) NOTE: All columns marked (*) were added to Projects P2-1, P2-2 and P2-10 only in order to perform further analysis on the contribution of knowledge occurrences according to roles (see Section 5.5.6).

Table B.1: Knowledge events for P2-1.

No.	Type	Proj-ect	Event Chain	Phase	KD	KP	R	L	KI	PK	NPK	Artifacts: Tools	Artifacts: components	Comments	DS con	PE con	IE con	Delta	Tool	Comp-onent
1	E	1	1	1		1	1			1			Role responsibilities		0	1	0		0	0
2	E	1	1	1		1	1			1					0	1	0		0	0
3	E	1	2	1		1	1			1			EM303		1	1	0	1	0	1
4	E	1	2	1		1	1			1					0	1	0		0	0
5	E	1	2	5	1					1					1	0	0		0	0
6	E	1	2	3		1	1				1			upset about group member changing the roles	0	1	0		0	0
7	E	1	3	1	1					1				Re: What to access...	1	0	0		0	0
8	E	1	4	1		1		1		1				Decide venue	0	1	0		0	0
9	E	1	4	1		1		1			1			Source GPS & GPRS modules	1	0	0	1	0	0
10	E	1	5	2		1	1			1			GPS module motorola oncore 8.4		0	1	0		0	1
11	P	1	6	2					1		1			see 58.6	0	0	1		0	0
12	P	1	6	2					1		1			see 58.6	0	0	1		0	0
13	E	1	7	2	1						1				1	0	0		0	0
14	E	1	7	2	1						1				1	0	0		0	0
15	P	1	8	2	1					1				looking at GPS modules	1	1	0	1	0	0
16	P	1	8	2					1	1					0	0	1		0	0
17	P	1	8	2	1						1				1	0	0		0	0
18	P	1	8	2					1	1					0	0	1		0	0
19	P	1	8	2					1	1				Initial version of program using GPS wrapper	0	0	1		0	0

University of Cape Town

20	E	1	9	2		1	1			1					0	1	0		0	0
21	E	1	10	2	1					1			OpenOffice.org	Relates to data 1c	1	0	0		0	1
22	E	1	11	2		1		1			1			Group meet because they were trying things that didn't work	0	1	0		0	0
23	E	1	11	2		1		1		1				Review choice of tools and components	0	1	0		0	0
24	E	1	12	2		1					1				0	1	0		0	0
25	E	1	12	2		1				1		enter_ESA OA		Success	0	1	0		1	0
26	E	1	12	2					1	1					0	0	1		0	0
27	E	1	13	2		1					1				0	0	0	1	0	0
28	E	1	14	3	1						1				1	0	0		0	0
29	E	1	14	3	1					1					1	0	0		0	0
30	E	1	14	3	1					1				was useful data provided to him earlier	1	0	0		0	0
31	E	1	14	6	1					1			MPI		1	1	0	1	0	1
32	E	1	15	5					1	1					0	0	1		0	0
33	E	1	15	5					1	1		vcLinux	Interrupts		0	0	1		1	1
34	E	1	16	3	1					1					0	1	0	1	0	0
35	E	1	16	3	1						1				1	0	0		0	0
36	E	1	16	6	1					1			GPS		0	0	1	1	0	1
37	E	1	16	6	1					1			GPS		0	1	0	1	0	1
38	E	1	16	6	1					1			GPS		0	1	0	1	0	1
39	E	1	17	3	1					1			Trimble GPS module, uCLinux bluetooth support, USB dongel, GPS module motorola oncore 8.4	checking a information	0	1	0	1	0	1
40	G	1	17	3					1	1			uCLinux bluetooth support		0	0	1		0	1
41	E	1	17	6		1				1			Trimble GPS module		0	1	0		0	1

University of Cape Town

42	E	1	17	6		1					1				0	1	0		0	0
43	E	1	17	6		1				1			VCLinux, bluetooth support, USB dongel		0	1	0		0	1
44	E	1	18	4		1					1	Hypertermi nal	Disokoleds.c, RS232, Wled		0	1	0		1	1
45	E	1	18	4		1				1		Micromonit or			0	1	1		1	0
46	E	1	18	4		1				1					0	1	1		0	0
47	E	1	18	6		1				1			PC interface for GPS module		1	0	0	1	0	1
48	E	1	18	8					1	1		vcLinux	DAC, circuit		0	0	1		1	1
49	E	1	18	8					1	1		ESAOA vcLinux	DAC, war files		0	0	1		1	1
50	E	1	18	8		1				1		mount-iso, umount-iso, ESAOA			0	0	0	1	1	0
51	E	1	18	8		1					1	vcLinux ftp		cannot execute elf files downloaded using ftp	1	0	0	1	1	0
52	E	1	19	8					1		1	ucLinux	timer		0	0	1		1	1
53	E	1	19	8					1	1		vcLinux	DAC, circuit		0	0	1		1	1
54	E	1	19	8					1	1					0	0	1		0	0
55	E	1	20	8					1	1		vcLinux	Test Driver		0	0	1		1	1
56	E	1	21	6					1	1				Success	0	0	1		0	0
57	E	1	21	8		1					1	vcLinux	MemDerInfo		0	1	0		1	1
58	E	1	22	8					1	1		ucLinux	MemDev		0	0	1		1	1
														Totals:	**16**	**25**	**20**	**13**	**13**	**21**

University of Cape Town

B.2 Requirements check sheets for Experiment 2

The table below provides the requirement check sheet ratings for each project of Experiment 2.

PROJECT REQUIREMENTS CHECKSHEET

Issue	Max	Section Total	Category Weight	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13
1. FUNCTIONAL REQUIREMENTS *Note: Rating of functional requirements for a particular product are based*			**44%**	**90%**	**77%**	**75%**	**78%**	**90%**	**81%**	**65%**	**51%**	**64%**	**70%**	**74%**	**83%**	**66%**
Hardware		**50**		**48**	**33**	**37**	**48**	**50**	**45**	**34**	**23**	**40**	**30**	**47**	**46**	**50**
Prototype assembly - robustness & elegance - for enclosure, see below	20			18	18	12	20	20	19	14	8	18	18	19	20	20
Schematics and/or circuit diagrams	30			30	15	25	28	30	26	20	15	22	12	28	26	30
Software / Program Execution		**70**		**60**	**59**	**53**	**45**	**58**	**52**	**44**	**38**	**37**	**54**	**42**	**54**	**29**
- Program runs smoothing: no penalties - Program hangs after a while or handles invalid/valid input poorly or terminates unexpectedly: up to 20 points removed - Program does not run: *see penalty notice below*				0	0	0	0	0	0	0	-5	-5	0	-2	0	0
Start-up message displayed	2			2	2	2	2	2	2	2	2	2	2	2	2	2
Argument parsing works	6			6	6	6	6	6	3	3	6	1	6	6	2	6
Menu works	14			6	6	6	6	6	6	4	6	0	3	6	6	0
Menu ease of use (6=user friendly; 0=very user unfriendly)	6			6	6	4	4	3	2	5	5	6	1	3	5	0
Communications	21			19	21	19	11	21	19	11	6	18	21	6	21	21
Event & service processing	21			21	18	16	16	20	20	19	18	15	21	21	18	0
2. TEMPORAL REQUIREMENTS: QUALITY OF SERVICE (QoS) (real-time			**11%**	**97%**	**63%**	**70%**	**63%**	**97%**	**37%**	**53%**	**67%**	**67%**	**100%**	**73%**	**57%**	**87%**
Total quality of service points		**30**		**29**	**19**	**21**	**19**	**29**	**11**	**16**	**20**	**20**	**30**	**22**	**17**	**26**
Predictability future response of system is predictable	6			6	6	4	2	5	2	5	3	2	6	2	3	3
Speed product is responsive	2			2	0	1	1	2	2	2	2	2	3	2	1	2
Timeliness time between the occurrence of an event	4			4	3	4	2	4	2	4	3	3	3	4	3	3
Throughput max no. of jobs/time unit	3			3	2	3	3	3	2	2	3	3	3	3	3	3
Responsiveness worst-case latency not exceeded	3			3	2	2	3	3	0	1	1	2	3	3	1	3
Capacity ability of system to meet all hard deadlines;	4			3	0	1	2	4	0	1	2	2	4	2	1	4
Reliability / sustainability system keeps important deadlines even in overload conditions	4			4	2	2	2	4	1	1	2	2	4	2	1	4
Safety where applicable, account for safety issues	2			2	2	2	2	2	0	0	2	2	2	2	2	2
Security extent to which product is protected against security misconduct	2			2	2	2	2	2	2	0	2	2	2	2	2	2
3. QUALITY OF ARTEFACTS			**40%**	**96%**	**94%**	**89%**	**80%**	**94%**	**85%**	**73%**	**73%**	**55%**	**62%**	**57%**	**66%**	**72%**
ESAOA Repository		**20**		**20**	**19**	**20**	**20**	**19**	**20**	**20**	**20**	**14**	**4**	**15**	**16**	**16**
All files within project root.	3			3	3	3	3	2	3	3	3	2	1	3	3	3
Artifact Organization Diagram (AOD)	14			14	14	14	14	14	14	14	14	9	0	9	10	10
Location of files	3			3	2	3	3	3	3	3	3	3	3	3	3	3
Code		**80**		**76**	**74**	**69**	**59**	**75**	**71**	**51**	**51**	**41**	**58**	**39**	**57**	**53**
Conformance to ANSI C	5			5	5	5	5	5	5	5	5	5	5	5	5	5
Use of comments	10			6	9	5	5	7	5	3	5	5	6	3	8	3
Module interfaces	20			20	18	18	8	20	20	15	15	10	18	6	18	18
Required modules																
- Main module	3			3	3	3	3	3	3	3	3	3	3	3	3	3
- Encoding of welcome / version information	4			4	4	4	4	4	4	2	4	4	4	4	4	4
- Configuration settings and command parsing	6			6	6	6	6	6	6	5	6	4	6	6	6	6
- Menu component	8			8	8	8	8	8	6	6	8	4	6	6	7	2
- Service module(s)	12			12	10	12	9	10	11	0	0	0	0	0	0	0
- Drivers	12			12	11	8	11	12	11	12	5	6	10	6	6	12
Compiling and Linking		**10**		**10**	**10**	**9**	**9**	**9**	**2**	**9**	**9**	**6**	**6**	**9**	**0**	**10**
- No warnings or errors: all 10 points - Warnings: 5 points (if many warnings) - Minor error: 0 points; Major error: penalties	10			10	10	9	9	9	2	9	9	6	6	9	0	10
4. QUALITY OF ENCLOSURE			**5%**	**33%**	**80%**	**87%**	**33%**	**73%**	**73%**	**20%**	**33%**	**67%**	**####**	**53%**	**20%**	**####**
Enclosure		**15**		**5**	**12**	**13**	**5**	**11**	**11**	**3**	**5**	**10**	**15**	**8**	**3**	**15**
A rough enclosure is provided with labels indicating important controls and displays	15			5	12	13	5	11	11	3	5	10	15	8	3	15
TOTAL POINTS FOR PROJECT		**275**	**100%**	248	226	222	205	251	212	177	166	168	197	182	193	199
PERCENTAGES				90%	82%	81%	75%	91%	77%	64%	60%	61%	72%	66%	70%	72%

University of Cape Town

B.3 Comments from requirements check sheets for Experiment 2

The comments provided by the review panel that accompanied the requirements check sheets for each project are given below.

P1

Nice attention to detail. Excellent AOD. Compiles perfectly.
Like the help display. Menu works well. Provides menu shortcuts as well.
Communications pretty good, occasional missed packets / non-responses
Handles events well. QoS good, except misses occasional deadline when comms active, or pressing pushbutton button frequently.
Good prototype setup and quality block diag and schematics for circuit. Enclosure is rather lacking!

P2

A high standard of work. AOD looks good. But some files are not located so well in the framework.
No compiling problems. No communication problems, robust comms!
Handles events well. QoS reasonably, but just not fast enough. Have not optimized the main loop suitably, or your algorithms are not efficient. Did miss a few deadlines (timing from LED2 pin using osc.).
Lacking schematic / circuit diagram (rather quick, unclear and missing components used in the demo setup)
Prototype setup was acceptable.
Have made some effort on enclosure, especially labels.
Gave all safty points since your product does not have safty needs.

P3

Minor warnings during compile. Good quality of work (visual result esp).
Menu not very user-friendly, too many sub-menus.
Much confusion between signed and unsigned, also un-casted conversion of int to pointer types.
Minor comms problems, lost packets.
Does not meet all functional or QoS requirements sufficiently, mainly due to the comms glitch.
Prototype setup was messy and disorganized. Wires looked insecure.
Acceptable circuit diagram. Well-labeled enclosure.

P4

Very good AOD. Like the color changes of crossing lines in the AOD diagram.
Minor error in menu: upload me->you should be you->me
Menu could be more user-friendly (check over menu names thoroughly). Minor compiling problems.
Some comms would not get through (possible calculation problems in error checking?)
QoS requirements not met, timing is a bit out. (timed from LED2 track)
Insufficient use of comments. Need prototype setup for demo
Schematic acceptable. Effort on enclosure is lacking.

