# New Models and Patterns For Traceability

Justin Kelleher



Dissertation submitted to the University of Cape Town
in fulfilment of the requirements for the degree of

**DOCTOR OF PHILOSOPHY**

in

Computer Science

Supervisor: Gary Marsden

February, 2008

Keywords: traceability, modelling, semantics, process, patterns, empirical study.

# New Models and Patterns
# For Traceability

Justin Kelleher

**(ABSTRACT)**

Traceability is a critical software engineering practice that manages activities across the product development lifecycle. It is the discipline of getting an entire organisation to work together to build better quality products. Traceability is also about relationships between traceability items, the management of change and requires good communication between personnel on matters that impact the system in any way.

At the start of the 21st Century it is evident that there was a proliferation in new traceability research promoting techniques from a number of emerging research communities. However, some researchers still report that there are still many problems, in particular the lack of empirical data from small, medium and large organisations.

In this study we address this shortcoming by performing two empirical studies. Firstly, we carry out a four year case study investigating traceability in a large multinational that develops complex enterprise systems. Ericsson's is a world leader in the development of large telecom's systems and is renowned for their mature development processes, tools and highly skilled staff. We examine the state of the art at Ericsson and the factors that influence traceability, paying particular attention to how these factors change during the study and the impact that these changes have on the traceability practices. Secondly, we execute an industrial survey across nineteen corporations to further our understanding of traceability in small and medium sized organisations.

Using this empirical data as the major design inputs, we design and test a Traceability Framework consisting of three solution components namely, a *TRAceability Model* (TRAM), a *TRAceability Process* (TRAP) and *Traceability Patterns*. The TRAceability Model (TRAM) consists of *semantic models*, designed using a layered approach, with each layer presenting traceability semantics from different user perspectives. The TRAceability Process (TRAP) consists of process models also utilising a layered approach but in this case capturing process elements that can be used in the creation of a traceability process in a variety of different contexts. At the lowest layer the models represent the actual traceability situation in a project at Ericsson. While patterns are a widely accepted method for describing best practices and recurring problems in many aspects of software development, they have not been applied to the field of traceability. Structural patterns emerged from the semantic and process models. Furthermore, we utilise a pre-defined pattern template for formalising the findings of the empirical data and communicating the outcomes to different users. The three components together promote better communication, reusability and understandability of traceability concepts and practices.

On the whole, this study has shown promise in a new modelling approach for describing aspects of traceability. Furthermore, it has shown the feasibility of using traceability patterns as a technique to overcome some of the problems identified during the empirical study.

# Acknowledgements

First and foremost, I wish to thank my advisor, Dr Gary Marsden, whose input and reviews greatly helped me with this work.

Thanks are due to my family; Ciara, Gary, Maeve, Owen, Michael, Annette, Roisin, Peter, and my little godson Fintan, who were all there when I needed them most. A special word of thanks to my parents, who never really understood the doctoral process but offered love and much needed support for its duration. A lawyer, a medic, a teacher and now a doctor-sit back and smile, you have done a great job. I love you both dearly.

Thanks to my friends, Dan, Gus, Sully, Gary L., Louise, John, Pat, Steve, Sharon and Aidan who provided the encouragement to start this research and support in many different ways throughout this project.

Without enumerating their names, I acknowledge the support and feedback from many collaborators especially the South African SPIN members and Cora Systems, who were very considerate to give me the time of work to carry out the many field trips.

A special, thanks to Ericsson, who let me carry out the Case Study. There are simply too many of you to thank individually, so thanks to you all.

To my Uncle Paul, the Old Man of the Sea, who in the final stages, provided invaluable, feedback, support and constant encouragement. Paul, you are a kind and generous man. BRAVO ZULU.

This dissertation would not have been possible, without the love and support of my fellow doctoral student Vanessa Rochester. It was the best time of my life and you brought me so much happiness throughout the whole process. I will never forget you.

Lastly, to my son Joshua, who I thought of every single day over the past four years and who gave me the strength to keep going. I can't wait to hand you this doctorate some day-because this is for you. I love you.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# Chapter 1 INTRODUCTION

## 1.1 INTRODUCTION

"*You tell me why. I will tell you why not*"

-Caroline Brennan, Project Manager Cora Systems.

When we first started to look at the concept of "traceability" as a viable subject worthy of doctoral research, the multitude of literature and the scope of its application were overwhelming. In the last twenty years, significant advances have been made in the field of traceability, however, the reality today is that software products still fail to meet the needs of the customers.(Hayes et al., 2006, Standish_Group, 2003)

In early 2004, we began informal discussions with practitioners and it was startling to find that so many people still lack an appreciation of the benefits of traceability. Today many larger organisations see the rewards of investing in sophisticated traceability tooling environments; however as we will show in this study, many smaller organisations still don't invest the time, budget, or resources to ensure that traceability is practiced by the entire development organisation.

While doing some consultancy, in 2003, with a project management company, the issue of trying to implement certain traceability practices into their company was raised. The project manager responded with "I spend my day fire fighting. In the morning I sort out the support issues from my last project, in the afternoon I deal with today's urgent design issues and in the evening I communicate with all the different customers. Traceability is a great idea in theory, but I don't have the time, budget or resources to apply this practice in my projects. Even if I did, I do not believe it will elevate the problems I have with support, implementation or customer related issues. You tell me why. I will tell you why not". While this statement does not give a true reflection of the current state of attitudes towards traceability, it does give us an insight into some of problems that software companies have when implementing traceability.

Pivotal to our research, is to understand the "why-not", by gaining an insight into the current state of the art of traceability in a variety of different industrial contexts. This includes understanding the factors that influence the implementation of traceability, not just in the large organisations but also in the small to medium enterprises and to provide a simple, easy to use but effective solution that meets the needs of the user community.

## 1.2 INTRODUCTION TO TRACEABILITY

Before launching into a detailed analysis of the problem space and the solution suggested by this work, it is essential to orientate the reader with an introductory explanation on traceability and illustrate its far-reaching importance.

### 1.2.1 A Review of Traceability

The problem of traceability is almost as old as man himself, since man became a hunter, which is several million years ago in the epoch of the Pliocene, when he learned to track his prey. The concept of traceability is neither a recent nor a novel construct. The notion that one must track, finds its roots in our earlier hunter-gather societies where such tracking was essential for survival. That is why, the etymology of the word "trace" is derived from the verbal root "track". The Latin translation of the word "trace", *vestigium,* actually means *footprint*. The Oxford English Dictionary, definition for the word "trace" is: The ability to "delineate" and "mark out" "perceptible signs of what has existed or happened" in the lifetime of a person, creature, man-made product or idea, to enable one to "pursue one's way along" this record.[1](Simpson and Weiner, 1989)

Mankind's interest in traceability in the context of history is as old as man's interest in genealogy: the study of the family relationships, tracing the line of descent backwards from children to parents and their ancestors. As one traces further back in time, with each step and generation, the number of persons of potential interest doubles, such that the number reaches astronomic proportions in 1000 years. Any genealogical study must therefore be able to deal with vast amounts of documentary material. The postmodernist philosopher Michael Foucault described genealogy in his essay "Nietzsche, Genealogy, History", as "gray, meticulous, and patiently documentary", requiring that "documents be scathed over and recopied many times". Genealogy, consequently, requires immense patience and knowledge of details, and it depends on a vast accumulation of source material."(Foucault, 1984). As we shall show in due course, the same requirements for patience, knowledge of details, and the ability to handle a vast accumulation of source material, is equally applicable to the assurance of traceability in the modern technological world. This discussion of genealogy exemplifies the point that traceability, in many instances, is about tracing relationships and dependencies.

Indeed, traceability is fundamental to all science, and scientific research. In order to investigate a problem, a scientist must conduct his investigation in a systematic manner, and be able to explain his results logically. In order to do so he will usually employ a functional model that can simulate past events, and with this model he can make predictions for further observations. This makes the point that tracing and tracking are not just about looking back, but also about looking forward. Even ancient man the hunter was more concerned with where his prey was going than where it had been.

Today, traceability has become the focus of a major trade dispute between the United States and the European Union. In 2002, the European commission mandated the general principles and requirements of food law defining traceability as the ability to trace and follow food, feed, and ingredients through all stages of production, processing and distribution.(European-Commission, 2002) This resulted in all products being used in the food industry being traceable to the original suppliers. Every animal born must be given a

[1] Gotel and Finkelstein also used this definition in their paper *"An Analysis of the Requirements Traceability Problem"*

passport with details of its birth, mother, health and farms. A review made by the US based industry consultancy, ARC Advisory Group (ARC-Advisory-Group, 2007), summarised the dangers faced by companies with ineffective traceability of finished product**(s)**: "Bungled product recalls can bankrupt companies. The most spectacular example was Japan's biggest dairy producer, Snow Brand. The company lost $107 M due to a food poisoning scandal that left 8000 people sick. Snow Brand, unable to identify contaminated batches, withdrew all its products from retailers' shelves".

In the 1960s, software development was in a state of crisis (Gibbs, 1994). This "software crisis" was characterized by: grossly inaccurate estimates of schedule and cost; quality that was less than adequate; and a level of productivity that could not keep up with demand. In order to overcome the problems associated with this "crisis," it was determined that software programming would have to become more disciplined, to evolve into an engineering science (Gibbs, 1994). To become more disciplined meant that software products would have to be "planned, designed, constructed, and released according to engineering principles." In the decades to come, improvements to the development process would result in advances that would move software development towards the disciplined practice of "software engineering."

By the 1990s, demands for more sophisticated approaches to requirement engineering were being made, driven by a number of major software disasters. In 1994, the Federal Aviation Administration (FAA) cancelled a 10-year effort to modernize the nation's air control system. About $1.3 billion was written off.(Assumptions, 1998) In 10 years, the requirements elicitation phase had never come to closure. A requirements specification with a size that could be measured in weight was produced, but it was fundamentally incomplete. One can argue that there were many other problems with the program, but post project analysis showed that it was during the requirements process that the program failed. Also in 1994, The Standish Group International, Inc., conducted a survey of software application development projects in which 365 companies representing more than 8,000 software projects responded. (Standish_Group, 1995)Their report included the following findings: (1) only 16.2% of the projects were completed on-time and on-budget, (2) About 30% of the projects were cancelled before they were completed, and (3) 50% of the projects were completed and operational, but over-budget, over the time estimate, and contained fewer features and functions than originally specified. Additionally, for the projects that were either cancelled or were completed: the average cost overrun was 189%; the average schedule overrun was 222%; and the features and functions included in the final product were only 61% of that originally called for. It is therefore imperative to note that **w**hilst the reasons for these software project over-runs and catastrophes are many, a common theme is a lack of project organisational discipline that comes with traceability.