P5

AOD lacks detail, important artifacts. Warnings during compile.
Menu not very logically arranged, cryptic/too short names.
The menu works, but displays characters user did not type, for instance entering setup (s) echoes characters 'ssss'. System is responsive, deadlines with time to spare.
No communication problems observed.
Good and elegant schematic, nicely drawn.
Rather a lot of wires in prototype setup, but that is normal. Acceptable enclosure

P6

Some problems compiling. Argument passing problems, hangs expectantly on some arguments.
Menus cannot be navigated properly, gets stuck in sub menus (have to jump to keep jumping to main menu / the top level menu). Not user-friendly.
Although the deadlines are met, the system is not reliably and hangs expectantly, therefore the QoS ratings are low. Considering that a system for a car has some safety requirements, 0 points were given for safety.
Reasonably need demo setup. Acceptable circuit diagram. Acceptable enclosure.

University of Cape Town

P7

Good job on AOD. Couple of warnings during compile.

Argument parsing workings, but is not robust; gives a signal 11 on the arguments "-h 1 2 3".

Communications not very robust, occasional packet missed, not acked.

Dead-lines were met, but the non-robust comms lowers the QoS rating. Safety and security were given 0 as field sensors on a public transport vehicle would need to be secure and safe.

Messy setup for demo!

Schematic does not provide sufficient details on how things were connected, which pins on the csb337 board were used, etc.

Enclosure lacking. A good effort in all.

P8

Acceptable AOD. Could be neater. Some warnings during compile. Argument parsing acceptable.

Poorly coded driver modules.

Communications not very robust, frequent problems or ignored messages.

Met deadlines, but non-robust comms reduces QoS rating.

Prototype setup was too neat : indeed things were missing. Could have connected up a osc scope to demo timing, etc.

Your schematic is lacking detail, such as pin numbers of the csb337 board in addition to which PIO lines were used. A close-up photograph of your breadboard and connections would help some.

Enclosure has much room for improvement, no labels.

P9

AOD was not part of framework, was provided separately (although it was supposed to be included in the framework).

AOD is not drawn accurately or according to the guidelines, for example the application main module is shown as the root of the project, which is not correct – it is within the applications folder within the software folder.

Many warnings during compile. Argument parsing functions.

Occasional comms errors, possibly due to driver problems. Drivers inefficiently coded.

Events not handled accurately – occasional glitch. Reduced QoS rating. Ok demo setup.

Acceptable schematics, although they could be neater. Reasonable enclosure and labels.

P10

AOD was not provided. All files not within framework! Simple do a: tz within the Project10 directory under ESAOA environment, else do 'tar -zcf Project10.tar.gz Project10' to compress framework manually.

Check the logic of the menus: confusing, not user-friendly.

Many warnings during compile. Pointer problems. Application not robust.

Menu does not function properly; sometimes hangs when a sub-menu is exited, and you have to reset the product.

Dead-lines met, good QoS operation. Good prototype setup.

Circuit diagram lacks detail of which pins on the csb337 and DAC were used.

Excellent enclosure and well labeled. Comms works fine. Reasonably well coded drivers.

P11

AOD notation is incorrect. For example, Menu.c module is incorrectly shown as depending on the Utility folder, where it is contained in the utility folder.

Only a few minor compile warnings.

No shortcuts implemented in menu (e.g. Cannot issue the sequence 'sb 19200' for 'setup comms baud 19200')

Comms not robust. Does not work with large downloads of prices.

Dead-lines were met, but system was not predictable. Occasional odd behavior to inputs. Lack of input checks.

Neet demo setup. Acceptable circuit diagram.

Inefficient drivers, isr not robust. Enclosure not logically designed.

P12

The AOD is lacking important artefacts and has too few comments and descriptions as shown in the example.

Error encountered while compiling, took some time to remedy. It seems that something was possibly added to a module at the last minute, and the compile not retested, before the framework was submitted.

The software crashed frequently, was not robust.

Menu Ok and reasonably easy to use (didn't seem to cause crashes).

Comms worked Ok, and managed to accomplish the tests even though the drivers do not look elegant.

Missed a deadline during test (it did respond, but response was too late); the other QoS test were acceptable; but this problem has lowered the QoS rating for this project. Perhaps the requirements should have just indicated a larger response time; timings in multiples of 100ms would probably have been Ok for this product.

University of Cape Town

Events not elegantly handled, could use jump table or switch. Should be colocated in a clearly marked 'cricital' section. Must have made timing of the events difficult. Furthermore, there were LED2 toggles left in the code, which could not be disabled (using #undef TIMING).

Setup a little on the messy side, but you had the needed instrumentation setup and arranged suitably.

Acceptable circuit diagram. Drivers inefficiently coded and buggy.

Enclosure is lacking.

P13

The AOD is Ok, but it should have been in the Documentation folder.

A thorough job was done on the AOD, showing the folders and contents.
But few descriptions and comments; more would have helped explain the structure of the framework. The StdIO library module should not be shown as contained in the software components folder, but rather shown as an external dependency as it is contained within the uCLinux operating system directory (outside your ESAOA project framework).

Menu does not function well and does not suitably meet the requirements, and was consequently thought to be rather user-unfriendly.
The product does not meet its requirements. The services are not implemented fully, nor do they function well.

There are missing event handles; not all services are implemented.
Should at least respond with messages like "SERVICE xx not available" or removed items from the menu. Reduces the predictability of the system.

QoS issues were otherwise acceptable and deadlines were met.

Comms works fine.

Drivers work well, and are well structured and commented.

Excellent demo setup for prototype.

Good, elegant schematic. Excellent enclosure, good labels.

University of Cape Town

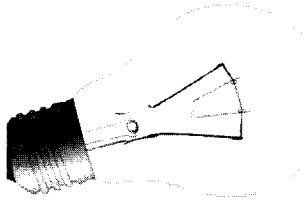
B.4 Evaluation forms used to rate code and design reviews

The forms below show samples of the forms that were filled out by the researcher during code and design reviews in order to evaluate: 1) the creativity of the prototype concept discussed during the first review; 2) the quality of the prototype design evaluated during the second review; and 3) the quality of the artefacts displayed during the final review.

B.4.1 Evaluation of concept creativity

The image below show a sample of the form filled in by the researcher to rate creativity of the teams proposed product during the first code and design review meeting.

CREATIVITY SCOREBOARD



TEAM: **3 (Vibynet)**

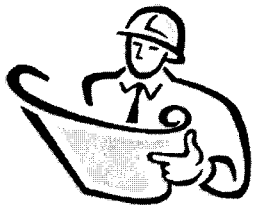
Each criterion is given a rating value from 0 to a maximum value. Fill in a number within this range to indicate how well the team has done.

Criteria Descirption	Rating	Boring = 0	MAX/2 = Middle of the road	MAX = Inspired!
How commonplace? Levels: 0 - 15	13	Very common / routine / everyday ideas		Unusual / novel / imaginative
The context. Levels: 0 - 5	4	Typical place of use, e.g. lab, home		Unusual place for use, e.g. aircraft,
Amount of 'playing with ideas' evident / variety of ideas contemplated: 0 - 10	7	Had few ideas / mostly focused on one initial idea and didn't consider broadening much else.		Thought of lots of things, was an effort settling focusing in on a single practical idea.
Application usefulness: 0 - 10	8	Not particularly useful / already easily obtained (e.g. can be bought cleaply In most shopping malls)		Very useful. Likely to be used frequently (e.g., every day), either hard to get / doesn't exist yet.
Level of challenge 0 - 5	5	Could be pieced togther quickly with little effort, little or no software coding needed.		Challenging. Still needs to be raighned in a whole lot more.
Sophistication of techniques & components needed: 0-10	7	Simple techniques / components		Developers likely to create own specialised, unusual processes.
Interfacing & connection to real-work: 0-5	5	Little or no connection to anything outside the system (even minimal or no HCI)		Connets to many external things, include some form of HCI.
Deversity of parts: 0 - 10	7	Just one or two parts, little integration needed.		Multiple parts, needing to be integrated
Enclosure: 0 - 10	9	Square, colorless, simple, unstylish.		Stylish, elegant selection/choices for colors, eyecatching.
Cost-saving ideas: 0 - 10	8	Little or no concern for costs or budgeting. Possible wasteful expenditures.		Cleaver tradeoffs, cost-saving strategies and budgeting plan.
Drawings & general clarity of concept description: 0 - 10	10	Little or no use of drawings or documents during meeting.		Illustrated ideas during meeting using sketches / 3D models, etc.
TOTAL / 100:	**83**			

University of Cape Town

B.4.2 Evaluation of design quality

DESIGN SCOREBOARD



TEAM: **3 (Vibynet)**

Each criterion is given a rating value from 0 to a maximum value. Fill in a number within this range to indicate how well the team has done.

Criteria Descirption	Rating	Boring = 0	MAX/2 = Middle of the road	MAX = Inspired!
Introduction / readme doc: Rating levels: 0-5	5	Not provided. Difficult to understand.		Good outline / starting point
Block diagram (overview of HW and SW componts & important connections). Levels 0-10	9	Not provided / unclear.		Good neat. Diagram. Concise.
Use case design: 0-5	5	Not provided. Unclear. Illogical.		Simple, clear. One diagram per use case. Correct use of notation.
Scenarios / sequence models. Levels: 0-5	4	Not provided / unclear		Sufficient detail, logical sequencing and labelling of steps.
UML class & object design and diagrams: 0-10	9	None given. Incorrect / non-standard notation. Messy.		Neat. Correct notation. Suitable level of detail used.
Drafs schematics / curcuit. Rating levels: 0-10	7	None given. Totally incorect / wrong one given / unreadable.		Neatly produced and labelled. Accompanying text to indicate important aspects / decisions.
Component list: 0-5	4	None. Vague. Incomplete. Little useful information given. No clear linkages to other aspects of the design.		Listing includes useful info such as part numbers, URLs to manuals. Relates back to block diagram / cicruit / UML models.
Enclosure model / specifications: 0-5	5	None. Minimal effort. Untidy. No labels. Missing aspects (e.g., design mentions buttons, but enclosure hasn't been design for any).		Neat enclosure design. Elements of design linked (e.g. via labells or accompanying comments) to design aspects.
No unwanted redundancy: 0-5	5	Duplicated design elements / excessive duplication (e.g., separate on and off buttons whereas single button reduce size and costs). Too many (mainly) useless features.		Developers have been meticulous in ensing aspects of the design are not duplicated or unnecessary.
Accuracy of design (i.e., no obvious mistakes with software needs, capacitance / resistance / timing calculations): 0-10	8	Mistakes / inconsistancies / glaringly wrong calculations		Designs thoroughly checked & tested. Suitable detail. Calculations clear and appear to be correct (during the time-limited review).
Completeness: 0-5	5	Important design details lacking		Design appears comprehensive
General: clarify and understanding: 0-10	10	Generally unclear / untidy / incomprehensible in many parts		Generally tidy, readable, well laid out.
General: internal consistancy. Levels: 0-5	5	The text of individual design documents/models are full of inconsistancies		Text and diagrams of individual documents agree / are logical.
General: external consistancy. Levels: 0-5	5	Little cosistancy between documents/models: one indicates one choice, different to other docs.		Good consistancy in different documents/modules. Same design decisions evident in all docs.
General: Cross-references / tracability / captions and annotations. Levels: 0-5	5	No cross-refs between designs, minimal labels, not tracable back to concept requirements.		Use of cross-refs between design aspects. Good captions, labels.
TOTAL / 100:	**91**			

University of Cape Town

B.4.3 Evaluation of artefact quality

ARTEFACT SCOREBOARD



TEAM: **3 (Vibynet)**

Each criterion is given a rating value from 0 to a maximum value. Fill in a number within this range to indicate how well the team has done.

Criteria Descirption	Rating	Confusing = 0	MAX/2 = Middle of the road	MAX = Excellent!
Naming of artefacts. Levels: 0-10	6	Meanigless / vague names.		Good, descriptive names.
Structure / organization of digital artefacts: 0-10	8	All lumped together, difficult to identify what particular files relate to.		Well organised. Easy to determine what particular files contain / are used for.
Tracability: 0-10	7	Difficult to determine what design feature or requirement the artefact relates to.		Easy to tell what design features / requirements the artefact provides or is related to.
Modifiability: 0-10	10	Difficult to tell how some of the files can be modified / adapted. (e.g., need some kind of obscure software programs)		Generally obvious (to a computer engineer) what tool to use; special formats and tools explained.
Reusability: 0-5 (somewhat related to modifiability)	4	Few if any of the artefacts are worth reusing.		Many artefacts could be reused in future projects / upgrades.
General Readability / clarity of content: 0-10	7	Difficult to make sense of artefacts. E.g. messy, illegible documents.		Neat easy-to-read documents, good diagrams, annotations.
Readability of code: 0-5	3	Poor/no comments		Suitable amount of meanigful comments.
Diagrams / models / AOD relating artefacts: 0-5	5	No attempt to model the layout of artefacts, how they relate.		Explained the organization and reason for important artefacts.
Accessibility / ease of location: 0 - 10	9	Difficult to find important information / files/ documents.		Easy to locate files / information.
Owner/author, change history. Levels: 0-5	3	No author/owner indification. No change history/log of fixes.		Each document contains author/ owner information, history of indicating dates of important changes/ fixes applied.
Non-redundancy: 0-5	4	Duplicated documents. Multiple versions scattered haphazardly. Difficult to determine correct version. Commented out code.		Little redundancy (only used where it is beneficial to do so). Backup / old versions of artefacts moved into separate folders / compressed archives to avoid confusion.
Cross-referencing: 0-5	4	No cross-refs used, likely to waste time trying to keep track of where related artefacts are.		Effective use of cross-refs (e.g. mention of paths, URLs, filenames, etc. making it easy to find / know of related artefacts).
Use of ESAOA tools / other artefact management and classification tools: 0-10	8	Poor used of IT tools / poor organisational structures / no use of scripts to automate procedures		Good organisation and documentation of artefacts. Application of IT tools to facilitate management of the artefacts. Use of tools like fclass to classify and relate artefacts.
TOTAL / 100:	**78**			

University of Cape Town

B.5 Design review 2 questions regarding knowledge production

In design review 2, teams were asked to comment on their knowledge production methods and information sources used in their project. The same set of questions was posed to each team; the question sheet is shown below. The researcher took a printout of this sheet to read out during design review meetings, and wrote point-form notes in the space provided to record responses.

Design Review #2 Questions to prompt comment

TEAM: VibyNet (Proj #3) DATE: 30-Sep-05

#	Question / contextulization	Notes / responses
Q1	(This question is intended to gain comments concerning data knowledge production and information sources) Please comment on ways in which you **gained facts** and general information that you needed during development tasks. Also, what **tools/documents** were used to answer these issues? For example, how did you answer ***'what'*** questions such as: (a) What software or hardware components (e.g., operating system,tools, devices, etc.) do I need to get? (b) What kind of features or requirements should we add to the prototype (NB:, what you were thinking about earlier, before you started solving specific implementation needs)? (c) What baud rate do I use for this device? What pins do I need to connect up?	• Main component RF transceiver • Try to find unlikely board • What is the fastest cheapest option for RF? • Answered by websites • Looking on sparkfun • Look at big companies, eg ATMEL and microchip. • Used forum to see what other designers used. • Looked easy getting, eg SPI or rather serial UART. • Searching in PDFs downloaded • Use of data sheets provided by ATMEL, eg ATMEGA128
Q2	(This question is intended to gain comments concerning process knowledge production) Please comment on ways in which you learned about and decided on **development methods** needed to carry out implemenetation, and how you learned to **use development tools**. These answer more ***'how'*** questions. For example, how did you answer questions such as: (a) How can I change the baud rate for the serial port? (b) How can I send characters out the serial port? (c) How do I compile this code for the embedded system and make the executable run on the platform?	• Mostly changing code, esp. the main.c code file and the PIV.h in AVR dir. • Some changes to gcc Makefile, had to change the configure make file. • Used forums to find examples (third place question) • Setting up bit masks in code to make config/pins. • Often used the scope. Also used memory tester, used the myAVRprobe or test mode to read unit's.
Q3	(This question is intended to gain comments about innovation knowledge production) Please comment on the more innovative aspects of your project, discuss any situations you may remember that occurred *during* implementation tasks, where you though of new requirements, design insights, and ways to test idea/product requestions. These answer more the ***'what should I try'*** and ***'what works'*** type of questions you may have had in the project. For example, how did you answer questions such as: (a) How am I going to implement this requirement? (b) What else could I try to get this to work? As an example, consider one of your requirements was to sent a welcome message out the serial port to confirm that your program is running. Clearly, to do so, you needed to determine how to use the serial port (this is a Q2 question). Once having done so, you might have decided something like "Ok, I'll send the message 'hello world' out the port, and do this just after the serial port has been configured" (this is a Q3 question).	• wrote notes in dev. log file. for steps used). Usually didn't write the steps in. • Some ideas come up when looking at web for solving other problems. • Mostly Q3 led to/caused Q1/Q2 problems. • Eg. thought of cool LED flash sequence. Happened when listening to music. • Our tools were specified, we only used what we were given + extra. • Hard to test the various requirements. • Sometimes did web searching (eg. for things to try).

University of Cape Town

Appendix C: ESAOA KMS version 2 appendices

C.1 Knowledge ontology for ESAOA KMS version 2

The top-level terms of the knowledge ontology are not indented; terms that are specializations of these top-level terms are intended depending how deep they are in the hierarchy of terms. For terms that are specializations of more than one higher level term, the other higher levels terms are indicated using curled parentheses. Terms added for ESAOA KMS version 2 are indicated using an asterisk (*).

- ☐ Activity: one or more related actions carried out by a role (or multiple roles) that relates to the development of an embedded system.
- ☐ Artefacts: a physical resource (e.g., books and equipment) or digital resource (e.g., software tools, documents and other files) used in activities.
  - Soft artefact OR digital artefact: a file stored on computer (e.g., a code file or PDF datasheet).
    - Data artefact: an artefact that captures data knowledge. These artefacts act as a source for process knowledge. Examples include datasheets, data files, sample source code and manuals.
    - Process artefact: an artefact that captures process knowledge, such as a how to manual, a baseline code module, or a log of commands used to configure a tool.
  - Hard artefact OR physical artefact: a physical artefact that is not stored in a file system (e.g., hardware components, printouts, and embedded system products that developers are working on).
  - Innovation artefact: an artefact that captures innovation knowledge. Modified baseline code modules and concept drawings are examples of code artefacts.
- ☐ Role: describes the behaviour, responsibilities, characteristics and needs that a developer takes on during certain development activities.
- ☐ Space: a location (digital or physical) where artefacts reside.
  - Communal space: space shared between multiple teams (e.g., laboratory).
  - Team space: space shared between members of a team.
  - Individual space: space used mainly by one member of a team.
  - Soft space OR digital space: space comprising only soft artefacts (e.g., website, file directory stored on a hard drive)
    - Workspace OR ESAOA workspace: a soft space where development activities that involve changes to soft artefacts take place, comprising software tools and other artefacts used to carry out development activities.
      - Communal workspace: a digital space where members from different teams share and interact (e.g., a wiki site or shared network folder).
      - Team workspace: a workspace that is shared by members of the same team. Stores the master version of a team's project repositories.

University of Cape Town

- Personal workspace: a workspace that is predominantly used by an individual team member. The personal workstation is usually a working copy of the team workspace.
- Master workspace: workspace containing master versions of a team's soft artefacts (generally the same as the team workspace).
- Workspace copy: a copy of the master workspace stored for example on the hard driver of the local machine (i.e., a backup).
- Hotspot OR knowledge hotspot: a demarked section of a soft artefact, within a particular workspace, that is of particular importance to formulating or sharing a certain type of knowledge.

◆ Hard space OR physical space: place where hard artefacts are located.
  - o Workstation computer: refers specifically to the computer of a workspace (i.e, the PC used by an engineer).

◆ Workstation OR ESAOA workstation: a computer system that provides the human/computer interface to a workspace, together with the surrounding physical artefacts used during development (e.g., printouts of datasheets, testing equipment and the embedded system being worked on).
  - o Communal workstation: a workstation accessible to multiple teams.
  - o Team workstation: a workstation accessible to all members of a particular team.
  - o Individual workstation: a workstation used predominantly by a specific team member.

☐ Classifications (*): ways in which an artefact is classified.

◆ Form classification (*): indication of whether an artefact is a soft or hard artefact. Also declares an artefact as represents explicit knowledge, or being a boundary artefact, or being a knowledge artefact, or a combination of these.

◆ Functionality classification (*): based on what the artefact provides or is used for (e.g., used for documentation, or is a code file). The functionality classification helps to identify what an artefact is, or what it provides in a project or workspace.

◆ Workspace classification (*): indicates where the master version of an artefact should resides, or from which workspace the initial version of the artefact was obtained. The URL or other reference information about the original version of the artefact can be added to this classification.

◆ Maintainer classification (*): indicates the role responsible for maintaining the artefact (this classification is partly implied by the workspace classification; for example, artefacts in the communal workspace are maintained by the CKS).

☐ Episode (*): an activity, or set of related and chronologically close activities, within a larger sequence of activities leading to a final result.

◆ Progression to innovation episode (*): a sequence of knowledge production tasks related to a common idea, which leads from an initial acquisition of data knowledge related to the idea, through zero or more instances of process knowledge creation, resulting in a final occurrence of innovation knowledge in which the idea is tested and possibly new ideas arise.

◆ Productive episode OR productive progression to innovation episode (*): a progression to innovation episode in which only productive knowledge is obtained.

University of Cape Town

- Non-productive episode OR non-productive progression to innovation episode (*): a progression to innovation episode in which both productive and non-productive knowledge is obtained.

Note: Only a subset of the terms listed above were tested using the prototype of the ESAOA ontology manager (OM) tool discussed in Chapter 6; the terms and their descriptions were stored in a CSV file which the OM tool read and wrote to.

C.2 Details concerning the ESAOA modelling language

This section provides supplementary connection details concerning use of the ESAOA modelling language.

C.2.1 Further detail on connectors

A "knowledge capture" flow links a knowledge atom to one or more artefact atoms, and is used to show artefacts used to make aspects of knowledge explicit. An "artefact adaptation" flow is modelled as an arrow that points from a role (or process) atom to an artefact atom, indicating which role (or process) is responsible for creating, adapting or maintaining that artefact. An "artefact use" association is shown as an arrow that points from an artefact to a role or process atom; it shows which role or process makes make use of the artefact. A "role support" association is in the form of a line that has a solid circle on one end – the end with the circle indicates which role is provided support form the role on the other end. Figure C.1 provides examples showing the use of these connections.

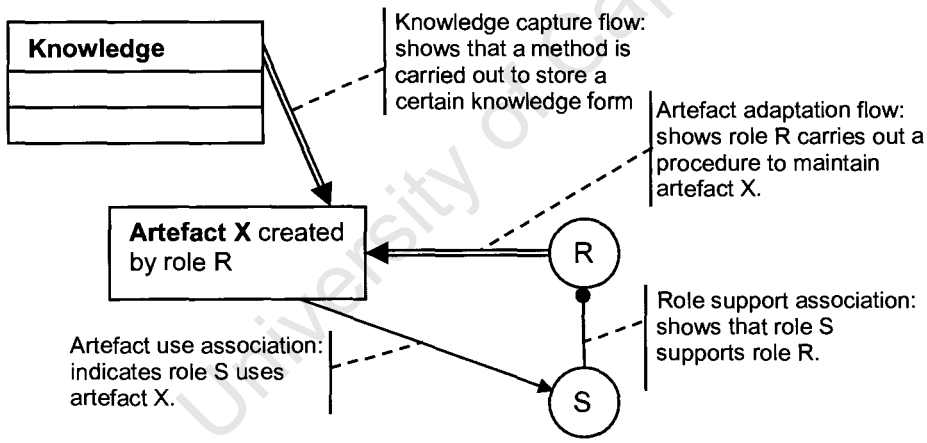


Figure C.1: Knowledge store, artefact maintenance, role support and artefacts use connectors.

The "knowledge use" association joins a knowledge atom to a process atom, indicating a knowledge form that the process depends on. The "role interaction" association indicates an interaction or some form of relation between two roles (these connectors are labelled to identify the relation involved). If the role interaction connection has an arrowhead, the arrow points to the role that is provided something by the other role (the role interaction connector junction is used to specify what is being provided – see Section 6.3.3). The "role perform" association connects a role atom to a process atom, showing a process that the role performs. The "process capture" flow links a process atom to a soft artefact, identifying a soft artefact used to document the process. The "process maintenance" flow shows a role that maintains

University of Cape Town

a process. An "artefact conversion" flow shows an artefact formulated into a different type. These connectors are illustrated in Figure C.2.

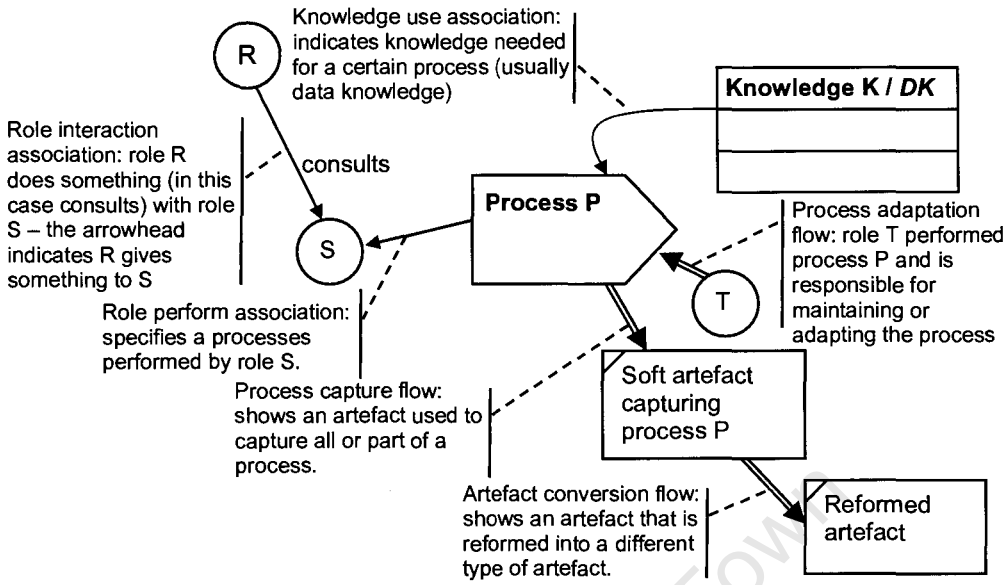


Figure C.2: Knowledge use, role interaction, process capture, process adaptation, role perform, and artefact conversion connectors.