## 1.3 BACKGROUND TO RESEARCH PROJECT

Before beginning work in Ericsson the author completed a degree in Engineering and post-graduate degree in Software Design and Development. Between 1997 and 1999, the author worked at Ericsson, Ireland as a senior technical trainer in PSTN[2] telecommunications technologies. Ericsson's main switch was called the AXE, which by today's standards was the Tyrannosaurus Rex of telephonic switching with a massive hardware backbone using old procedural real-time switching software called PLEX

[2] Public Switching Telephone Networks or simply landlines were the main components of the telephone networks worldwide in the mid-nineties.

(Programming Language for the AXE), which had its origins in the 1970s. In this timeframe, the author gave courses in the PLEX and the two supporting processes; the design process which was called MEDAX (Method for the AXE) and the Project Management process, PROPS. While both methods suffered the pitfalls of "waterfall development" processes, they did however provide a common framework for communicating all matters related to AXE development in a uniform and consistent manner. Furthermore, Ericsson also developed their own propriety tools to support the underlying processes.

By the late nineties, the race for 3rd generation systems began, and in 1997 Ericsson and Rational entered into a joint development initiative (JDI) project and a new era of software development began based on the Object Orientation technologies. It was during this JDI project that the author moved to Montreal, Canada, to work in the software process development team, and then eventually as an instructor and mentor in the roll-out phase of this project. It was during this period that the author began to understand the true importance of Requirement Engineering and the implicit need for traceability. At the start of the 2000s the author started working in projects developing the 3rd Generation (3G) Operating Support Systems (OSS's), which is the domain that we base our case study and for which the challenges that were encountered we shall elaborate upon later.

In late 2000 the author joined a Key Project Assessment (KPA) team at Ericsson with the primary objective of assessing all critical 3G projects with the primary goal of improving the quality of the development and delivery of those projects and products. These assessments provided the author with further insights into the factors that influence the implementation of requirement engineering and traceability, for better and worse. As a result of the KPA team's reports, a Project Acceleration Teams (PACT) was formed with an expert from each discipline in the development lifecycle to speed the design and delivery of the products, and the author was dispatched to work in one of these PACT's (Project Acceleration Team's) as a requirement manager and process prime for the C-MGW [3]in Helsinki Finland. In the final few years of his work in Ericsson he became the Requirement Manager for the RANOS product, then process prime, line manager and integration development environment engineer. It was with this experience that the author moved to the University of Cape Town to pursue a doctoral research project aimed at improving traceability tools and practice.



## 1.4 PURPOSE, SCOPE & GOALS OF RESEARCH

The basic purpose at the start of the research project in 2004 was as follows

1. *To gain an understanding of the current state of play of traceability and the factors that influence traceability in development projects*
2. *In order to formulate a general solution framework that can accommodate these factors*

The purpose of this dissertation is to expand current knowledge of software traceability by examining the current state of the art of traceability practice in small, medium and large software organisations. The emphasis is on software project teams as the unit of analysis. For the purposes of this dissertation, a software development team is defined as the group

[3] C-MGW (Cello-Media Gateway). This is a propriety 3G IP router which is part of the Ericsson's 3G product line

of individuals who are directly responsible for the development and maintenance of a software product. In particular, this dissertation focuses on software traceability in the evolution of the Third Generation (3G) Operation Systems Support (OSS) telecommunication systems being developed by Ericsson's.

On analysis of the literature a gap emerged; a lack of empirical studies on traceability especially in the 21st Century. Therefore, our initial intention, when we started this project in 2004, was first to build on our practical experience at Ericsson in order to gain a better understanding of the factors that influence traceability and to evaluate the changes that took place over a four year period. Our contribution to the research community includes a description of the processes, the tools, the attitudes that all roles involved in the software development process have towards traceability and the challenges that Ericsson faced. To broaden the sample size we carry out a survey of small and medium sized organisations, also investigating the state of the art of traceability. In all we gather 83 samples of data, from nineteen organisations in both Ireland and South Africa. Using the case study, the survey and relevant literature as the main inputs we design, test and validate a Traceability Framework

It should be noted, however, that the scope of this dissertation is to focus on the real problems or challenges that are faced by the practitioners. Therefore, decisions were made that certain aspects of traceability are outside the scope of this project. For example, due to the complexity of traceability between design elements and code, which is worthy of a research project in its own, we do not address design traceability in this study. Furthermore, an important aspect of the scope of any research effort is that it must abide by the constraints imposed upon the researcher; such as restrictions on the publication of information protected by confidentiality agreements, or corporate sensitive information that would give considerable advantage to competitors. Thus, this research effort is limited in scope to the high level organisational factors that influence traceability rather than addressing some of the micro level applications of traceability within Ericsson.

With this background, the overall goals of our research are as follows:

- To carry out an empirical study that identifies the factors that influence traceability in small, medium and large organisations and describe the key challenges that these organisations face.

- To create *formal models*, using the latest modelling techniques, of traceability concepts and processes in a projects' lifecycle, which attempt to add benefit to both small and large organisations;

- To devise a new formalised approach for describing aspects of traceability that promotes better communication, reusability and usability by all resources in the software product development lifecycle.

- To design, implement, test and validate an integrated Traceability Framework that formulates a tailored solution of best practices in a scientific manner.

## 1.5 BASIC CONCEPT

### 1.5.1 Traceability Traverses the Entire Product Development Lifecycle

*"Traceability is about managing all aspects of a product so that the product satisfies the needs of the paying customer"*

-Eoghan Lynch, Requirement Manager OSS-RC R5, Ericsson

In Figure 1-1, *Traceability Across the Product Development Lifecycle*, we illustrate a simplified process diagram of the development phases of a telecom product in Ericsson. Software product development generates and maintains a wide range of artefacts: requirements documentation, design specifications and models, and test scenarios; all of which impact the system under development. Gotel and Finkelstein describe the practice of requirement traceability as "The ability to describe and follow the life of a requirement, in both a forward and backward direction, i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases". (Gotel and Finkelstein, 1994b)

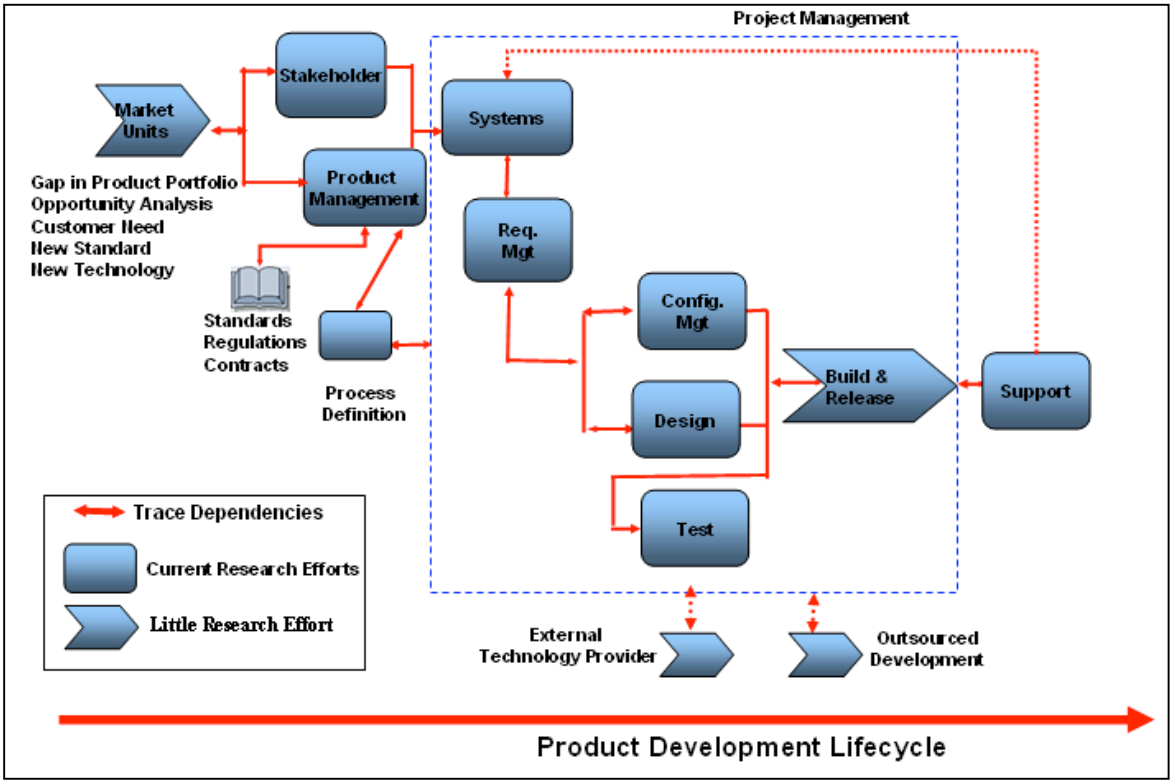



**Figure 1-1 Traceability across the product development lifecycle**

Spence and Probasco refer to modelling elements in their definition of traceability, saying "it is an explicit tracing of requirements to other requirements, models, test requirements, and other traceability items such as design and user documentation". (Spence and Probasco, 1998) Edwards and Howell define traceability as a technique used to "provide a relationship between the requirements, the design, and the final implementation of the system". (Edwards and Howell, 1991)

However, requirements traceability encompasses a whole lot more. It is an important means to facilitate *communication* among the success-critical stakeholders, it eases the determination of the *impact of changes*, it *preserves knowledge* and *dependencies* created during the design process, which *assures product quality*, preventing *misunderstandings* in the process. Egyed states that: "Requirement traceability is especially important for analyzing the impact of new requirements or changes to existing ones" (Egyed and Grunbacher, 2002)

We believe, however, that the term "requirement traceability" is misleading to many, because the term alludes that this is only a requirement related practice. During our empirical study we observed that in some cases, software engineers did not understand that the requirement traceability encompasses traceability in the entire product development lifecycle, including support and maintenance. Therefore in the context of this study we use the term software traceability to mean the practice of tracing all the traceability items that impact the successful delivery and support of a high quality software product. A traceability item is "[a]ny textual, or model item, which needs to be explicitly traced from another textual, or model item, in order to keep track of the dependencies between them." (Spence and Probasco, 1998)

## 1.6 DESIGN AND TEST PHILOSOPHY FOR SOLUTION FRAMEWORK

### 1.6.1 BASIC PRINCIPLES

The purpose of this section is to explain what we did: conduct research in the field; build models; create an experimental framework; and test our hypothesis in the field in order to demonstrate that each component satisfies the requirements in a coherent scientific and engineering manner. While the fields of science and engineering have well defined research paradigms, the fundamental principles of software engineering are still open to debate. There is, however, a general consensus that all new theory should be supported by empirical results. Generally speaking, software engineering researchers seek better ways to develop and evaluate the creation of better quality software. Software engineering research can only be considered sound when it is conducted using a valid *experimental framework*. Like any experimental paradigm this framework requires an experimental design, observations, data collection and validation on the domain being studied.

One of our early research goals was to establish a scientific and an engineering basis to study traceability. Within this framework of scientific and engineering experimental practice, models help us to understand the problems, carry out experiments, analyze and evaluate alternative tactics and strategies, with which we can refine and tailor the models for future research. A prime goal of this endeavour was to create a model-based experimental framework in order to enable future researchers to explore better ways of practicing traceability.

Traceability is both complex to understand and complex to implement, as it concerns itself with keeping track of traceability items throughout the entire product development lifecycle. It is complex because as a product develops from an idea, to a requirement specification, then a design specification, to functional drawings, then algorithms, then code, and finally hardware and a working system, it takes many different forms. At each stage, from requirements engineering to design to test, different kinds of people are involved in the process. These software engineering roles have different jobs to do and use different terms to describe traceability. Here, we argue that one of the reasons

traceability is such a difficult subject to understand is the lack of models, both conceptual models that describe the main concepts and semantics, and process models that describe how to implements these concepts. Following this logic, we observe existing traceability solutions, propose models, then measure and analyze the model's performance in a real-world context, before validating our hypothesis that the models and their experimental framework can be used as a tool or facility for further research work and educational work.

### 1.6.2 OBSERVING THE PROBLEMS IN THE FIELD

*"When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind. It may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science."*

-Lord Kelvin (1824-1904)

Our initial review of the literature quickly led us to the view that there was a lack of understanding of the challenges that practitioners face in the field, and the fact that the causes of these challenges could only be understood by an indepth study in the field. While academic research communities continue to focus their efforts on proposed new traceability techniques there is still a large gap between the theory and the realities encountered by the practitioners. It became apparent that traceability research is beset and held back by the following:

**1) Lack of Empirical Data:**

Considering the amount of new research being undertaken in traceability we still have very little evidence that gives us an understanding of the current status of the problems that organisations still face today or the factors that influence traceability both from an organisational and human perspective. Furthermore, there are few, if any, studies that track the traceability practices and the changes that take place in particular organisations over a number of years.

**2) Lack of Empirical Validation:**

The fundamentals of any science lie in its ability to prove or refute theory through observation. Research into traceability is no exception to this rule; where many new theories are not being supported by empirical evaluations.

These observations motivated our decision to carry out a case study and a wider industrial survey to gain a better understanding of current traceability practices in the field. We completed an industrial survey between 2004 and 2005. Some 19 organisations took part in the survey using structured interviews and questionnaires to capture the data. The problems that these organisations face include:

- *Poorly defined traceability processes:* New agile processes are now so compact that important practices like traceability are nearly unmentioned. During the survey we analysed this problem further. As illustrated in Figure 1-2, *Sample results from Survey*, only 52% of respondents believed that requirement engineering was sufficiently defined in their organisation. However, more worryingly, 18% believed that traceability was defined in their processes.

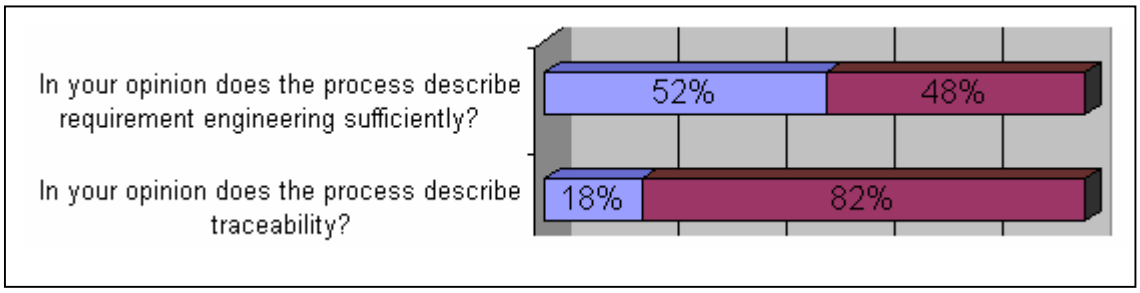



**Figure 1-2 Sample Results from Survey**

- *Poor communication:* Caused by poor processes and complex organisational structures leads to communication problems. These problems can effect change management, impact analysis and project management activities leading to communication problems in all matters related to traceability.

- *Lack of common terminology* caused by the lack of a widely accepted traceability *body of knowledge* which consistently describes the main concepts of traceability. (Hayes et al., 2006)

- *Poor tooling strategies:* In many cases smaller organisations don't use any formal tool, but use informal approaches, for example spreadsheets to implement traceability.

- *Poor education or training: In the aftermath of the downturn of the software economy, repressed* IT budgets means less money for training and education leading to competence shortfalls and a lack of understanding of the core traceability principles. *(Hayes et al., 2006)* Furthermore, in this study we investigate the link between university education and poor attitudes towards traceability. While requirement engineering is now apart of most IT related courses, many of those who participated in this study believed that further improvements are needed if practices like traceability are to be taken seriously.

- *Poor Capture of Experience or Knowledge:* Due to the speed of the deliveries projects don't have time to reflect, share information or transfer knowledge between resources in the development lifecycle.

Focusing specifically on the problems we set about defining a solution that satisfied the needs of the user community. As shown in Figure 1-3 below, *Traceability Patterns: Problem Space to Solution Framework*, we mapped the identified problems to possible solutions which led to the design, test and validation of the Traceability Framework that we propose in this study.

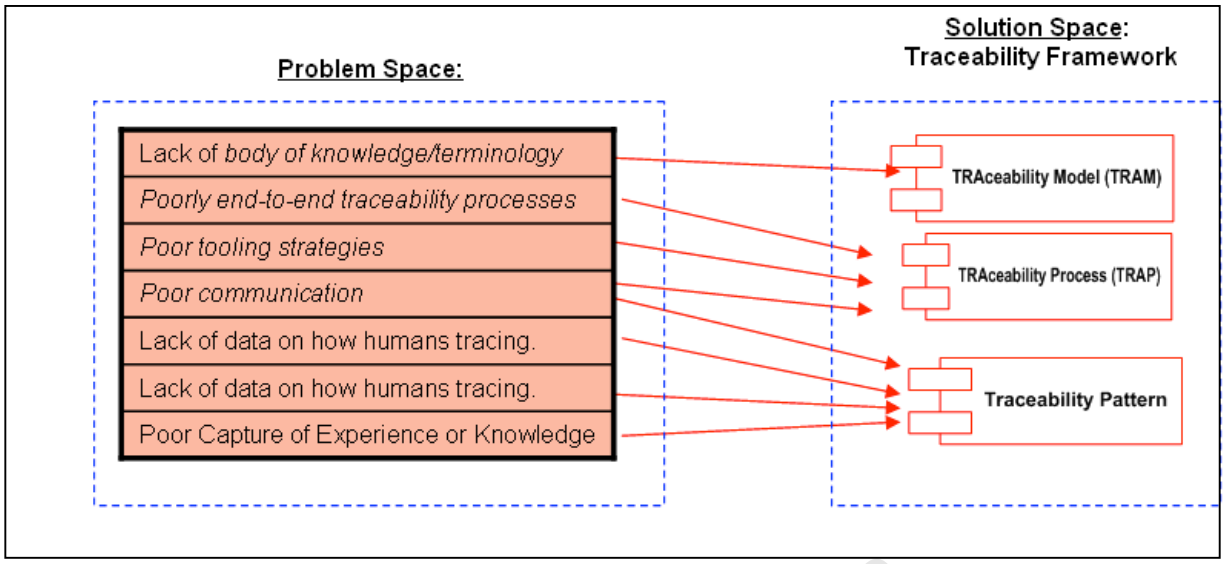



**Figure 1-3 Traceability Patterns: Problem Space to Solution Framework**

### 1.6.3 MODELLING & DESIGN PHILOSOPHY

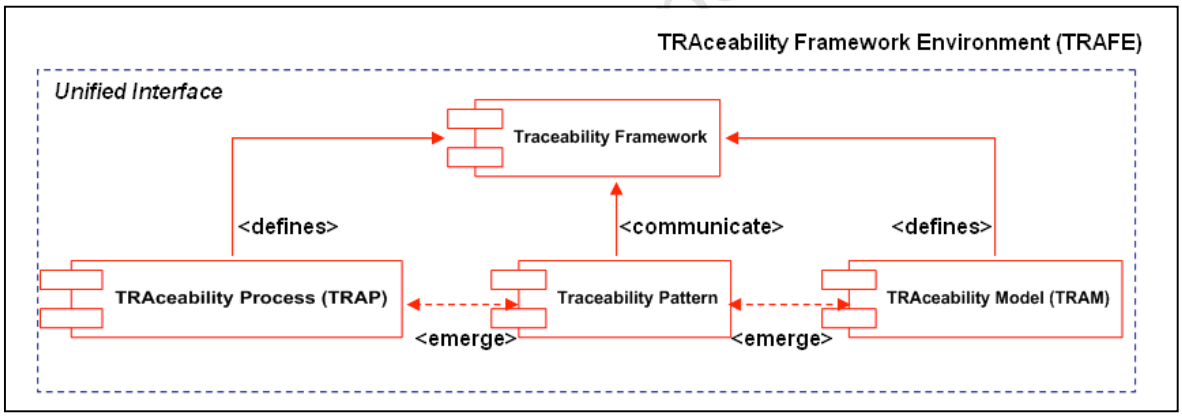



**Figure 1-4 The Main Components in Our Proposed Solution Framework**

As shown in Figure 1-4 above, *"The Main Components in Our Proposed Solution Framework"*, our proposed framework consists of three main components; *TRA*ceability *P*rocess (*TRAP*), a *TRA*ceability *M*odel (TRAM) and *traceability patterns*. The TRAP has the fundamental objective of describing and modelling the traceability development process and its workflows, plus all entities necessary for keeping track and managing the process workflows. The TRAP describes the development phases, the software engineering roles, their responsibilities, the tooling elements and any activities that are needed to implement traceability. Moreover, the TRAP contains workflows conveying the development time as a sequence, with the traceability best practices and traceability guidelines taken from literature incorporated in the models, whilst using traceability patterns as a mechanism for simplifying, understanding and communicating all practices related to the implementation of the traceability process. Our approach focuses on the implementation of traceability in the context of a specific product line in Ericsson's. We follow a layered approach, with the topmost layer being domain and technology independent suitable for future tailoring by

practitioners and improvements by academicians. We validate the TRAP component in the context that it was designed, the Ericsson situation, while using the latest modelling verification techniques.

The primary objective of the TRAM component is to describe the concepts and semantics of traceability. TRAM is a set of models that describes the fundamental principles of traceability, for instance traceability items, types, the trace types and the trace relationships at a level of abstraction that can also be used by both practitioners and researchers. Similarly to TRAP, we follow a layered approach with the topmost layers describing the most abstract and basic components; the traceability semantics independent of any domain or technology. The lower layers assemble these components into instantiations in the context of the Ericsson domain. We use similar validation techniques to TRAM with the results being made available for future work.