The containment association, modelled as a line ending in a solid diamond shape, is used to indicate a type of atom (usually an artefact or process) contained within another atom; the container atom is on the end with the diamond shape. The containment association can connect two artefact atoms (indicating artefacts stored inside another artefact), or can link two process atoms (showing a hierarchical breakdown of a high-level process into sub-processes). A dependency association is represented as a dashed line ending in an arrowhead (the atom on the end without the arrowhead depends on the atom on the end with the arrowhead). Figure C.3 provides an example model using these connectors.

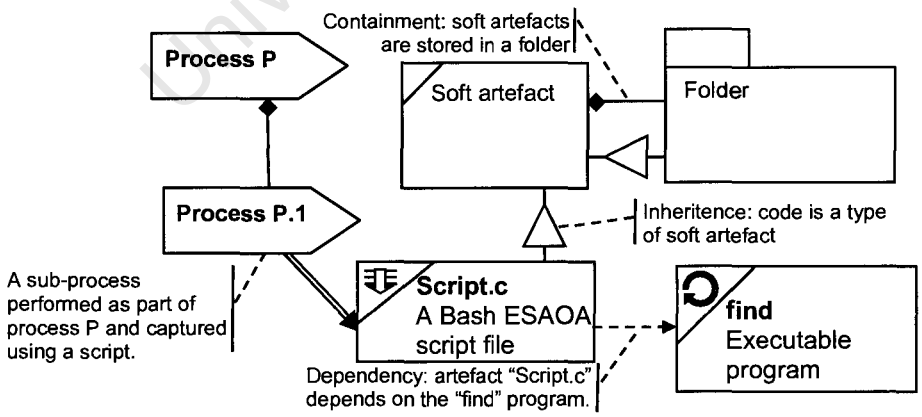


Figure C.3: Containment, dependency and inheritance associations.

University of Cape Town

C.3 Design details related to ESAOA support tools

This section provides supplementary design details concerning ESAOA support tools that are included in version 2 of the ESAOA KMS.

C.3.1 Design issues of the Personal Expert Program (PEP)

The Personal Expert Program (PEP) was added to version 2 of the ESAOA KMS as a means to access and manipulate file classifications more efficiently. The PEP application starts by loading the .fci file corresponding to the ESAOA workspace that is loaded when the user enters the ESAOA environment using the *enter-aoa* script. The PEP program then remains running as a background task, waiting for IPC messages to be sent to it from the fclass program.

The PEP program can be started in stand-along mode, in which the user can use a menu to interface with the program. Alternatively, command line arguments can be specified to invoke one PEP command (after which the application exists; for example the command 'PEP Count' will return the number of entries in the .fci file of the currently active workspace). The IPC messages, and corresponding commands, that can be used with PEP are listed below:

Command ID	Menu Item / command line arguments	Description
90	Select *path*	Change the active workspace file (uses the ESAOA_TEAM, ESAOA_ROOT and ESAOA_COMMUNAL environment variables) to determine the .fci file to make active. If no path is specified, the personal workspace is selected.
91	SelectP	Select the personal workspace.
92	SelectT	Select the team workspace.
93	SelectC	Select the communal workspace.
100	Add *code description*	Add a new classification and its description
101	Inherit *code_sub code_super*	Instruct PEP to record that the classification *code_sub* inherits the existing classification *code_super*.
102	Uninherit *code_sub code_super*	Remove any entry recording that classification *code_sub* inherits classification *code_super*.
103	Suggest_desc *description*	Request PEP to suggest a classification code value for the given description.
104	Ask_desc *code*	Request the description of a classification with given code.
105	Ask_inh *code*	List all inherited classifications for the specified classification with given code.
110	Count	Return the number of classification entries in the .fci file.
111	Get *n*	Return the classification code of the nth classification entry in the *.fci* file.
112	Deln *n*	Remove nth classification from .fci file.
120	Copyto *code destination_file*	Copy a classification from the current .fci file (loaded in memory) to a .fci file in a different workspace (the destination path will be compared to the ESAOA_TEAM and ESAOA_COMMUNAL environment

University of Cape Town

		variables to determine the appropriate .fci file to change.
121	Moveto *code destination_path*	Move a classification from the current .fci file (loaded in memory) to a .fci file in a different workspace (using environment variables as above).
122	Remove *code destination_path*	Remove a classification from the .fci file corresponding to the path indicated.
123	LRemove *code*	Remove a classification from the .fci file of the current workspace.

C.3.2 Detailed design issues related to *fclass* and related CSV files

The fclass program is used to access and manipulate file classification information related to files in an ESAOA workspace. The second version of fclass also supports functions for recording source information and URL references related to particular files in an ESAOA workspace. The table below lists command line parameters of the fclass program.

Syntax	**Description**
fclass <files> + <class>	Apply classification <class> to the files indicated.
fclass <files> class <class>	*Same as above*
fclass <files> - <class>	Remove indicated classification from all files listed that have that classification applied.
fclass <files> rmclass <class>	*Same as above*
fclass <files> -	Remove all classifications from files
fclass <files> rmclass	*Same as above*
fclass <files> class? <words>	Return all the classifications (one per line) that have been applied to the files in <files>. If <words> is empty, returns all classifications found, otherwise returns only results that contain all the words listed in <words>.
fclass <path> qclass <class>	Return the names of all files in directory <path> that have the classification <class> applied. If <class> is an empty string, all files that have a classification are returned.
fclass <path> -f <class>	*Same as above – for compatibility with version 1*
fclass <path> find <class>	*Same as above – for compatibility with version 1*
fclass <files> desc <desc>	Apply description <desc> to all files indicated. If <desc> is blank, existing description is removed.
fclass <files> desc? <words>	Returns description for all files in <files>. If <words> is empty, returns all descriptions found, otherwise returns only descriptions that contain all the words listed in <words>.
fclass <files> qdesc <words>	Returns all files in directory <path> that are assigned a description that contains all the words listed in <words>. If <words> is empty, then all the names of files that have a description linked to them are returned.
fclass <path/file> -l	List all classifications applied to the files or paths specified. Each line returned starts with the filename followed by the description applied. This provides compatibility with version 1 of fclass.
fclass <path/file> list	*Same as above*

University of Cape Town

fclass <file> count	Counts number of classifications applied to a file.
fclass def <class> <desc>	Define a new class and give it a description
fclass def <class>	Delete the description for a class
fclass qdef <class>	Display the description for a class
fclass isa <class1> + <class2>	Define a new inheritance (class1 inherits class 2)
fclass isa <class1> - <class2>	Disinherit class2 from class1 (i.e., class1 no longer inherits class2)
fclass qisa <class>	Display super-classes of class
fclass nisa <class>	Count number of classes that <class> inherits
fclass <file> url <url>	Link a URL to a file (note that of <url> is blank, then the URL is deleted). This command manipulates the .fos file in the directory that <file> is located in.
fclass <path> qurl <words>	Display URLs linked files in <path> in which all the words in <words> are found in the URL text. If <words> is empty, then all the URLs are returned.
fclass <file> src <path>	Indicate from which workspace or path a file originates. The <path> string can be assigned to "common" to indicate the communal workspace. Alternatively, set <path> to a team number (such as "team1" to record the team name). Whatever <path> is set as, is stored into the .fos file directly (i.e., fclass does not checks to see if the path actually is valid). If <path> is not given, then the src record for the file in the corresponding '.fos' file is deleted.
fclass <path> qsrc <words>	Search .fos files in directory <path> that has src entries that match all the words in string <words>. If <words> is blank, then all the src entries for files in <path> are returned.
fclass –h ...	Sets fclass to use generate html output for classification or description queries – the result is saved in a file called 'out.html' that resides in the root of the user's personal workspace. The –h option must be followed by a query for results to be produced.
fclass –hd ...	Similar to the –h option, except the html output of the query is displayed to the console.

University of Canterbury

C.3.3 Hotspot logging (hsl) tool

The *hsl* tool was introduced in Section 6.4.2. In summary, the *hsl* program works in a similar way to *fclass*, maintaining file meta-data, in terms of *hotspots*, in a CSV file. As defined in the ESAOA knowledge ontology, a *hotspot* (or *knowledge hotspot*) demarcates a section of a soft artefact that resides in a particular workspace and is of importance to the formulation or sharing of knowledge related to the artefact concerned. This section provides further details on the *hsl* tool.

C.3.4 List of ESAOA support tools

The collection of ESAOA support tools available in version 2 of the ESAOA KMS are listed below.

Command and syntax	Short name	Description
ESAOA SCRIPTS		
acro		Display a PDF file (using preferred PDF reader).
aod		Display AOD for current project (uses *acro* script).
build-esaoa		Compile and link any ESAOA command implementations in the current directory.
check-comments.sh		Indicate which c and h files for the current project have or have not been commented (intended for used by the Workspace Administrator).
esaoa-clean	clean	Remove make and object files.
cls		Calls the Linux clear command to clear the screen.
dayplans.tcl *[date]*		TCL script to display a dayplan (i.e., list actions that the team or a developer recorded to be performed on a certain day – defaults to current day)
del-csv.sh *file*		Delete a CSV, but check that it is not needed by any files in the directory. This script is used in cleaning up an ESOAO workspace.
del-fcs.sh *file*		Delete a FCS file, but only if it does not refer to any files that actually exist (used in clearing up an ESAOA workspace).
desc *file description*		Add a description to a file, stored in a CSV metafile.
dus		The Disk User Summary (DUS) command presents a summary of disk space used by active workspace.
enter-esaoa (or enter-aoa)		Enters the ESAOA framework environment.
esaoa-access		Script to assist setting file access rights in an ESAOA workspaces. Can be configured to keep track of which users should have access to a particular project.
esaoa-addpath *path*		Add *path* to the PATH environment variable.
esaoa-artifact *file*		Add meta-data to artefact (stored in CSV file in local directory).
esaoa-bm *name [path]*	b	Add a bookmark (if two parameters are given) or change to directory associated to a bookmark name (if one parameter is given). Bookmarks are stored in a CSV file in the personal workspace root directory.
esaoa-bmark *name*		Change directory to a bookmarked directory (same as esaoa-bm but gives more output).
esaoa-burn *[file]*	burn	Script to write a compiled program to the boot sector of flash memory on the CSB337 (this script can be overridden for different plafroms).
esaoa-cd *path*	cd	Change directory (makes use of the standard Linux cd, but allows for recording directory changes are used most frequently, allowing for possible future automatic suggestions and bookmark additions).
esaoa-checkdir		This script is called by esaoa-cd and esaoa-goto to check if the user has changed to the directory of a different project, or to a different subproject, and updates the environment variables as needed.
esaoa-checkin		Called from a personal workspace. Overwrites files and their metadata in the team workspace that have been changed in the personal workspace.

University of Cape Town

esaoa-checkout		Called from a personal workspace. Overwrites files and their metadata in the personal workspace that have been changed in the team workspace.
esaoa-checks		Perform checks to verify that the ESAOA environment is operating and installed correctly. Also can test the ESAOA tools.
esaoa-clean		Removes all generated files in application directory.
esaoa-cleanup		This method saves the program (if any) and then deletes all the intermediate object files and other generated files used to compile the program.
esaoa-cp *files dest*	cp	This script copies a file or a set of files to a given destination. The alias cp is configured when the ESAOA environment is entered to override the standard Linux cp program.
esaoa-create-c *name*		Creates a C file called *name* (by copying file template.c from the Templates directory in the active workspace).
esaoa-create-cpp *name*		Creates a C++ file (as above).
esaoa-create-script *name*		Creates a copy of the ESAOA Script template (by copying template.sh from the Template directory).
esaoa-dt	dt	Returns the current date and time in format DD-MM-YYYY HH:MM:SS.
esaoa-edit *file*	e	Edit the specified file using preferred editing tool (set using environment variable ESAOA_EDITOR); can be overridden using another esaoa-script in current workspace's Scripts directory.
esaoa-exeapp *filename*	x	Execute the exe file in current directory – if using PCBox executes on PC, otherwise instructs embedded platform to run the program.
esaoa-exit	ex	Exit the ESAOA framework environment.
esaoa-extract		Script to extract an archive file based on its extension name (uses tar or unzip).
esaoa-find *name*	f	Find an artefact based on keywords. Can search based on filename or on text within files.
esaoa-flash-free		Show free space on preferred USB flash drive; used to check space for backups (the file $HOME/mounted-flash.inf indicates which directory corresponds to the preferred USB flash drive).
esaoa-flash-list *[path]*		List files on preferred USB flash drive.
esaoa-flash-mount		Mount preferred USB flash drive.
esaoa-gf *keywords*		Script to find a certain artifact and change directory its parent folder. Keywords are in the description metadata of the artefact concerned.
esaoa-help *[name]*		Display summary of frequently used ESAOA support tools. If parameter given, shows help for ESAOA program/script corresponding to that name.
esaoa-home		Change to root of current project if current directory is in a valid ESAOA workspace, otherwise changes directory to root of personal workspace.
esaoa-info		See framework-info (provides version information).
esaoa-l *[path]*	l	Display file listing for current directory, including summary of classifications applied to files.
esaoa-m	m	Compile application, based on current directory.