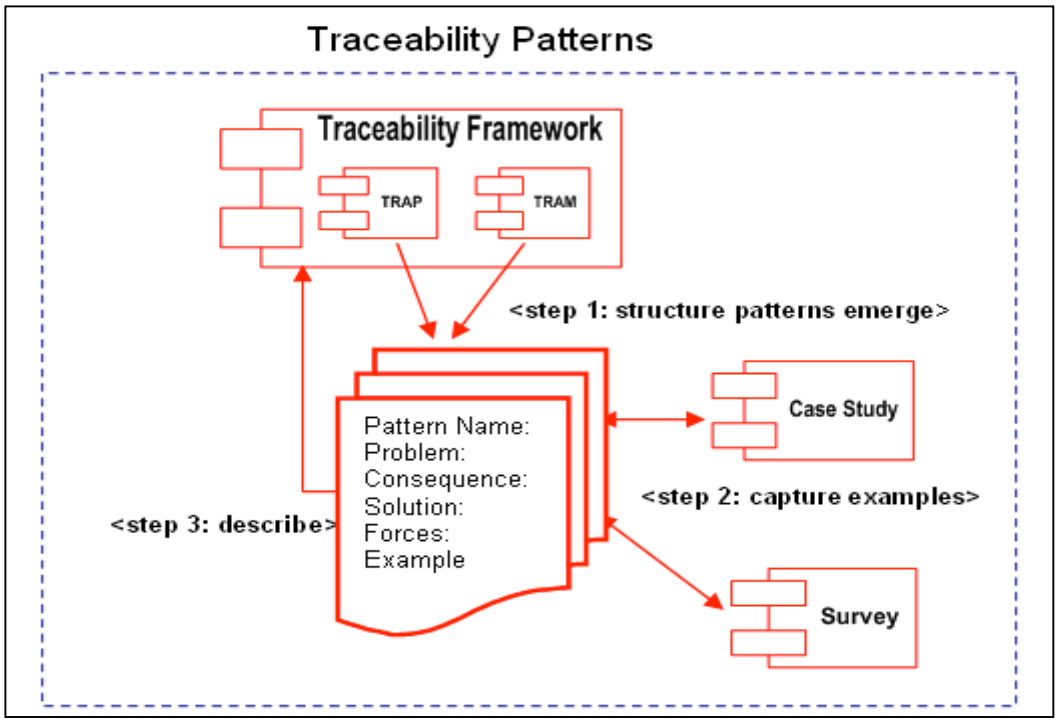



**Figure 1-5 Traceability Patterns**

During the design and model building phase of TRAM and TRAP, the author noticed that distinctive patterns began to emerge, which resulted in recurring model structures. By giving these patterns names we could use these names to refer to the structure or experience. In other words these patterns create a shared language for communicating experience and insight. These experiences from our modelling efforts provided a compact way of referring to a set of decisions and designs while suppressing the details not relevant at a given level of abstraction. By 2006, other research was being carried out on traceability patterns, however, these patterns focused on traceability in software design activities and the enabling of reusable class mechanisms which were applied to a goal-oriented framework.

Using Christopher Alexander's instructive definition of patterns that, "[e]ach pattern describes a problem that occurs over and over in our environment and then describes the core of the solution to that problem in such a way that you can use this solution a millions

times over without ever doing it the same way twice," (Alexander, 1979) the next step was to create pre-defined forms or templates for capturing these experiences. Once we had a template in place, we also realised that patterns were an effective way of describing traceability in practical settings. They are a structured abstraction that many users can understand, that integrate visual models and structured text as an effective mechanism to communicate and capture knowledge. Their real benefit became apparent during the case study and industrial survey. When presenting our work to different experts in the field we used a template to communicate the following: the problem we were trying to solve, the context in which it was set, the consequence of these problems and our proposed solution. By distributing the patterns to the user community we had an effective, structured way of eliciting experiences.

### 1.6.4 FRAMEWORK DESIGN PHILOSOPHY

The main design questions that were asked were as follows:

- Does the solution framework satisfy the problems identified during the case study and industrial survey?
- Does the framework provide benefits, or in other words is it usable by practitioners and the research communities?
- Does this framework describe all the main semantics and concepts of traceability in a simplified approach, promoting better communication, reusability and transfer of knowledge?
- Does this framework capture all the software engineering roles, along with their respective activities and responsibilities, in a time sequence manner?
- Are the models based on a foundation of acceptable standards?
- Is the framework both reusable and adaptable for real world scenarios and future work by research communities?
- Can we validate the framework by applying it in the context of laboratory trials and real-world scenarios?

In summary the main design considerations which we later explore as the validation criteria are; usability, understandability, completeness, correctness, reusability, extensibility and applicability.

### 1.6.5 Underlying Technologies

The Object Management Group is a consortium consisting of a broad range of research institutions, software companies and government organizations, whose aim is to promote the adoption of standards for managing distributed objects. With the creation of the OMG in 1989, the exploitation of modelling technologies has been revolutionized. The OMG promotes a concrete architecture for any Model Driven Development (MDD). The core elements of this architecture are the Unified Modelling Language (UML) which is the foundation for the OMG's Model Driven Architecture (MDA).

The MDA provides us with a conceptual framework and a set of standards to express models and model relationships in a platform independent manner. In other words, the MDA provides us with a solid structure of supporting standards to describe the

Traceability Framework divorced from the technologies that implement it. We also use the OMG's Software Process Engineering Metamodel (SPEM) specification in the design of TRAP. SPEM is object-oriented specification which describes how to model a software process.

## 1.7 OUR APPROACH

In Figure 1-6 below, *Overview of Our Approach* we illustrate our research roadmap, and how the research components fit together. The approach described below briefly outlines the main activities at a high level, their outputs and illustrates the scale of the project undertaken. A more detailed account of what we did at each of the phases follows in the remainder of this section.

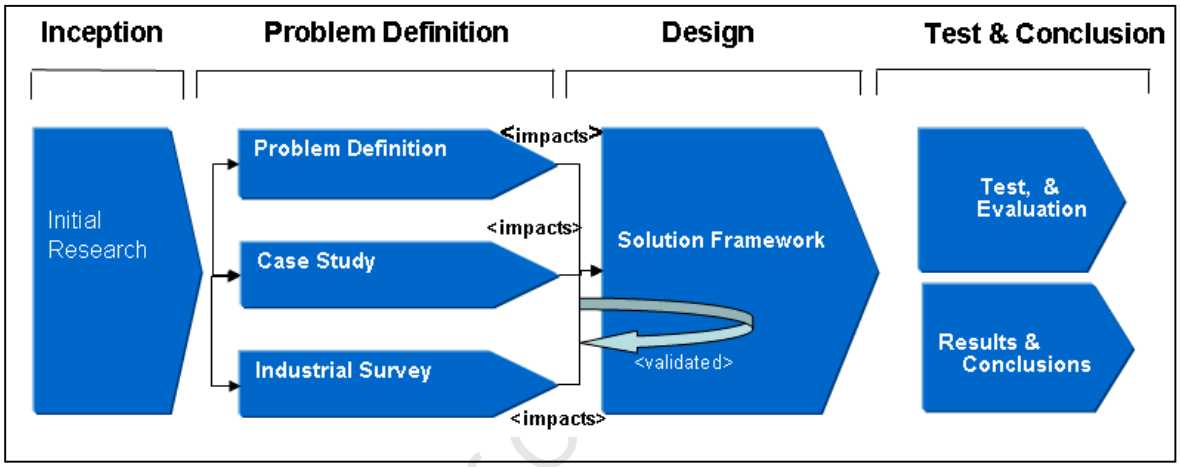



**Figure 1-6 Overview of Our Approach**

As described in Section 1.3 the author commenced this project with a good working knowledge of the practical aspects of traceability but we needed to broaden our knowledge by reviewing all relevant literature. First, there was a review of all papers from the 1990s which gave us an insight into the traceability research trends, the new techniques proposed and the empirical studies undertaken which provided us with a state of the art of traceability at the start of the 21st Century and also the methods they employed to extract this data.

By the start of the 21st century a number of exciting initiatives took place. In 2002, the first of a number of traceability workshops called the Traceability in Emerging Forms of Software Engineering (TEFSE 2002, 2003, 2005), took place in conjunction with the ACM Automated Software Engineering Conference. These events drew in a broad range of participants, both practitioners and researchers. The workshops were held to: broaden awareness within the software engineering community of the potential for the application of traceability; facilitate the exchange of ideas and interaction between international researchers; to define open research problems faced in realizing usable approaches for traceability; and to create a foundation for future research on traceability. These three workshops provided the author with a great deal of current insights into the traceability challenges being experienced in the wider world, and a valuable forum for exposing and exchanging ideas.

Then, in early 2004, as part of the process of ongoing Problem Definition, Ericsson's agreed to partake in an extensive Case Study. With great enthusiasm we incorporated a four year Case Study into our plan. Not only would Ericsson provide us with an environment to understand the problems and solutions that they face it also provided us

with the perfect test environment for our solution. The fact that we understood the background to their requirement engineering approaches, their corporate climate, their organisation structures, their product structure, their telecom technologies, and their methods and tools with many essential contacts for carrying out a case study, the criteria for choosing Ericsson were simple. As soon as the collaboration had been agreed, we set about defining clear objectives which were; to document the traceability environment, the resources involved, the tools being used, the processes, the product structures, the corporate strategies, the attitudes and also how they evolved or changed over a fixed duration of time. In so doing, we were in fact gaining an understanding of the factors that influence traceability in a large organisation developing enterprise systems. Furthermore the initial findings provided us with the problems or design considerations that we needed to overcome in our proposed traceability framework.

However, as already stated this case study only provided us with a single source of data and we were concerned that the results obtained may not represent the state of the art of traceability across the entire software industry. For example, it was obvious that special factors were at work, like the complexities of the telecom standards, the fact that the development was dispersed across many geographical boarders that the product families were unusually complex due to new systems being built on top of old complex systems and so on. We evaluated that further empirical data was required, particularly from smaller firms.

In order to acquire data from small and medium size firms, in 2005, the author initiated an industrial survey utilising a newly formed consortium of software engineers that met once a month at the University of Cape Town as the sample source of this study. The Software Process Improvement Network (SPIN) was loosely amalgamated to the Carnegie Mellon, Software Engineering Institute (SEI) which had a primary objective of knowledge sharing on all aspects of software engineering from a variety of industrial contexts in South Africa. On analysis of the attendees it became clear that this group were mainly from small and medium sized organisations, solving our problem that arose from only having a single source of data. The primary objective of this survey was to gain a better insight to the actual attitudes, practices, processes and tools being used by smaller development organisations.



However, we were still concerned that the data from South Africa, may not give us a true reflection of the state of the art because of its status as a developing country. Furthermore, the author from previous experiences had a large professional network in Ireland, which would provide another source of data. It was however, decided early on into our survey design, that the objective of this study was not compare the practices in a developing country against the practices in a country with one of the most mature software industries in the world, but rather to gather as much data as possible on the state of the art. In Ireland, the survey was completed between December 2004 and August 2005. Overall, 19 companies took part in the survey, 12 from South Africa and seven from Ireland. One of our goals was to gain access to as many different software engineering roles and as we will see in the Chapter 5, the survey chapter this was achieved.