University of Adelaide Library

esaoa-mkrc		Create an ESAOA RC file (a Bash script file executed when the user changes into the directory containing the RC file).
esaoa-mv *source destination*	mv	Move or renames a source file to a description, and moves related metadata along with the file. This script overrides of the standard Linux mv command when the ESAOA environment is entered.
esaoa-nop		Does nothing.
esaoa-platform *name*		Alias for platform command (see platform below).
esaoa-project		Script to create a new ESAOA project (note that a template project must exist in directory $ESAOA_INSTALL/Templates). Can also be used to manage current project metadata.
esaoa-request-ids		This script is obsolete. Reserve an ID range for this user. This script should only be executed on the central server, which keeps track of which user owns which Ids.
esaoa-rm	rm	Deletes a file together with its associated metadata. Overrides the standard Linux rm command.
esaoa-scratch		Manage temporary file: creates and automatically disposes (on esaoa-ex) of temporary files for an ESAOA workspace.
esaoa-snap	snap	Take zip archive snapshot of files in current directory and below, excluding object and exe files.
esaoa-term	term	Open favourite terminal (default: xterm).
esaoa-test		This script tests a C/C++ module if it includes a body of code that contains a main() function that is only included if ESAOA_TEST is defined.
esaoa-to-flash *file*		Copy a file to the preferred USB flash drive.
esaoa-tz	tz	Tar and gzip current directory and subfolders.
exit-esaoa		Same as esaoa-exit (exits ESAOA environment).
fileswap *file1 file2*		Swap the names of two files (i.e., changes file1 to file2 and the original file2 has its name set to file1).
framework-info		Provide information about the ESAOA framework (e.g., version number, installation paths, etc). Includes info about active ESAOA communal workspace and ESAOA team workspace.
gendoc *[file]*		Script to generate documentation for a module using Doxygen (and provided that Doxygen-compatible comments are used in the module).
godir *string*	go	Calls the esaoa-godir program. Changes directory to subdirectory in current folder that most closely matches the characters in *string*.
gui-kmtools.tcl		Prototyped GUI menu for accessing ESAOA tools without having a type them at the command line.
gui-menu.tcl		Prototyped GUI menu for accessing commonly used tools without having to write them at the command line.
home	h	Executes command "goto \$H" to change current directory to the active framework root.
isa *file class*		Uses fclass to associate an artefact classification to a file.
keyw *file keywords*		Associate keywords to a file.
kw		Loads the Kwrite text editor.

University of Adelaide Thesis

platform *name*		Display or change current target platform.
platform-info		Display information for current target platform.
rm-objects		Removes all compiler-generated object files from the current directory (used by esaoa-clean).
start-pep		Start the Personal Expert Program (PEP) background process (called by enter-esaoa).
title *file name*		Associate a title or longer and more descriptive name to a file or directory.
tree *path*		Display directory tree for path given.
ESAOA PROGRAMS		
ascii-table		Display table of ASCII codes and characters
esaoa-apps		Display list of applications in applications directories for current workspace. Also used to determine the directory for applications for the *go apps* script.
esaoa-arg		Program to assist argument processing.
esaoa-askpass		Ask user for password (echoes '*' characters for characters entered).
esaoa-bar *num char*		Display a status bar (num indicates number of characters *char* to print out)
esaoa-capture		Redirects input from RS232 serial port to a file.
esaoa-cwd *[path]*		Prints current working directory from given root *path* directory(defaults to '/' if *path* is not given).
esaoa-defs *[options...] file*		List or add a definition to project defs.csv file.
esaoa-download		Downloads bytes from a RS232 serial port or IP socket connection.
esaoa-err	err	Add a line of text or file to the ESAOA error log file for indicating user concerned. Can also be used to redirect error stream (depending on environment settings specifying where errors should be sent).
esaoa-fm	fm	Filename manipulation application. This program is used to access parts of a filename, such as removing the path from a file.
esaoa-godir *keywords*		Based on current directory and keywords, the program decides which directory the usually probably wants to change to (this program is used in conjunction with the godir script which performs the actual change of directory).
esaoa-kdb *operations*		This program implements a console interface to the ESAOA database comprising metafiles within an ESAOA workspace.
esaoa-log *[time] text*	log	Program to add an entry to a text log file, display a log for a certain day, or to construct a day plan.
esaoa-menu		Used in displaying a menu to the console (used by scripts such as the ESAOA install script).
esaoa-mm *[file]*	mm	Make *.make files for application. Run in the relevant application directory.
esaoa-moddep *file*		Determine dependencies for a C or C++ code module.
esaoa-neaten *file*		Neaten the comments and code indentation in a C file.
esaoa-pathtool		Add a directory to the path in the team workspace. Also provides functions to determine which paths are already in the environment PATH variable, and

University of Pune - Sivn

		changing the order of paths in the PATH variable. Only implemented for Linux/Cygwin; version 2.0 does not handle Windows paths.
esaoa-platform		Access information regarding a particular platform, or compute environment variables that need to be changed to reconfigure compiling for a different platform.
esaoa-ppc *file*		C / C++ file pre-processor. Use in C or C++ pre-processing (e.g., finding comments, author names, etc). By default, inputs a C file and outputs to the console a version of the file with all comments removed and sections of code that would be eliminated by #ifdef constructs.
esaoa-projroot		Return the project root directory for the project corresponding to the current directory.
esaoa-prompt		Compute the Linux prompt to display. Called by Bash when in the ESAOA environment to display a command prompt.
esaoa-readln		Read a line of text from console entered by user.
esaoa-setup		Program to display and manipulate ESAOA setup information stored in shared memory and copied to the repository .esaoa file. Generally used to customise the ESAOA environment.
esaoa-shm		ESAOA SHared Memory data access program. This console application provides an interface to access shared memory segments manually (e.g., copy a file from disk into a shared memory segment, or copy a shared memory segment to a file on disk).
esaoa-sm	sm	Send / log a message to fellow team member.
esaoa-status		Add status entry or annotate existing status entry.
esaoa-stubgen		Generate stub functions.
esaoa-synch		Synchronise two directories (generates a .sync file in each directory used to determine which files have been added or deleted).
esaoa-tally *tally_file [val name] [d]*		Program to create and manipulate a table where each row contains a count value and a name.
esaoa-tdb *file operations*		Text file database. Program that manages rows of data (strings or floats) in a text file.
esaoa-termcnf		Generate terminal configuration script (used when ESAOA environment is entered).
esaoa-upload		Upload a datafile over RS232 serial port or IP socket connection. Expected to be used mostly to send compiled executables to the target platform.
esaoa-version	ver	Display current version of ESAOA communal workspace.
esaoa-xfer		This program uses the esaoa-sync application to import or export files from a particular personal workspace to a team workspace.
esaoa-zip		Zip archive manager (used to manage archives stored in Project directory). (This program is meant to track zip files created, and can access certain files within a zip file, but is not meant as a replacement for the standard zip/unzip programs used to do the actually file de/compression.

fclass		ESAOA file classification program. Program that is accessed by the user form the console to change file classifications stored in csv files.
fcs		ESAOA file classification system data manipulation program. Manipulates CSV files, and implements many of the functions needed by fclass.
fim		Functionality classification index file manager.
istime		Check if a certain date/time has been reached.
penv		Modify/displays ESAOA environment variable.
pep		Personal Expert Program (PEP). This program is started as a service to maintain the .fci files used by fclass that contains the classification indexes for the active workspace.
sdb		Simple database application (SDB). Console-based prototyped program that uses a subset of SQL commands to create, update and manipulate CSV files.

C.4 Kit for Information Technology (KIT)

The Kit for Information Technology (KIT) is a C++ application programming that was developed during the first iteration of Framework Construction. KIT provides high-level functions for navigating ESAOA directory structures and for manipulating CSV files that contain meta-data for files. The KIT API consists of a set of C++ H files that incorporate the functional interface and a library file that is linked to the user application being compiled. The KIT C++ modules can be incorporated directly into an application without linking the precompiled library file (this was done to make it easier to port individual programs or reuse code from KIT for other purposes, such as inclusion into embedded software). The KIT API is included in the ESAOA communal distribution.

The UML class model for KIT is given in Section C.4.1 and is used in outlining the various modules and classes available in the library. Section C.4.2 provides a flowchart and code snippets showing the design and implementation of a C++ ESAOA program using KIT.

University of Cape Town

C.4.1 KIT modules and UML class model

This section lists the Kit classes and modules of the KIT API and provides a UML class diagram to illustrate the relation between the classes in the KIT API.

List of KIT classes and module files

Class name(s)	Module files	Description
	KitTypes.h	Includes all the commonly used Kit datatypes (such as KitObject and KitString).
	XMLCommon.h	Routines that are used by both KitXMLOut and KitXMLIn.
Kit, KitCli	Kit.cpp Kit.h	Included by all applications that uses Kit. This module implements top-level classes.
KitActionExpert	KitActionExpert.cpp KitActionExpert.h	Record an action and add to list of actions (used if command

		capture is enabled in the ESAOA environment).
KitAssoc	KitAssoc.cpp KitAssoc.h	Abstract class for a top-level association class (a class used to relate two other classes).
KitAtom	KitAtom.cpp KitAtom.h	A type of class that is used in a KitDisplay and can display itself.
KitBaseRepository	KitBaseRepository.cpp KitBaseRepository.h	Abstract / top-level repository class (has no methods).
KitContainer	KitContainer.cpp KitContainer.h	A KitObject that can contain zero or more KitObjects. Similar to a file folder.
KitDisplay	KitDisplay.cpp KitDisplay.h	A display class. Displays KitObjects to the console or a GUI window (depending on subclass used).
KitExternalRepository	KitExternalRepository.cpp KitExternalRepository.h	Corresponds to a folder that is outside an ESOA workspace.
KitFile	KitFile.cpp KitFile.h	Corresponds to a file on disk.
KitFileRepository	KitFileRepository.cpp KitFileRepository.h	Corresponds to an ESAOA workspace.
KitFolder	KitFolder.cpp KitFolder.h	Corresponds to a folder on disk.
KitGUI	KitGUI.cpp KitGUI.h	A wrapper class for GUI window display functions. Subclasses of this class implement the actual calls to underlying libraries.
KitGUICreate	KitGUICreate.h KitGUICreate.cpp	A wrapper class that handles creation of GUI windows. Subclasses implement calls to underlying GUI libraries.
KitImageAtom	KitImageAtom.cpp KitImageAtom.h	A KitAtom with an image/bitmap representation.
KitImageResource	KitImageResource.cpp KitImageResource.h	An image resource (stored within the Common directory of the active workspace).
KitInternalRepository	KitInternalRepository.cpp KitInternalRepository.h	A repository that is hidden from the user (stored within Comon directory of active workspace).
KitKB	KitKB.cpp KitKB.h	Wrapper class for access to a SQL database engine. Subclasses implement library calls to access the database.
KitList	KitList.cpp KitList.h	A list of KitObjects.
KitLogFile	KitLogFile.cpp KitLogFile.h	Corresponds to a CSV log file in a KitRepository.
KitLogger	KitLogger.cpp KitLogger.h	Routines for adding or updating entries in a KitLogFile.
KitModel	KitModel.cpp KitModel.h	Routines to draw a line and box diagram/model (experimental routines to automate AOD generation – not fully functional).

University of Cape Town

KitObject	KitObject.cpp KitObject.h	An item that Kit can manipulate.
KitObjectLoader	KitObjectLoader.cpp KitObjectLoader.h	A means to load a KitObject from a file.
KitParser	KitParser.cpp KitParser.h	Used in parsing text files to interpret commands strings.
KitRegion	KitRegion.cpp KitRegion.h	A named region (groups a set of KitObjects, but does not relate to a physical folder – e.g. set of objects that certain keywords).
KitRenderer	KitRenderer.cpp KitRenderer.h	Draw an object using KitGUI commands.
KitReporter	KitReporter.cpp KitReporter.h	Output a report (e.g., information about certain file groups).
KitRepositories	KitRepositories.cpp KitRepositories.h	A list of KitRepositories.
KitRepository	KitRepository.cpp KitRepository.h	Abstract repository class (could be a KitFileRepository or some other kind of repository).
KitResource	KitResource.cpp KitResource.h	A resource used by the Kit library (e.g., an image).
KitSortedList	KitSortedList.cpp KitSortedList.h	Holds a list of objects that has been sorted. Has methods for sorting text in various ways.
KitSpell	KitSpell.cpp KitSpell.h	Wrapper class for aspell, used in checking spelling in text fields.
KitStack	KitStack.cpp KitStack.h	A stack datatype of KitObjects.
KitStream	KitStream.cpp KitStream.h	A stream datatype of KitObjects.
KitString	KitString.cpp KitString.h	A variable length string class.
KitUniquePointerList	KitUniquePointerList.cpp KitUniquePointerList.h	A list which does not hold duplicate pointer values (attempts to add pointer values already in the list are ignored).
KitUtils	KitUtils.cpp KitUtils.h	Various utility functions (e.g., converting string to integer).
KitVar	KitVar.cpp KitVar.h	A kit variable type (e.g., corresponds to system environment variable).
KitXMLIn	KitXMLIn.cpp KitXMLIn.h	Class for loading XML files.
KitXMLOut	KitXMLOut.cpp KitXMLOut.h	Class for saving XML files.

Figure C.4 provides a UML model showing a selection of the KIT classes. As the model shows, many of the classes in KIT inherit the KitObject class. The KitContainer contains zero or more KitObjects, and can itself be treated as a KitObject. The most commonly used classes are KitFile, KitFolder, and KitRepository; these classes correspond respectively to a file within a repository, a folder within a repository, and a repository stored on disk. In terms of using KIT with ESAOA workspaces, a KitFileRepository corresponds to an ESAOA workspaces (this is the repository class

University of Canterbury

used in all the ESAOA tools; however, KIT was design for wider flexibility for use outside of ESAOA workspaces).

A KitAtom and KitRegion are intended to be used with a KitDisplay. KitDisplay is used to display information to a console or (more graphically) in a Window. For version 2 of ESAOA, the console display is operational.

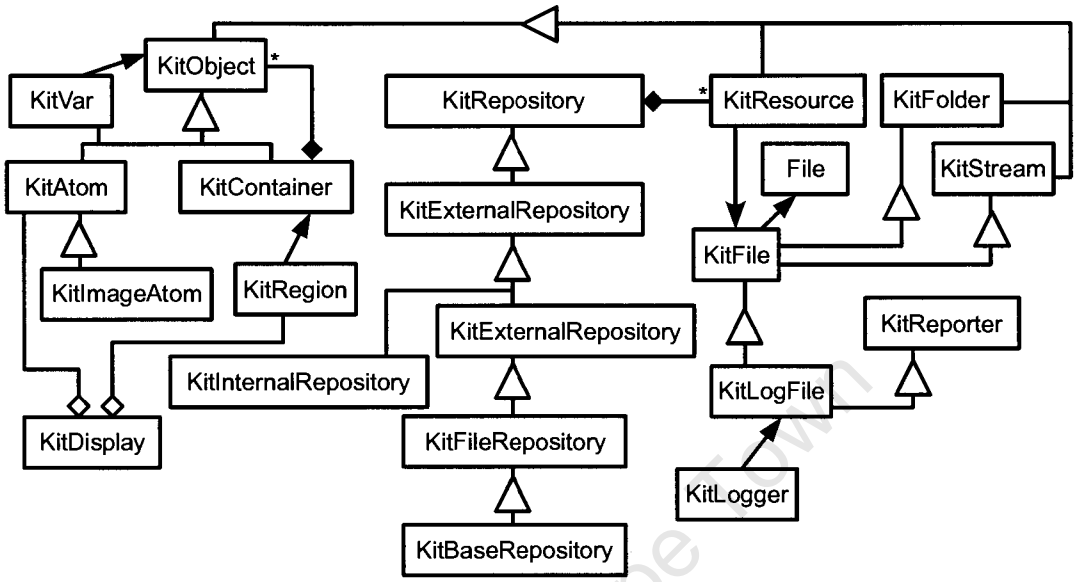


Figure C.4: Containment, dependency and inheritance associations.

The KitLogger class implements most of the functions provided by the esaoa-log program that is included in the communal ESAOA workspace. The KitLogger makes used of a KitLogFile (which maintains a log in CSV file format), and in turn the KitLogFile uses functions it inherits from KitReporter to display lines of text.

A KitResource is used to refer to data (e.g., images or help messages) that is stored within a KIT resource files in repository. These resources can be added, deleted, and changed via KitResource methods. This was intended to allow for a means for programs using the KIT library to access program resources based on a resource ID, without having to know where in the repository the resources are stored as files.

C.4.2 KIT sample application: esaoa-apps

This section describes the implementation of the *esaoa-apps* ESAOA support tool; this is done to provide a scenario showing how the KIT API can be used.

The requirements for the *esaoa-apps* program is to display the number of applications in the active project, and list the folders names are in the directory $P/Software/Applications (note that $P corresponds to the root of the currently active project directory). For each folder that has description metadata attached to it (i.e., in a .fcs CSV file), the description is displayed after the folder name. The displayed information needs to be in the form of a table.

Example: if the project had two folders in its applications directory, named App1 and App2, and only App1 had a description "This is application#1" attached to it, then the following output would be displayed when esaoa-apps is executed:

University of Cape Town

```
+====================================+
| 2 Applications                     |
+====================================+
| Name | Description                 |
+====================================+
| App1 | This is application#1       |
| App2 |                             |
+====================================+
```

The implementation of *esaoa-apps* starts by checking to see if the current working directory if within a valid ESAOA workspace; if it is not, then it displays an error message and exists. Next, the program checks if the current directory corresponds to a folder that is a project folder of a subdirectory of a project folder (all projects folders are stored as a subdirectory in folder $H/Projects, where $H corresponds to the root of an ESAOA workspace). If the current directory is not within a project directory, the program quits; otherwise, execution continues. The code snippet below shows how these first steps of the program are performed using KIT library calls.

```
int main ( int argc, const char** args )
{
 int res = esaoa.init();  // Initialize KIT with ESAOA settings
 if (res) return -1;
     // esaoa.init() returns true if not currently in the
     // ESAOA environment, in which case the program exists.
 ...
 show_app_folders();  // this function continues execution
 ...
 return 0;
}

void show_app_folders ( )
{
 ...
 // The esaoa.AppsDir() returns the path that corresponds
 // to the directory that applications are stored for the
 // current project. If the current directory is not in a
 // project, or the project has no applications, then the
 // esaoa.AppsDir() call returns a blank sting; this blank
 // string would be ignored by the traverse_dir call, leaving
 // the KitList apps empty; otherwise apps will be filled with
 // the names of all the folders in the Applications directory.
 traverse_dir(esaoa.AppsDir(),handle_file,0,0);
 int n = apps.count();
 if (n) {
      ... display directories names and descriptions ...
     } else
     printf("No application directories found.\n");
}
```

The show_app_folders function call above shows the use of the traverse_dir function, which is a function included in the KitFolder.cpp module. This function is given a path (in the case above, it is esaoa.AppsDir()), a function to call, namely handle_file, and the number of subdirectories to descend into (in this case 0), and finally a extra parameter to pass through to handle_file (in this case 0). Essentially, handle_file is called for each folder in the Applications directory and is passed a KitFile that corresponds to the folder concerned (in the example above, this would involve the call handle_file(*app1folderreference*,0) and handle_file(*app2folderreference*,0) – note that the extra parameter 0 is simply passed verbatim to the handle_file function on each call, and *app1folderreference* and *app2folderreference* correspond to KitFiles that would be instantiated to refer respectively to the folder

University of Cape Town

$P/Software/Applications/App1 and $P/Software/Applications/App2). All that the handle_file function needs to do is add the first parameter (i.e., a copy of the folder object) to a list; in this implementation the list is called *apps*. When the traverse_dir function returns, the *apps* list will contain instances of KitFolder objects that refer to the directories within the applications directory. The KitFolder::get_name() method can then be called to display the name of the folder, and a call to KitFolder::get_meta("description") can be used to display the description linked to the folder (note that this call would return an empty string if no description was given).

The flowchart shown in Figure C.5 illustrates the behaviour of the *esaoa-apps* program as described above.

University of Cape Town

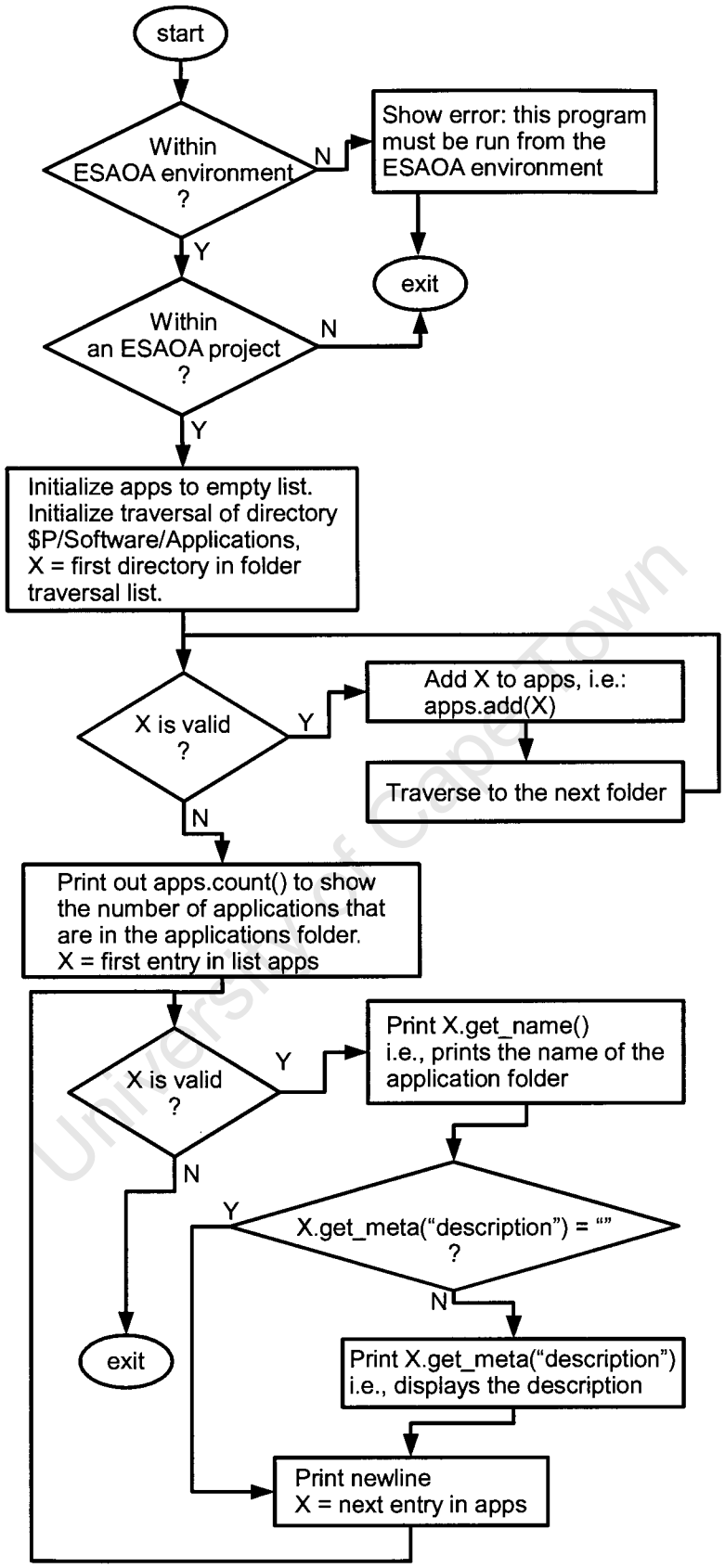


Figure C.5: Flowchart describing operation of the esaoa-apps program.

University of Cape Town

Appendix D: Case study participants

D.1 Experiment 1 participants

The first experiment comprised four participants, each working on a fourth year undergraduate project that involved developing or modifying an embedded system and writing embedded software. The participants worked in two groups of two. In the case of both groups, the developers worked on different higher-level project topics, but collaborated on the development of a common subsystem. In the case of the ANTCON project (P1-2), the common subsystem involved developing interfacing hardware and software to control an antenna pedestal. Similarly, in the case of the Software Signal Generator (SoSiG) project (P1-1), both group members worked on different final year projects, but collaborated on developing the SoSiG systems; in this case however the development of the SoSiG was more of a pre-study for the students as a means to experiment with development tools.

The participants in this experiment were of a similar level, both in the electrical and computer engineering (ECE) programme, and working on embedded systems as part of their final year projects. All the participants had completed the first, second and third year core courses (except EEE3074W) listed in Table D.2.1 below.

D.2 Experiment 2 participants

The case studies used in the second experiment comprised participants registered in the Embedded Systems EEE3074W third-year course at the University of Cape Town. The students are part of the ECE programme. In order to provide details on the experience level of the students, the core course and prerequisite courses that the students would have passed to enter EEE3074W are described in Table D.2.1. An electronic version of the Engineering and Build Environment (EBE) handbook, which includes a detailed description of the current ECE programme, is available from: http://www.ebe.uct.ac.za/images/documents/EBE%20Handbook%202008.pdf. The EEE3074W course was largely project-based which was suited to having the students work on group projects, a practice that is commonly used in embedded systems courses in other universities. The EEE3074W course had three prerequisite courses, namely completion of both the second year computer science courses (CSC2001F and CSC2002S) in addition to completion of the Electronics Engineering course (EEE2040W). These prerequisites were in place because students doing the EEE3074W needed an understanding of designing and implementing circuits containing electronic components (skills gained in EEE2040W), in addition to software design and programming skills (gained in CSC2001F and CSC2002S).

Table D.2.1: ECE programme from first to third year. Source: EBE Faculty Handbook 2005.