In 2004, as part of his initial efforts at formulating a solution, the author also began work on creating a semantic and process models. We presented our initial efforts with TRAM and TRAP at the TEFSE 2005 (Kelleher, 2005b) before presenting the traceability patterns at a dedicated traceability workshop in conjunction with Object Management Groups (OMG's) European Conference on Model Driven Architectures. (Kelleher, 2006a)

The author published two further papers throughout this study, the first describing the evaluation of TRAP against the capabilities of the traceability workflows in the Rational Unified Process (Kelleher, 2005a) and the second on utilising use cases for requirement and traceability modelling. (Kelleher, 2006b)

### 1.7.5 EXPLORATORY, TEST & VALIDATION PHILOSOPHY

This leads us to our basic hypothesis of the solution framework:

*"On the basis of data gathered from industrial sources and previous research efforts, traceability practices are in need of a structured, systematic and disciplined rule-based modelling approach to overcome the problems being encountered in the field"*

*And that "the package of the TRAM and TRAP models, plus the patterns provide a flexible basic package, easily adaptable to a wide range of users, with potential to overcome many of the problems in the field."*

This hypothesis is further described by the following statements:

- The proposed framework builds on experiences from a variety of sources and assists us in gaining a deeper understanding of the field of traceability.
- The proposed solution is an experimental framework that builds on previous research efforts to provide a platform for future developments in the field of traceability.
- The proposed framework incorporates the problems that were identified during explorative studies undertaken in an industrial context.

To prove the stated hypothesis, a traceability solution framework based on the OMG's standards and specifications was designed, implemented, and tested. The aspects that were tested and analyzed include:

- that the design and implementation are effective in meeting real users' needs;
- that the models are tested against real data captured using industrial expertise where possible;
- that the component framework conforms to the underlying standards, is easily understood and extensible;
- that most traceability concepts can be modelled within this framework; and
- that most traceability practices can be modelled within this framework.

## 1.8 RESEARCH CONTRIBUTIONS

As illustrated in Figure 1-7, *Research Contribution*, this research makes the following contributions to the field of traceability:

- A case study identifying the factors that influence traceability and the challenges faced in a large telecom corporation developing enterprise systems.
- An analysis of the changes that occurred over a four year period and the impact these changes had on the implementation of traceability. .

- An industrial survey study of small and medium enterprises also identifying the factors that influence traceability.

- The design of a component based traceability framework that attempts to solve the problems encountered during the empirical studies.

- An introduction to the novel concept of traceability patterns.

- An evaluation of the framework design to show that it meets its stated requirements.

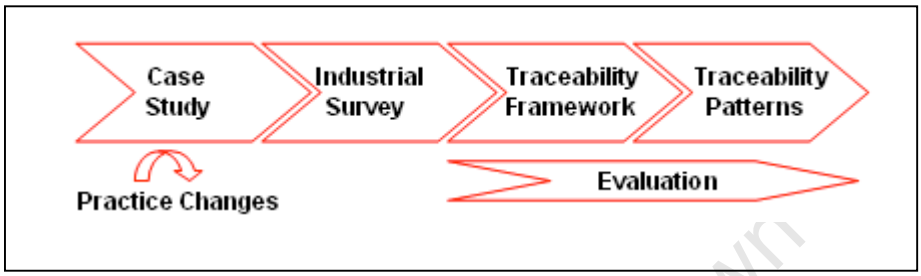



**Figure 1-7 Research Contribution**



## 1.9 OUTLINE OF REPORT

| Chapter | Title & Contents |
|---|---|
| Chapter 1 *Introduction* | Outlines the motivation, problem space, and scope of the research. |
| Chapter 2 *Review of State of the Art of Traceability* | Discusses historical and current research in traceability. We highlight the research or contributions that had the biggest influences on this project. |
| Chapter 3 *Research Method* | Outlines the research method followed and how it all fits together. |
| Chapter 4 *Case Study: Factors that Influence Traceability in Ericsson's OSS Domain (2004)* | Presents the background, context and initial observations on the factors that influence traceability in the development of a telecom system in Ericsson (2004) |
| Chapter 5 *State of the Art Industrial Survey* | Presents the background, context and results from an industrial survey carried out in 2005. |
| Chapter 6 *Introduction to Traceability Solution Framework* | Presents the components of the proposed solution framework, the philosophy and the design considerations. This introductory chapter introduces the components that we further develop in Chapters 7, 8 and 9. |
| Chapter 7 *TRAceability Model (TRAM)* | Presents the TRAceability Model (TRAM) component of our proposed traceability solution, the high level models and the model instantiations |

| | |
|---|---|
| | and the design philosophies that influenced our work. |
| Chapter 8 *TRAceability Process (TRAP)* | Presents the TRAceability Process (TRAP) component of our proposed traceability solution, the high level models, the project instantiations and the design philosophies that influenced our work. |
| Chapter 9: *Traceability Patterns: A Pattern Approach to the Formal Specification of Traceability* | Introduces the background, context, principles of the novel traceability patterns. We categorise the traceability patterns and conclude with examples of these patterns discovered during the case study, industrial survey or those that emerged from the modelling activities. |
| Chapter 10: *Case Study Revisited: Major Changes and how they Effected Traceability in OSS Domain* | The conclusion of the case study, including the changes that took place over a four year period and how these influenced the practice of traceability. |
| Chapter 11 *Traceability Framework Validation & Assessment.* | The evaluation of the traceability framework. We describe the laboratory and field trials and the evaluation frameworks that we used to assess the capabilities of the solution framework that we propose. |
| Chapter 12 *Conclusions & Future Work* | Summarises the findings, the lessons learned and presents recommendations for future work. |

**Table 1-1 Outline of Report**

# Chapter 2 REVIEW OF TRACEABILITY STATE OF THE ART

## 2.1 INTRODUCTION

Before creating a solution framework that attempts to solve a set of problems we must understand what these problems are and the context that they occur. Understanding trends in traceability research and identifying relevant research is an essential aspect of any research project. The aim of this chapter is to describe all related research efforts and how it influenced this study.

We describe the history of traceability from the 60s right through to present day. We describe the pivotal publications, the research communities, the workshops, the research projects, the technical reports, the standards and the processes that influence this work.

### 2.1.1 History of Requirement Engineering

*"...one of the difficulties about computing science at the moment is that it can't demonstrate any of the things that it has in mind; it can't demonstrate to the software engineering people on a sufficiently large scale that what it is doing is of interest or importance to them."*

-Christopher Strachey, NATO Conference, Rome 1969

This famous statement was made the year after the term "software engineering" was provocatively coined at the first Software Engineering Conference convened by the NATO Science Committee in 1968. It is an interesting coincidence that at the very same conference that software engineering was first used that the catchphrase "software crisis" was also coined by F.L. Bauer to describe the state of the software industry. Over the decade of the '60s theoretical computer science achieved standing as a discipline recognized by both the mathematical and the computing communities. During this time new higher level programming languages like FORTRAN, COBAL and ALGOL emerged for the computer programmers who worked in what were then called the "machine rooms". In reality Bauer was right, there was a software crisis. The lack of project management practices, the lack of tried and tested processes, best practices and most importantly the lack of knowledge was causing poor requirement management, poor programming, poor testing which resulted in poor development. It in this context that Christopher Strachey made the above statement and the birth of software requirement engineering had well was truly begun.

The development and use of requirements tracing techniques originated in the early 1970s. Traceability was originally developed as an approach to ensure that the product delivered to the customer satisfied the original requirements. Traditional traceability systems often used paper logbooks and required production operators to document the requirements, the development components and the key equipment attributes. Keeping handwritten or typed documents up to date with the changing system requirements was not a top priority during this mathematical era of software development. However, as

computers became more cost effective in the 70's and database technology evolved new developments in managing traceability items started to take place

During the 80s new definitions of requirement engineering and traceability were appearing. The most relevant definition of requirement engineering was by Zave who stated that: "[r]equirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the *relationship* of these factors to precise specifications of software behavior, and to their evolution over time and across software families."(Zave, 1983)

In the 60's, the entire computer industry was in crisis, while in the 70s, the computer world focused on aspects of Systems Engineering. However, it was during the 80s that the first "modern" definitions of requirement engineering appeared.

## 2.2 LITERATURE REVIEW

### 2.2.1 Requirement Traceability Research 1990-2000

Research in the 1990s created new definitions and terminology for traceability, described the importance of traceability and identified the problems implementing traceability using case studies. They defined traceability items,(Spence and Probasco, 1998) they reviewed the available traceability tools, (Jarke, 1998) they outlined the steps involved in implementing traceability, while others established the factors influencing requirement traceability. (Ramesh, 1998)

In 1991, Edwards and Howell defined traceability as a technique used to "provide a relationship between the requirements, the design, and the final implementation of the system". These relationships allow designers to show that the design meets the requirements and help early recognition of those requirements not satisfied by the design. (Edwards and Howell, 1991)

In our opinion, however, it was the efforts of Gotel and Finkelstein that had the biggest impact on the traceability community when they published a number of key papers that are still widely referenced today.(Gotel and Finkelstein, 1994b, Gotel and Finkelstein, 1994a) In 1994 they presented their work at the first IEEE Requirement Engineering conference describing traceability as follows:

*"Requirements traceability refers to the ability to describe and follow the life of a requirement, in both forwards and backwards direction (i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases.)"*

Today, we understand that this definition in fact only describes part of the complexities of the traceability picture. While Gotel emphasizes the life cycle aspect of traceability, she imputes that the requirements are the only traceability items that need to be traced when in fact any definition of traceability must include tracing to the change requests that impact the requirements, model elements, design elements, test elements and any item that affects the system under development. In another study, Gotel presents the result of an empiric study which describes the identification and understanding of the problems and practices associated with requirements traceability. They reported that the majority of the problems attributed to poor requirements traceability are due to inadequate pre-Requirement Specification (RS) traceability. She attributes this to poor pre-RS traceability.

She continues to divide traditional requirement engineering into pre-requirement and post-requirement traceability with the description:

- Pre-RS Traceability: Referring to those aspects of a requirements life prior to inclusion in the Requirement Specification.

- Post-RS Traceability: Referring to those aspects of a requirements life after inclusion in Requirement Specification.

Bidirectional links or (Ecklund Jr et al., 1996) "downstream" links allow team members to see the scope of a change while "upstream" links allow the team to see why a particular decision was made, based on the high-level requirements that were suggested. Research into link types, for instance bidirectional links opened up new avenues of research into impact analysis brought about by changing customer or system requirements.

Similarly, Ian Sommerville, described three types of traceability relationships that he believed should be maintained. Source traceability links are between the requirements and the stakeholders who propose the requirements. Requirement traceability links between dependent requirements within the requirement document. Design traceability links between the requirements and the design elements. (Sommerville and Sawyer, 1997)

However, by the late 90s researchers were realising that there was no silver bullet for implementing traceability. According to Scott Ambler "it is rare to find a software project team that can that can honestly claim full requirements traceability throughout a project, especially if the team uses object orientated technology" (Ambler, 1999) In fact, the new object paradigm was causing many problems in the implementation of traceability, which we will discuss during our review of requirement engineering in Ericsson (Chapter 4). Other researchers observed that by neglecting traceability or by capturing insufficient or unstructured traces, can lead to a decrease in system quality and therefore, increasing project costs. It resulted in a loss of knowledge if individuals left the project, leading to wrong decisions, misunderstandings, and miscommunications (Dömges and Pohl, 1998)

It was, however, Ramesh's paper in 1998, "Factors influencing requirement traceability practices" that had a major influence on this study.(Ramesh, 1998) His research provided us with a conceptual framework which we used as the foundation for carrying out investigations into the *factors that influence traceability* during the industrial survey (Chapter 5) and the Ericsson case study (Chapters 4 and 10). Ramesh argues that institutional contexts and the strategic conduct of organizations influence each other over time. He subdivided institutional contexts into organisational contexts, environmental contexts and system development contexts. The organisational context is the organisations commitment to traceability, which is mandated using a corporate traceability strategy. Environmental context describes the technologies that support the use of traceability information. The systems development context encompasses the policies for both staffing and the use of standardised methodologies for implementing traceability across the system development lifecycle. In summary, Ramesh's key factors are the *corporate traceability strategy*, the tooling situation, plus the staff training and the process policies in place.

On the basis of his survey, Ramesh divided organisations into two distinct groups; *low-end* and *high-end* traceability users. As expected, low-end users view traceability simply as something forced upon them by the project sponsors. On the other hand, high-end users view traceability as an important component in the development of high quality systems. High-end users generally have a corporate traceability strategy, which not only describes the traceability problem but also the goals and objectives for carrying out

traceability tasks. Such organisations recognise that, in order to adopt and use traceability, one should develop methods, and either acquire or develop tools, and change system development policies with regard to staff allocation and training.

In conclusion, few would argue that during the 90s the field of traceability had ascended from an unknown practice to one that was now firmly accepted as an intricate part of the requirement engineering process. The next hurdle, however, to overcome was to move from acceptance to actual application in the user domain. This new "hot-topic" still lacked direction, collaboration and funding as we reached the end of the twentieth century, but this was all to change in the early days of the 21st Century.

### 2.2.2 Research Efforts in the 2000's

*"We find that on average only 54 percent, down from 67 percent in 2001, of the originally defined features of a project are delivered. Even more troubling is the realization that of those features that are delivered — a full 45 percent are NEVER used."*

*- Standish Group, 2002 "What Are Your Requirements?"(Standish_Group, 2002)* [4]

The economic slowdown in 2001 was almost unprecedented in the history of software development causing the rapid decline in investments into IT. Managers had never faced such a time of uncertain business and poor economic outlook. The aftermath of the dot-com implosion and the impact this had on IT budgets had long lasting impacts on the development of new technologies, processes and tools as organisations prepared for the worst. While one consequence of this economic slowdown was repressed IT budgets, it also caused organisations to restructure their product portfolios. The impact on traceability practices was the inevitable move from simpler requirement traceability matrixes to tracing between families of complex software-intensive products. This complexity combined with tighter markets, tough competition and repressed budgets had far reaching implications on the future direction of traceability research.

#### 2.2.2.1 ACM TEFSE (2002-2005)

In 2002, the first dedicated traceability workshop took place bringing researchers together from around the world to share their ideas. The Traceability in Emerging Forms of Software Engineering (TEFSE 2002, 2003, 2005) in conjunction with the ACM Automated Software Engineering Conference signalled the start of a new era of traceability research encouraging publications from academicians and practitioners alike.

These workshops addressed a multitude of issues ranging from modelling traceability to traceability in emerging forms of software engineering including production lines, frameworks and components. An addition to the consideration of the key factors impacting traceability, the workshops aimed to broaden awareness within the software engineering community of the potentials that applying traceability brought. It facilitated the exchange of ideas and interaction between international researchers, to define open research problems faced in realizing usable approaches for traceability and to create a foundation of materials for future research into traceability.



[4] In Section 12.3.1, Insights From Empirical Study, we describe the results from our empirical study in Ericsson which validates many of the findings from this Standish Group report.

In the remainder of this section we describe the papers that had the greatest influence on this research. At *TEFSE 2002*, the University of Newcastle presented a position paper defining their traceability approach for recording, analysing and tracing development and assessment artefacts.(Arkley et al., 2002) Over the coming years Arkley and Riddle continued to produce high quality empirical research. For example, they described the reasons why many aspects of requirement traceability problems still exist today has as much to do with process and human factors as it does to the tools and distributed team factors.(Arkley et al., 2006) They concluded this line of enquiry by describing how to overcome the traceability benefit problem and how to tailor traceability to meet the business needs. (Arkley and Riddle, 2005) Elena Perez Minana, also produced results from an empirical study undertaken in Philips Consumer Electronics (PCE) on the composability of the elements involved in requirement management specifications in the product family. (Pérez-Miñana et al., 2002)

In 2002, however, it was Letelier's contribution that had a major influence on our decision to commence development of the TRAM and the TRAP components. He describes the use of UML to create a common traceability framework suitable for UML based projects. Letelier presents a UML reference metamodel for requirements traceability, representing all the software development artefacts and traceability links among them. He further uses the UML extension mechanisms, for adapting UML for every traceability situation in any projects. (Letelier, 2002)

It was at TEFSE 2003 that DePaul University, described an approach for establishing traceability between certain types of Non Functional Requirements and code artifacts, by using design patterns as intermediary objects. This paper influenced our research efforts into traceability patterns, referring to some of the general principles described. For example, they state that the "use of design patterns as intermediary objects introduces the possibility of rule-based link generation in place of lexical-based generation…." (Cleland-Huang and Schmelzer, 2003)

At TEFSE 05, we initially presented our conceptual framework. [5] The proceedings of this workshop were sub-divided into a number of sections namely; early traceability concepts, traceability techniques, and the final section utilizing traceability links categories.

In essence these workshops were not only a perfect forum to present our conceptual ideas but also as a source of the major research efforts and the key researchers leading us to other conferences and publications. In many instances, the ideas and concepts presented at the TEFSE workshops described concepts and empirical studies that lead to major publications at prestigious conferences and journals in the coming years.

Overall, excellent research was being produced by the research communities. However, in our opinion there was a shortage of empirical evidence of the challenges faced by small, medium and large organisations encountered today. The lack of empirical data describing the current state of the art of traceability and a shortage of solid validation and results to support these new techniques were noticeable shortcomings from these workshops.



[5] It was at TEFSE 05 that we presented the initial paper a Reusable Traceability Framework.

#### 2.2.2.2 ECMDA Traceability Workshop

A second group of traceability workshops began in conjunction with the OMGs Model Driven Architecture community however the focus of this group was on traceability in model driven architectures. This was a forum for discussions on practical and theoretical issues associated with languages, mechanisms, tools, and processes, addressing the management and application of traceability in the model-driven development community at large. The goal was to open discussions on the aspects of change management of models, impact analysis of model modifications and other research in model driven engineering.

While many of the ECMDA publications influenced our work we will refer here to those that impacted our approach or solution framework. In 2005, Martin Gills, a fellow doctoral student from the University of Latvia presented a Survey of Traceability Models in IT projects. Using questionnaires and interviews he surveyed 6 IT companies and 32 projects to correlate his results. We referred to the method he used and the results he obtained in our project. (Gills, 2005). The objectives of his survey was to learn the general experience of projects in the selected group of IT companies; to learn the attitudes of project members towards the traceability; to capture the most typical traceability relationships between other traceability items and testing; to evaluate the amplitude of differences between traceability practices; to compare his results with similar research efforts; and to gather data that would act as a basis for his traceability tool, TraceIt. His results demonstrated that:

- Attitude to Traceability: 53.1% of projects practiced traceability while 21.9% paid no attention to traceability.

- Traceability tools: 18.8% used a tool for implementing traceability, while 37.5% used some form of traceability approach.

Other relevant papers included, Vanhooff et al addressed the automation of model transformations by inserting *semantically* rich transformation traceability links to better understanding of these transformations. He defined a UML transformation profile that allowed the addition of semantically rich traceability links.(Vanhooff and Berbers, 2005)

#### 2.2.2.3 Other Research Influences

A number of relevant postgraduate research projects influenced our work either because of the context that they were set, the methods that were followed or the results that they obtained. In 1994, Mikael Lindvall carried out research on traceability in object-oriented systems at Ericsson Radio Systems (Sweden). Lindvall described examples of traceability in object-oriented projects from 1992 until 1996.(Lindvall, 1994) Object orientation was a new paradigm in Ericsson's in the early 90s and his research methodology provided us with a foundation for our case study. Lindvall continued his research work to doctoral level with investigations into impact analysis in the requirement driven development at ERA.(Lindvall, 1997) While Lindvall's efforts created an authoritative body of knowledge on traceability in OO projects, its main influence on our project was that he proved that a case study based project in Ericsson was possible. The success of Mikael's work is reflected in the number of publications that he achieved in collaboration with his supervisor Sandahl over the coming years. (Lindvall et al., 2001, Sandahl et al., 1998))(Lindvall and Sandahl, 1996)

Another empirical case study was completed Ericsson's Eurolab in 1999. The studies primary objective was to improve requirements engineering practices in Ericsson.

They identified the root causes of requirement engineering in Ericsson as; an insufficient understanding of the customer requirements and the ambiguity that surrounded these requirements. Furthermore, they introduced a new experimental method into Ericsson, which was used by a number of projects and the results of its usage were presented. On analysis of this data they discovered that the biggest benefit of using their method was the positive impact it had on the requirement engineering culture within Ericsson and the advancements that the project team made in understanding ambiguous requirements. The results showed that poor communication between the internal stakeholders was a factor especially amongst those who did not use their method. (Jacobs and Eurolab, 1999).

Limon et al analyzed current traceability schemes, for instance link types, in order to obtain relevant features and identify overlaps and inconsistencies among the approaches. (Limón and Garbajosa, 2005) In the paper "Analyzing and Systematizing Current Traceability Schemas" the authors analyse several current traceability approaches, to identify overlaps and inconsistencies between them, and to select the best traceability practices. (Espinoza et al., 2006) Hu et al present a pattern- oriented approach for building a metamodel and defining the basic elements of a XML metamodel pattern. They further introduce pattern examples and define some rules on how to use these patterns.(Hu and Vollmar, 2001) This paper provided us with useful information during the creation of the traceability patterns.