Course Code	Description
First year core courses	
CAS1001S	Culture, Identity and Globalization in Africa
CSC1015F	Computer Science 1A
CSC1016S	Computer Science 1B
EEE1004W	Engineering I

University of Cape Town

MAM1003W	Mathematics I
MEC1003F	Engineering Drawing
PHY1010W	Physics
Second year core courses	
CSC2001F	Computer Science 2A **
CSC2002S	Computer Science 2B **
EEE2026S	Electrical Engineering Part 2
EEE2035F	Signals and Systems I
EEE2036S	Probability and Statistical Design in Engineering
EEE2040W	Basics of Electronic Engineering **
MAM2080W	Mathematics II
Third year core courses	
CSC3012Z	Operating Systems I
EEE3064W	Digital Electronics & Microprocessors (also co-requisite / pre-requisite for EEE3074W)
EEE3073S	Professional Communication Studies
EEE3074W	Embedded Systems*
EEE3081F	Control Engineering A
EEE3086F	Signals and Systems II
EEE3084W	Communication System and Network Design

* The EE3074W course was established only in 2005, a year after Experiment 1.
** Prerequisite courses for EEE3074W (i.e., students taking EEE3074W must have passed these courses)

The students were formed into groups of three. The process for allocating the groups involved a series of steps as follows:

1. Students were required to read about an embedded-system related product of their interested, and to produce a short essay reporting on what they read.
2. Each student was then required to post a short project proposal based on their area of interest, describing a system they would like to prototype. These projects did not have to be based on the essay the student write – the essay was merely an encouragement for the students to do some reading which may help inspire them for this second step.
3. Based on the topic of the postings for step 2, the postings were grouped into related fields, such as security, entertainment, military, etc. The students were then assembled into corresponding groups, for example the students interested in security applications were put into the same group.
4. Groups that were too big (more than three members) or two small (less than thee members) had members moved out or moved in so that all the groups were the same size. A size of three was chosen because there were 39 participants, which is divisible into exactly 13 groups. The movement between groups was done to maintain a close-as-possible relation to the student's interests. For example, there were four students specifically interested in security systems (who did the 'Campus Protection Device' Project P2-12) but one of the students were moved to the 'Central Alarm Clock' Project P2-9 so that both projects had three members (the reason for doing so was the potential for the 'Central Alarm Clock' having some aspects similar to that of a security system).

The experiment proper (i.e., when data was captured from the experiment) was only started after the final teams had been allocated, the requirements decided, the high-level designs completed and the teams had entered the implementation phase of the project.

University of Cape Town

Appendix E: Supplementary documentation

E.1 A comparison of search results

In order to obtain an indication of how many literary contributions have been made that relates to KM of embedded software development, in comparison to KM of software development in general, a search was done using Google Scholar. The results shown below were updated on 28 Jan 2009. First searches related to embedded software was performed (see step 1 below) and the total number of results was totalled. Next (see step 2) a search related to KM of software was done and the results were tallied. To perform a percentage the first total was divided by the second; i.e.: (total embedded software KM hits) / (total software KM hits).

Step 1: Searches performed for KM of embedded software development

Search Expression	Estimated Hits
"embedded software" "software development" "knowledge management"	348
"embedded system" "software development" "knowledge management"	129
"real-time system" "software development" "knowledge management"	107
Total:	584

Step 2: Searches performed for KM of software development

Search Expression	Estimated Hits
"software development" "knowledge management"	14,000
Total:	14,000

Step 3: Percentage of step1 results to step2 results

584 / 14000 = 0.042 = 4.2%

Based on the estimated number of hits provided by Google Scholar, the number of embedded software KM hits was 4.2% of the number of general software KM hits.

University of Cape Town

G Glossary

This section lists all important terms used in this thesis. All terms are presented in singular form. Underlined terms in definitions indicate that the term (in its singular form) is defined elsewhere in the glossary. Synonyms for a term are listed after the definition of the term.

The ESAOA KMS users (i.e., knowledge worker roles such as component researcher and process engineer) do not need to understand all these terms. Different roles do not need to know the same set of terms. This glossary is designed in relation to the second version of the ESAOA KMS (detailed in Chapter 6); correspondingly, the acronyms [CR], [WA], [PE], [IE], and [CKS] are placed in the 'term' column to indicate respectively terms the component research (CR), workspace administrator (WA), process engineer (PE), innovation engineer (IE) and communal knowledge steward (CKS) need to understand. Terms that need to be understood by all roles are indicated using a [*]. The chief knowledge officer (CKO), or a developer wanting to customise an ESAOA KMS, needs a broader understanding of this terminology. Terms in the ESAOA Knowledge Ontology (Appendix C.1) that are seldom used are not repeated below.

Term (Acronym)	Description
Activity [*]	A form of work that can be performed by a person filling a certain role. See: ESAOA activity
API [*]	Application Programming Interface
Application framework [*]	A set of software libraries or modules used to implement the standard structure of a software application within a particular application domain. This application domain is defined in terms of an operating system, hardware platform, operating environment, and general requirements relating to what the software does (e.g. connecting microcontrollers in a sensor network). By integrating a significant amount of reusable code into an application framework, development time can be saved by reducing the amount of code that needs to be written for new applications in the application domain concerned.
Artefact	Artefacts are objects used or worked on by developers in a project. Artefacts are broadly classified as soft artefacts or hard artefacts. See: implementation artefact.
Artefact adaptation (AA)	An activity in which a developer creates or modifies an implementation artefact.
Artefact organisation (AO)	An activity in which a developer classifies or organises implementation artefacts (e.g., changing file names or arranging files in directories).
Artefact organisation and adaptation (AOA) [*]	Method of organising artefacts in a work area (e.g., file directory or code module) and adapting these artefacts to accomplish specific needs.
Artefact Organisation Diagram (AOD) [*]	A diagram used to represent the organisation of an ESAOA workspace.
Automatable process knowledge [PE]	Process knowledge that can effectively be represented in the form of a script (specifically, in the case of the ESAOA KMS, this type of knowledge refers to knowledge that can be effectively represented as one or more Bash scripts or a combination of Bash scripts and C++ executable programs.

University of Cape Town

Back end	This term refers to software system components that process the output from the front end.
Background program	Background programs, or 'daemons', run without user interaction and provide services to other programs [Enderunix.org, 2008].
Bash [PE+IE]	Bash is a shell available on most Linux and GNU-based operating systems. Bash includes aspects of both Korn shell (ksh) and the C shell (csh). Bash offers a variety of functional improvements over sh for both programming and interactive use. In the ESAOA KMS, Bash was used as the scripting language for ESAOA workspaces and to capture automatable process knowledge.
Boundary artefact [*]	An artefact used to in the transfer of ESAOA knowledge. Generally refers to a soft artefact in an ESAOA workspace.
Breadboard [*]	A board on which experimental electronic circuits can be temporary constructed without using solder.
Capability [*]	Ability of a person to complete a specific set of actions of a particular level of complexity, within a certain context, and for a specific purpose. Capability is a result of learning, skills, abilities and values.
Chain of command	The chain of command is a term commonly used in a military context to indicate the line of authority and responsibility that orders are passed along.
Chief knowledge officer (CKO)	The CKO is responsible for administrating the KMS as a whole and for training knowledge workers in its use.
Commercial off-the-shelf (COTS) [*]	Commercial off-the-shelf (or COTS) refers to a commercially manufactured system that is integrated (and possibly adapted) for use as a component in a product, fulfilling certain requirements for the product. Source: http://en.wikipedia.org/wiki/COTS
Communal knowledge	A type of knowledge shared between members of different development teams in the same organisation.
Communal knowledge steward (CKS) [*]	Role responsible for helping the CKO manage communal knowledge and maintain the communal workspace.
Communal workspace [*]	An ESAOA workspace shared between all teams.
Communities of practice (COPs)	A community of practice (COP) is a group of volunteers who learn and work together. Generally, the members of a COP generally have similar jobs and skills.
Component	A part from which a product is built. A component can be considered a special form of implementation artefact. Components can be separated components into software and hardware components.
Component integration	The process by which a software developer writes or adapts code to connect components.
Component knowledge [*]	Knowledge related to a component (e.g., the function of certain control pins on an integrated circuit). Component knowledge could also be categorised as data, process, or innovation knowledge depending on which form of knowledge was produced during the knowledge occurrence concerned.
Component researcher (CR) [*]	A knowledge worker role found in the second version of the ESAOA KMS. This role is responsible for locating and studying components, and provides highly focused

	component knowledge to the PE and IE.
Concept [*]	A product idea.
Concurrency [*]	Each high-level task involving multiple simultaneous lower-level operations taking place.
Concurrent Versioning System (CVS)	CVS is an open-source collaboration and version control system. Source: http://www.nongnu.org/cvs
Corroborate [*]	Checking that information is valid.
COTS [*]	See Commercial off-the-shelf
Cross-compiler [*]	A cross-compiler is a type of compiler used to create executable code for a platform than is different to the one on which the cross-compiler is run on. Source: http://en.wikipedia.org/wiki/Cross_compiler
CSB337	An embedded system platform used in the experiments discussed in chapters 5 and 6. Details of the platform is provided by [Cogent Computers, 2005].
Daemon	See background program.
Data knowledge [*]	Knowledge of information sources and understanding of knowledge artefacts (e.g., knowledge produced while reading a datasheet).
Data Steward (DS) [*]	A knowledge worker role used in the first version of the ESAOA KMS. This role was responsible for the location and study of data knowledge.
Data synthesis	The process of collating data into a systematic representation, from a variety of sources that store data in dissimilar representations. Source: http://dictionary.reference.com/browse/synthesis?
Embedded engineer [*]	An engineer who develops an embedded system. An embedded engineer can be a hardware engineer, or a software engineer, or both. But generally, an embedded engineer needs a general understanding of both the hardware and software aspects of embedded systems.
Embedded system [*]	A single-purpose computer built that is built into a larger system to control and monitor the larger system. See definition in Ball [Ball, 2002, xi].
Embedded system artefact organisation and adaptation (ESAOA) [*]	The process of structuring, arranging, and adapting implementation artefacts during the implementation of an embedded system product.
Embedded system engineer (ES engineer) [*]	See: Embedded Engineer
ES [*]	See embedded system
ESAOA [*]	Embedded system artefact organisation and adaptation
ESAOA activity [*]	The act of performing embedded system artefact organisation and adaptation.
ESAOA conceptual modelling language [*]	A specialised modelling language, which is an adaptation of the UML, used in modelling aspects of the ESAOA KMS and interrelations between aspects of this KMS.
ESAOA directory structure [*]	A file directory organised according to the organisation and naming guidelines specified by an ESAOA KMS.
ESAOA knowledge management system (ESAOA KMS)	A KMS design to facilitate KM within ESAOA activities to aid the successful completion of implementation tasks.
ESAOA knowledge ontology (or ESAOA ontology) [*]	A specialised terminology structure used to specify the ESAOA KMS.

ESAOA support tools [*]	Software tools and scripts that assist in maintaining an ESAOA workspace and for accessing the knowledge base stored within it.
ESAOA workspace [*]	An ESAOA workspace is a digital, computer-based work area that comprises: 1) a shell environment; 2) soft artefacts (i.e., computer files) organised into an ESAOA directory structure; 3) an integrated knowledge base; 4) ESAOA support tools; and 5) a related collection of externally stored and maintained development tools.
Evaluation board [*]	A pre-built hardware platform used for evaluating a certain microcontroller architecture and set of peripherals [Berger, 2002].
Event chain	A sequence of associated knowledge events leading from a problem to a solution or dead-end.
Executable [*]	A file that can run on a platform. See: executable image.
Executable image [PE+IE]	A binary file that contains a program; can be installed and executed on a platform.
Final product	The product that a project is focused on building.
Framework construction	Part of the research design described in Chapter 3 that involves designing and implementing the documentation, supporting tools and other parts of the ESAOA KMS.
Front end [*]	In software engineering, the term 'front end' refers to a part of a software system that interacts directly with the user. Partitioning a system between 'front ends' and 'back ends' is an abstraction that helps to keep the different parts of the system separated.
FTP [PE+IE]	The File Transfer Protocol (FTP) is a TCP/IP standard for transferring files across the internet or between networked computers.
Functionality classification index file (.fci) [*]	An .fci file resides in the root of each ESAOA workspace. It is a comma separated values (CSV) that stores classification terms used by .fcl files.
Functionality classification lookup file (.fcl) [*]	An .fcl file resides in each folder within an ESAOA workspace that contains files that have classification metadata linked to them. All classification keywords added to an .fcl file should be defined in the .fci file for the ESAOA workspace.
Hard artefact	Physical objects an ES engineer works with in a laboratory, e.g. electronic components, hardware devices, tools and equipment (see Section 1.1.6).
Hardware component	A part of the product that is hardware (e.g., microchip).
Hardware platform [*]	A hardware platform is the minimum required hardware components, circuit board, chips and other electronics needed to support the software platform around which embedded software is constructed.
Implementation Artefact [*]	Implementation artefacts are objects used or worked on in the implementation phase of a project. These objects store knowledge (in the form of information), are used as tools, or form part of the product being developed. Implementation artefacts are accordingly divided into two categories: soft artefacts or hard artefacts.
Implementation Knowledge [*]	Implementation knowledge is used by a developer during the implementation phase of a development project when the developer transforms a design into a product.