Another influence was Hayes et al, who carried out research into building a traceability framework for comparing requirement tracing experiments. They present an experimental framework for evaluating requirements tracing and traceability studies. They describe common methods, metrics and measures and provide suggestions on how the field of tracing and traceability research may move to a more mature level. (Hayes et al, 2005)

It was with this analysis of the state of the art into traceability that laid the groundwork for our research effort. In this next section we investigate the research projects and the research communities that influenced this project.



#### 2.2.2.4 Research Since 2006

Research since 2006, has shown that inadequate traceability is an important contributing factor to software project failures and budget overruns (Domages et al, 2008). As a response, there was an outpouring of research and literature on the subject of traceability, in the early 2000s and many organizations have been striving to improve their traceability practices. These efforts have not been in vain. In 2006, The Standish Group updated their 1994 study [Standish, 2006], showing that only 19 percent of software projects failed outright with another 46 percent challenged by budget overruns. Although the importance of traceability seems to be well-accepted in the software engineering industry, research suggests that many organizations still do not understand the principles of traceability and are struggling with implementing traceability practices in the software development life cycle [Ramesh et al, 2006].

One of the major developments in the past three years is the development of traceability practices in the SOA life-cycle which differs from traceability in the software life-cycle. [Seedorf et al, 2009] Seedorf et al distinguish the following types of traceability in the SOA life-cycle:

1. Tracing business and SOA elements to their sources, i.e. the responsible internal and external stakeholders.

2. Tracing between business entities, in particular business processes and services. There is an *m* to *n* relationship between business processes and services. In addition, business processes in different organizations may be supported by a single service.
3. Tracing the life-cycle of a service from requirements, design, implementation, and test to run-time and change.
4. Tracing the dependencies between services in a service ecosystem. This differs from the usual black-box view taken by service-orientation.

Traceability has been universally accepted as being a key factor for the success of software development projects. However in recent years, the multitude of different, not well-integrated taxonomies, approaches and technologies impedes the application of traceability techniques in practice. In 2009, a group of German researchers presented a comprehensive view on traceability, pertaining to the whole software development process. Based on graph technology, they derive a seamless approach which combines all activities related to traceability information, namely definition, recording, identification, maintenance, retrieval, and utilization in one single conceptual framework. They validated their approach in the context of the ReDSeeDS-project aimed at requirements-based software reuse. (Schwarz, 2009)

## 2.3 RESEARCH PROJECTS

Traceability practices were studied in a number of projects in different contexts. For example, Alexander describes the experiences he gained using the DOORS traceability platform on a Railway Control System Project; a project to replace a control box on a commuter railway. (Alexander, 2002) In 2006 Arkley and Riddle describe how the development of a new traceability system by a company resulted in an increase in profitability as well as other benefits for the development engineer, project management and customer. (Arkley et al., 2006) They observed traceability practices in BAE SYSTEMS E&IS (Plymouth, UK) to get a better understanding of the Traceability Benefit Problem. (Arkley and Riddle, 2005) Their published work did not focus on the development of the traceability system but rather on the extrapolation of traceability information.

Andrea Zisman a well established and respected traceability researcher at London University received funding for a number of small traceability projects. In 2002, she received funding from Philips Research Labs for a project called *"Concerning automating generation of traceability relationships between early software lifecycle artefacts"*. In another project funded by Madihol University Thailand her research team investigated software traceability for Product Family Systems. This project was concerned with creating a traceability metamodel for product families with a supporting tool for automatic generation of traceability relations for software artefacts. In another project (2002-2006) on Software Traceability for Agent-Oriented Systems, their primary investigation was concerned with developing automatic generation of traceability relations for agent oriented software artefacts. The success of these projects is reflected in the number of publications Zisman et al, achieved in the past number of years. (Zisman et al., 2002) (Zisman et al., 2003b) (Spanoudakis et al., 2004) (Spanoudakis et al., 2003) (Zisman et al., 2003a)

One of the largest and most important research projects into traceability in Model Driven Development (MDD) is the Model Ware project. The project consists of a

consortium of 19 partners[6], including leaders of software-intensive industries, tool vendors, academia, and consultancy companies based in eight European countries. This project had a total budget of € 20,295,032, € 10,937,777 of which was funded by the European Commission.[7] The MODELWARE Project initiated the annual European conference on Model-Driven Architecture, with the support of the OMG. Two Traceability workshops were run in conjunction with European Conference on Model Driven Architectures (ECDMA) 2005 and 2006.

The SINTEF research group is the largest independent research group organisation in Scandinavia. In 2006 SINTEF presented a paper at the ECDMA traceability workshop with the title: *Towards a Generic Solution for Traceability in MDD*.(Walderhaug et al., 2006) This paper was the result of research carried out during the MODELWARE project. They proposed a traceability metamodel supported by guidelines and templates. Many of the SINTEF research team are currently working on the MODELPLEX (*MODELling solution for comPLEX software systems*) project, which is a continuation on from the now complete MODELWARE project. The projects objectives are to build an open solution for complex systems engineering based on model driven engineering techniques, which will be based on industrial use cases, ensuring successful adoption and improving quality and productivity. The outputs from MODELPLEX and their traceability research efforts are not yet available in the public domain. (ModelPLEX.org, 2007)

## 2.4 RESEARCH COMMUNITIES

### 2.4.1 Centre of Excellence for Traceability

One of the main initiatives that arose from the TEFSE workshops was the formation, in 2006, of a consortium of researchers, called the Centre of Excellence for Traceability. Its primary objective is to create a community of researchers and experts in the field of traceability with a goal of improving traceability practices and techniques. They promote research collaborations while building a body of knowledge on all aspects of traceability. The Centre of Excellence held a traceability collaboration workshop in August 4-5, 2006. The goal of the workshop was to bring researchers and practitioners in the area of traceability together for discussions on the state-of-the-art and to gain consensus on future traceability direction.

The main output from this workshop was the *Grand Challenge Report.*(Hayes et al., 2006). This technical report describes and categorizes the challenges faced while implementing traceability highlighting focus areas for future research. Figure 2-1, *Grand Challenges Taxonomy*, is a simplified version of the taxonomy released in accordance with the Grand Challenges Report. It illustrates the challenges faced and the relationships between these challenges. As the report did not appear until 2006, the biggest benefit it brought to our research was that it confirmed that we were in fact addressing contemporary



[6] MODELWARE consortium: Thales (France), IBM (UK and Israel), Schlumberger WesternGeco (Norway), France Telecom (France), Enabler Informática (Portugal), WM-Data (ex-Aprote) (Estonia), SOFTEAM (France), SINTEF (Norway), imbus AG (Germany), Adaptive Limited (United Kingdom), INRIA (France), ESI (Spain), Université Pierre et Marie Curie LIP6 (France), Universidad Politécnica de Madrid (Spain), University of York (United Kingdom), Fraunhofer FOKUS (Germany) and Zühlke (Germany).

[7] This project was funded as part of the "Information Society Technologies" Sixth Framework Programme (2002-2006)

issues or challenges. In this rest of this section we review the most relevant challenges to this report.

Firstly, the report describes that a traceability *body of knowledge* is required to develop effective educational material that can be used for traceability certification programs. The next aspect of the report addresses how to *support the evolution of traceability*. For example, the challenge of how to develop techniques that supports traceability across products, within a product line by maximizing reuse and providing traceability between various versions of the product line. The report continues to state that further work is needed to maximize reuse of traceability links and to develop link recovery techniques for textual artefacts and change management.

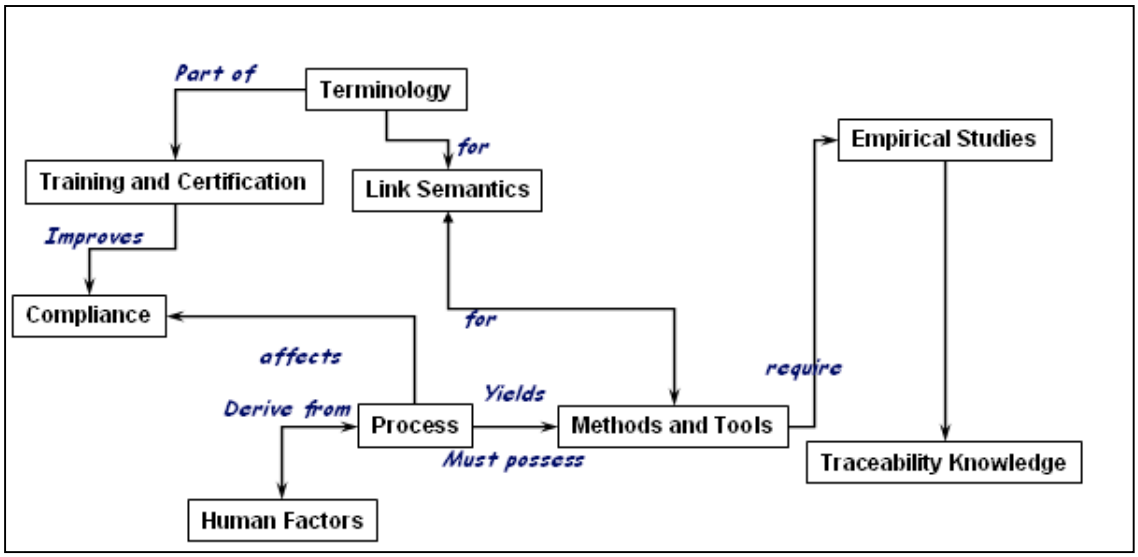



**Figure 2-1: Grand Challenges Taxonomy**

Furthermore, the report describes the demand for a new *meta-model* to represent the *semantics* on traceability links, supported with instantiated examples from specific domains. This semantic model should develop techniques and processes for determining the notion of domain specific models. They suggest that the semantic model should develop an infrastructure that supports experimentation and produce results based on industrial data.

In the *Human Factors* category the problems highlighted include the need for an understanding of humans in the product development life-cycle. It continues to say that without this data it is difficult to build useful traceability methods and tools or accurate stakeholder usage models. While traces bridge semantically different artefacts, these artefacts are created by different people who use different dialects. For example the lingo used by the product managers is different to the developers or system testers. As a result, users of traces often do not fully understand artefacts on the "other side" of the links. The major challenge is to develop methods to help humans overcome semantic barriers in tracing to artefacts of other stakeholders.

In the methods and tools section the main challenges are to build methods and tools with maximized levels of automation to support the entire trace life-cycle and develop methods for tracing non-functional requirements. End-to-end tracing is critical to the success of a project, but organizational boundaries make it difficult to achieve this due to differences in skill-sets, processes, terminology, and tools. The difficulties arise from the

need to define effective end-to-end processes with supporting tools for cross-boundary traceability such as tracing between outsourcing companies or tracing between different business units within a corporation.

In the process category the report states that "in order to generate and maintain quality and sound traceability information, an organizational process is required" However, traceability is often not included as an integral part of the development lifecycle. It continues to say that automated tracing can provide a cost-effective alternative to manual tracing, but practice shows that some data sets are difficult to trace when using automated methods due to inconsistencies in terminology, standards, terseness of requirements, lack of structure, heterogeneous formats.