University of Pretoria

Implementation phase	The fourth phase of the embedded systems lifecycle model presented in Section 2.2.
Implementation tasks	Tasks carried out in the implementation phase [Schach, 2005]. ESAOA activities are carried in many implementation tasks. In this thesis, 'implementation' refers to the process by which development moves from a product concept/design to a final product; in this case, the act of implementation refers more to a process of building, coding and piecing together parts of an electronic circuit, as apposed to a process of designing and modelling a system.
Information source [*]	Resource from which information is obtained, e.g., a document.
Innovation [*]	Innovation is a process for converting concepts and knowledge into better ways of performing tasks, into novel products, or into new services.
Innovation engineer (IE) [*]	A knowledge worker role of the ESAOA KMS responsible for innovation. In version 2 of the KMS, this role depends on the process engineer and component researcher to provide component knowledge, techniques and solutions needed to test design concepts.
Innovation knowledge [*]	Knowledge related to innovation, such as which concept designs work effectively.
Innovation Process [*]	Incorporates the various strategies by which innovation occurs, including: research and development, commercialisation, and technology diffusion.
Instructions [*]	A command that causes the processor to perform a certain action, for example to add two numbers stored in registers. Also referred to as opcodes or machine code.
Intangible knowledge assets	Non-physical assets. Includes patents, copyrights, contracts, trademarks, and goodwill. Contrasts to physical assets such as equipment, and vehicles. Source: http://en.wikipedia.org/wiki/Intangible_assets
Intellectual capital	Knowledge that is of value to an organisation, intellectual capital is made up of human capital, structural capital, and customer capital [Edvinsson & Malone, 1997].
Intellectual property (IP) [*]	The intangible property that results from creativity. IP can be legally protected using patents and copyrights.
Inter-Integrated Circuit (I2C) [*]	The I2C bus (pronounced I-squared-C) is an inter-device communications bus invented by Philips. It is used to connect *low-speed* peripherals in an embedded system. Source: http://en.wikipedia.org/wiki/I2C
IP [*]	Internet Protocol (IP) is a packet-based protocol for delivering data across networks. Source: www.ipowerweb.com/hostingdictionary/
Joint Test Action Group (JTAG) [PE+IE]	JTAG is a standard for providing external test access to integrated circuits serially, via an external interface. Source: http://www.netsilicon.com/support/embeddedglossary.jsp
KB * [*]	See: Knowledge base
KIT API	See Kit for Information Technology.
Kit for Information Technology (KIT) [PE]	A C++ application framework and set of libraries included in the ESAOA communal workspace to facilitate development of ESAOA programmes that need to navigate ESAOA workspace directories.
KMS analysis	Part of research design presented in Chapter 3, which

	involves analysis data obtained from development teams working on ES development projects.
KMS aspect [*]	A KMS is broken into four KMS aspects to assist the description of the system by describing each aspect independently by abstracting the other aspects.
Knowledge [*]	Knowledge is information combined with understanding and capability.
Knowledge acquisition (KA) [*]	A knowledge management activity that involves obtaining knowledge from a source outside the organisation.
Knowledge analyst	The role of the knowledge analyst in a KMS is similar to that of a system analyst in a software engineering project. The knowledge analyst studies the KMS, such as procedure that users employ, or physiological impacts of the system on the users and the resultant effect on the working of the organisation. Typically, the knowledge analyst uses statistics to discover means to make the KMS more efficient.
Knowledge artefact [*]	A knowledge artefact contains information about another soft artefact or hard artefact. See knowledge object.
Knowledge audit [CKO, TL, WA]	A knowledge audit is done by the workspace administrator in ESAOA KMS version 2 to determine statistics relating to use of the KMS, e.g., which artefacts in the communal workspace are most valuable.
Knowledge base (or KB) [*]	Generally speaking, a knowledge base (KB) comprises a collection of interrelated concepts. A KB assists in archiving and sharing explicit knowledge for later use. Often, a knowledgebase is implemented using a database application to capture a wide range of information. However, in the ESAOA KMS, this term refers to a collection of knowledge artefacts interwoven in an ESAOA workspace. Specific ESAOA support tools (e.g., *fclass*) assist in accessing or manipulating the KB stored within a workspace (e.g., accessing file meta-data and searching code comments).
Knowledge dissemination [*]	A knowledge management activity in which a person, or group of people, share their knowledge with others.
Knowledge economy	The term 'knowledge economy' typically refers to using knowledge to produce economic benefits [Wikipedia, 2007]. A knowledge economy is based on the production, distribution and use of knowledge as the major driving force for growth, wealth creation and employment opportunities in industries [Drucker, 1998]. Source: http://en.wikipedia.org/wiki/Knowledge_economy
Knowledge engineer (KE) [*]	The KE role designs and implement the support infrastructure for a KMS, updating or pruning the system while it is in use (see Section 2.7.6).
Knowledge event [*]	An action carried out by a knowledge work at a specific time that involves the production or management of knowledge, e.g., finding information, applying a process, testing an idea, or solving a problem.
Knowledge event chain	See: event chain
Knowledge management (KM) [*]	The capture, organisation, classification and dissemination of knowledge [McDermott, 1999a].
Knowledge management	The governing body that establishes and directs a KMS

University of Wollongong Thesis

steering committee	in an organisation, more common in large corporations.
Knowledge management system (KMS) [*]	A system for performing KM within an organisation, which supports the creation, capture, storage and dissemination of knowledge among members of the organisation [Alavi & Leidner, 2001].
Knowledge object	A knowledge object (called a knowledge artefact in this thesis) is a document or collection of documents in which knowledge is maintained [Knorr-Cetina, 2001]. In this thesis, certain soft artefacts can be classified as knowledge artefacts.
Knowledge occurrence	A numbered knowledge-producing event that occurred in an event chain.
Knowledge occurrence graph	Plot of the accumulation of productive and non-productive knowledge occurrences ordered chronologically on the x-axis, most recent knowledge occurrence on the left.
Knowledge production	Actions carried out by a knowledge worker to learning new knowledge.
Knowledge worker [*]	The term 'knowledge worker' refers a person whose primary form of work involves using and creating knowledge [Drucker, 1998]. An example is an engineer who accumulates and uses knowledge related to development tools and electronic components used to construct an embedded system.
Management of knowledge	See: Knowledge Management.
Metadata	Data about data – in this thesis, the term is generally used to refer to classification information about a particular file stored in an ESAOA workspace.
Metafile	A file that stores metadata about another file.
Non-productive event	A knowledge event not used in developing the final product (i.e., or resulted in a dead-end).
Non-productive time	Time (in hours) that a knowledge worker spent on non-productive events.
Organisation	In terms of KM, an organisation refers to a collection of people (for example the employees of a company).
Organogram [CKO, TL]	An organogram is a chart showing the structure of an organisation. An organogram is primarily used to show lines of authority (or chain of command) and the way in which information flows in the enterprise. The chart is used to indicate that all members of the organisation (or project team) understand the operations and information flows in the organisation.
Paradigm	An understanding of how things work. Paradigms are used to set the basic rules of how things operate within a specific environment.
Partitioned Time History Calculator (PTHC)	Inputs a knowledge register spreadsheet and outputs the sequential accumulation of productive and non-productive time (for ESAOA version 1) or productive and non-productive knowledge occurrences (for version 2).
PCB [*]	See: Printed circuit board
PEP [*]	A program added to ESAOA version 2 that runs as a background process to access functionality classification index files in ESAOA workspaces.
Personal knowledge	A special case of knowledge management in which a

University of Cape Town

management (PKM)	single user is the only user of the KMS.
Personal workspace	An ESAOA personal workspace is a copy of a team workspace that an individual works on. It is usually synchronized with the team workspace using version control software.
Platform [*]	In this thesis, the term 'platform' describes a combination of hardware and operating system on which embedded software executes. Generally, the hardware aspect of a platform (often termed the hardware platform) is explained in terms of a microprocessor or microcontroller architecture, and the peripherals available to it. The software aspect of a platform (often called the software platform) is defined in terms of operating system and runtime libraries. A platform on its own can be considered a 'blank slate' that needs application software installed on it to make it do perform useful operations.
Printed circuit board (PCB) [*]	A printed circuit board (PCB) is a thin board usually made of fibreglass on which hardware components are mounted. Connections between the components are printed onto the board (hence the name). A printed circuit board joins electronic components without discrete wires. Source: http://en.wikipedia.org/wiki/Printed_circuit_board
Process engineer (PE) [*]	A knowledge worker role of the ESAOA KMS responsible for producing process knowledge.
Process knowledge [*]	Knowledge related to performing tasks, such as using a tool to compile embedded software.
Processor [*]	A processor is an electronic device that manipulates data according to a sequence of instructions.
Product	In this thesis, the term 'product' relates to an embedded system that is being built in a project.
Productive event	A knowledge event within an event chain that led to a final solution (i.e., did not result in a dead-end).
productivity graphs	Plots productive time and non-productive time on the vertical axis against knowledge event numbers ordered chronologically on the horizontal axis.
Project management [TL]	The process of planning, managing and coordinating all elements of a project from the start to the end.
Productive time	Time (in hours) that a knowledge worker spent on productive events.
Prototype (or ES prototype)	A prototype serves as an early product sample that is built to test a concept [Floyd, 1984] or to determine experimentally an effective means to build a product [Brinkkemper *et al.*, 1996]
Quality management [TL]	Organisational structures, protocols, responsibilities and evaluation methods to ensure development teams deliverer a product to specified standards.
Rational Unified Process (RUP)	The RUP is an iterative software design method that describes how to produce and deploy software effectively using proven techniques. It was created by the Rational Software Corporation (now a division of IBM). Source: http://en.wikipedia.org/wiki/Rational_Unified_Process
Research and development (R&D) [*]	R&D is creative work performed systematically to acquire knowledge to produce new products or applications.
Role [*]	The part an individual plays in a KMS, specified

University of Pretoria

	according to contributions made by that person's through application of their knowledge and abilities. A role describes a type of person involved with a KMS. One person can perform multiple roles; for example a person can be both the CKO and a knowledge engineer.
Serial Peripheral Interface (SPI)	The SPI is a general-purpose high-speed synchronous serial interface originally produced by Motorola. Source: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
Shell [*]	A command language interpreter, or in the more specific case of an operating system shell, a shell refers to a software tool that provides an interface to users allowing the user to use a language for passing data between, and invoking, operating system routines and programs.
Soft artefact [*]	An implementation artefact in the form of a digital file stored on a computer or a paper document. A soft artefact that stores information about a hard artefact or a soft artefact is considered a knowledge artefact.
Software component	A component that represents a software part of a product (e.g., a code file).
Software platform [*]	A software platform* is the framework around which software is constructed. In this thesis, the term refers to the lower level of software on which application software is deployed. In ESAOA, Platform Deployment Modules (PDM) can reside in the software platform, or may exist between application code and the software platform. *Source: http://en.wikipedia.org/wiki/Software_Platform
SPI	See: Serial Peripheral Interface
SQL	Structured Query Language. Used to control or modify a database, or to extract data from a database.
Steering committee	See knowledge management steering committee.
Stereotype	A facility provided in UML to extend the vocabulary of UML to create new model elements derived from existing ones [OMG, 2005].
TCP/IP	Transmission Control Protocol/Internet Protocol (or TCP/IP for short) refers to a standard suite of Internet communication protocols used to communicate between computers. The TCP protocol is responsible for an error free connection between two computers, while the IP protocol is responsible for the data packets sent over the network. Source: http://en.wikipedia.org/wiki/Internet_protocol_suite
Team leader (TL) [*]	The role assigned to a person allocated to lead a development team. Generally this individual is involved with administrative tasks, such as allocating tasks and checking team member performance. This role needs leadership skills and an understanding of the organisation's quality requirements.
Team member (TM) [*]	A member of a development team.
Team workspace	An ESAOA team workspace is an ESAOA workspace that is shared by a team of engineers. It is usually the master version of the ESAOA workspace that is synchronized with team members' personal workspaces.
TFTP [*]	The Trivial File Transfer Protocol (TFTP) is a simplified version of FTP that lacks authentication and relies on UDP rather than TCP for transferring data. As for FTP,

	the system involves a TFTP client and TFTP server (the client requests files to uploads or download to/from the server). TFTP is easier to code than FTP. See also FTP. www.idea-esolutions.com/glossary/index.php3
Tool chain [*]	A tool chain is a set of software tools used to create a software application. The tools may be used in a chain, the output of each tool becoming the input for the next. This term can refer to any set of linked development tools. Source: http://en.wikipedia.org/wiki/Tool_chain
Tool knowledge [*]	Knowledge related to a tool (e.g., the function of certain command line arguments for a compiler). Tool knowledge could also be categorised as data, process, or innovation knowledge depending on which form of knowledge was produced during the knowledge occurrence concerned.
Topology	The way that components of a system (or a certain subject matter) are arranged or interrelated.
Workflow [*]	A sequence of activities carried out to achieve a certain objective. A workflow can involve activities carried out by people of different roles.
Workspace [*]	See ESAOA workspace.
Workspace administrator (WA) [*]	A role of ESAOA KMS version 2 responsible for maintaining the integrity of a team workspace. The WA supports other team members in terms of organising and locating artefacts.

University of Cape Town