The *Compliance* category discusses the multitude of standards which help ensure consistent and complete processes. The traceability community is knowledgeable about traceability techniques and processes, but has little influence over the traceability related content of software engineering process standards. The issue is how can this community influence the standards community and develop traceability standards.

Other problems outlined include the *need for empirical studies* to demonstrate the effectiveness of traceability methods. While researchers claim successes in new traceability methods and techniques, there are few benchmarks in place for comparative studies. It continues to describe the need for the identification of *successful case studies* in order to demonstrate the cost and technical effectiveness of the traceability techniques to the industry. The report concludes with the need for the identification of traceability users.

In summary, this report addresses many of the same issues that we had encountered both as a practitioner and during our literature review. In Figure 2-1 above, *Grand Challenges Taxonomy*, we illustrate a simplified representation of the grand challenges taxonomy, which was released in accordance with Grand Challenges Technical Report. This report reflects the shortcomings that have been discussed by the research community and it is undoubtedly the basis for future research in the coming years.



## 2.5 TRACEABILITY IN STANDARDS

In this section we review traceability as it is defined in a number of standards. The goal of this section is to illustrate that traceability is defined in a number of standards and assessment frameworks. In particular we review how traceability is mandated in the ISO 15504 and the Capability Maturity Model (CMM) and in Chapter 11, the validation chapter we discuss how we used both standards to assess aspects of our proposed solution framework.

### 2.5.1 Traceability in IEEE Standards

The Institute of Electrical and Electronics Engineers (IEEE) is a technical professional association, composed of engineers, scientists, and students. The IEEE fosters the development of standards that often become accepted national and international standards as well as publishing volumes of useful information on the software development process activities.

The IEEE Standard 1362 defines software traceability as the identification and documentation of derivation paths (upward) and allocation or flow down paths (downward) of work products in the work product hierarchy. Important kinds of traceability include: to

or from external sources to or from system requirements; to or from system requirements to or from lowest level requirements; to or from requirements to or from design; to or from design to or from implementation; to or from implementation to test; and to or from requirements to test. (IEEE, 1998) The IEEE 610.12 standard also defines traceability as "the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match" (IEEE, 1998)

During the 90s the acute "requirement problem" was under serious investigation and the IEEE published the IEEE Std 830-1998 standard to elevate the problem of creating good software requirement specification. A number of examples of good SRS are included. This standard is relevant to our work because it defines traceability as a base practice. The IEEE describes an SRS as traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation. They recommend to types of traceability:

a) Backward traceability (i.e., to previous stages of development). This depends upon each requirement explicitly referencing its source in earlier documents.

b) Forward traceability (i.e., to all documents spawned by the SRS). This depends upon each requirement in the SRS having a unique name or reference number.

It continues to describe that forward traceability of the SRS is especially important when the software product enters the operation and maintenance phase. As code and design documents are modified, it is essential to be able to ascertain the complete set of requirements that may be affected by those modifications.

One of the criticisms that we have with the IEEE 830 standard is that the requirement specifications described are different from those used in the real world, focusing too much on fulfilling the requirement list rather than on the intended user's objectives.



### 2.5.2 ISO 15504

In this section we review the ISO15504 assessment framework, in particular the description of implemention of traceability in the development of quality software. We will see in Chapter 11 how we evaluated the capabilities of our proposed TRAceability Process (TRAP) against the capabilities of the Rational Unified Process (RUP) using ISO 15504. In this section we merely illustrate that the ISO 15504 describes traceability practices across the product development lifecycle, in the hope that future researchers can identify these practices and perhaps incorporate them into future research.

In ISO 15504, traceability is defined in the Engineering Process (ENG), Software Design and Software Construction processes in the assessment framework. The ENG process category consists of processes that directly specify, implement or maintain the software product, its relation to the system and its customer documentation. Base Practice 2 in the engineering process is described as the "establishment of traceability practices." The traceability process is described as the activities in the development process to define the intermediate work products and methods for tracing requirements through these work products to the software product or system that is accepted by the customers. In the *Development Process*, Base Practice 7 dictates to "establish traceability between the

customer needs and the system requirements." Base Practice 8 instructs the user to evaluate the consistency between the software requirements and system requirements.

In the *Software Design Process* the assessment framework mandates that the user establish traceability but this time between the software requirements and the software design. In the *Software Construction Process* it instructs the developer to establish traceability between the software design and the software units and ensure consistency with software requirements.

The ISO 15504 document suite has a set of categories in which the assessors can place the data being assessed. The result is that the assessors can give an overall determination of the organisation's capabilities, which in the context of this study is the capability to implement traceability in the product lifecycle.

### 2.5.3 Capability Maturity Model

The Capability Maturity Model was developed by the Software Engineering Institute (SEI) and is used as a model for judging the maturity of the software processes, for identifying the key practices that are required to increase the maturity of these processes and to guide process improvement across a project or product development organisation. One of the primary goals of using the CMM model is to have processes that are repeatable, defined, managed, and optimized. It integrates separate organizational processes for example business processes with software development processes. (Paulk et al., 1995) In Chapter 11 we utilise the CMMI assessment model to assess the capabilities of the traceability framework that we propose.

In CMMI, the *Engineering Process* area influenced our research effort because it dictates traceability goals, practices and sub-practices. The Engineering Process areas integrate the processes associated with different engineering disciplines into a single Product Development Process, supporting a product-oriented process improvement strategy. The Engineering Process areas of CMMI are Requirements Development, Requirements Management, Technical Solution, Product Integration, Verification and Validation. The *Support Process* group area of CMMI are; Configuration Management, Process and Product Quality Assurance, Measurement and Analysis, Decision Analysis and Resolution, Causal Analysis and Resolution.

A specific goal describes a unique characteristic that must be present to satisfy a process area. A specific goal is a required component which determines if the process area is satisfied. For example a specific goal in the requirement management process area is to "Manage Requirements". A specific practice is the description of an activity that is important in achieving the specific goal of a process area. For example, a specific practice in the Requirement Management process area is "Maintain Bidirectional Traceability of Requirements". A sub-practice is a detailed description that provides guidance for implementing a specific or generic practice. For example, a sub-practice for the specific practice in the Requirement Management process area is "Maintain requirements traceability to ensure that the source of lower level (derived) requirements is documented." Generic practices are called "generic" because the same practice applies to multiple process areas. A generic practice is the description of an activity that is considered important in achieving the associated generic goal. A generic practice is an expected model component. (CMMI, 2006) In Figure 2-2, *CMMI Overview*, we illustrate a conceptual diagram of the CMMI Process Areas, specific goals, specific practices and sub-practices.

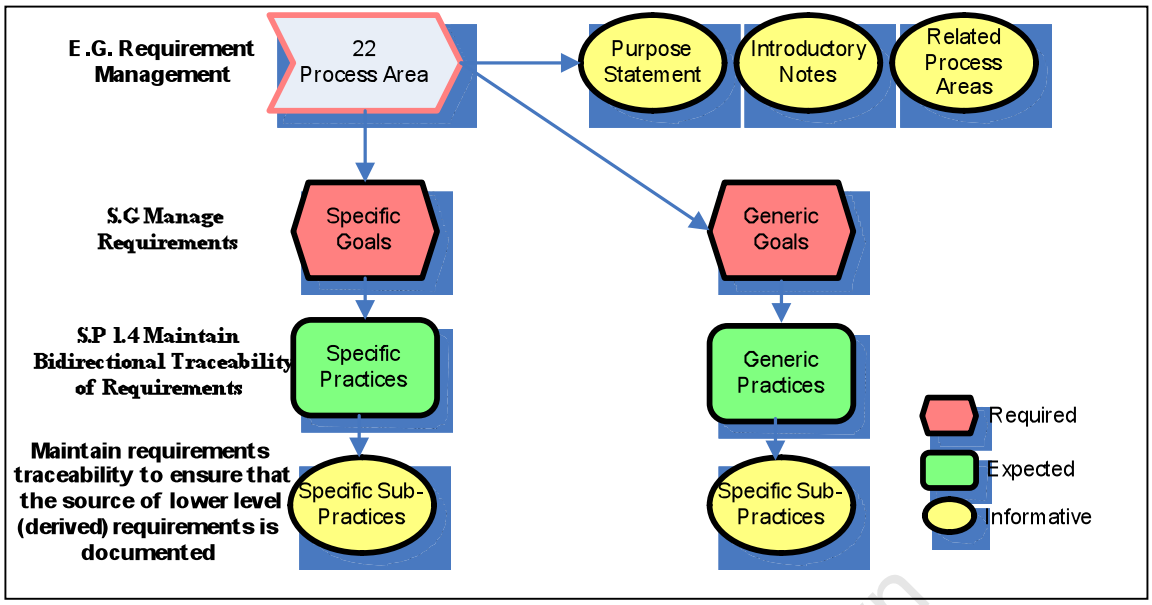



**Figure 2-2 CMMI Overview**

In conclusion, both ISO 15504 and CMM are the assessment frameworks that we utilise to assess certain aspects of our proposed solution framework. They describe traceability practices that should be undertaken to improve the quality of the software, based on best practices and lessons learned from the IT community, including both government-related and private industry. It is interesting to note that very few if any traceability researchers have used with ISO 15504 or CMMI to assess their work. This phenomenon seems strange to us, because both standards clearly describe traceability practices and they are both widely accepted by the software world as the primary assessment framework in use the world over. As we will see in Chapters 11, the only criticism that we have with these standards is there magnitude or size. It took the researcher a number of months to gain a full understanding of the content of the framework and its relationship to traceability.

## 2.6 CONCLUDING COMMENTS

Deriving traceability relationships from requirements is a principle that has been around since the term "software engineering" was first inaugurated in 1968, by the NATO Science Committee. If F.L Baurer was asked in his memoirs, to reflect on the most notable software engineering changes he has witnessed since he first coined the phrase "software crisis" in 1969, would he describe changes to the field of traceability? Perhaps not, and yet academicians and practitioners alike would stress their reliance on traceability as a core practice for ensuring the delivery of quality products and services to their customers. If, however, we were asked to reflect upon the period from 1960 to 2007 and the changes that have occurred in traceability, a definite sequence of events could be revealed. Traceability has its genesis in some of the great systems engineering books that appeared in the 70s; during the 80s new requirement engineering principles were formed, while unquestionably, the golden age of requirement engineering took place in the 90s when traceability appeared with regularity in scholastic journals, while the more realistic 00's saw the formation of communities, and the beginnings of collaborative projects that are still leading the way with new and exciting research.

Telling the story of traceability has given us time to reflect on the leading researchers of traceability in the past few decades, to mention but a few, Gotel and Finkelstein, Lindvall and Sandahl, Ramesh, Jarke, Zismann, Arkley and Riddle. The future for the field of traceability looks good since the formation of the Centre of Excellence for Traceability and the publication of the Grand Challenges technical report was indeed, a major leap forward for the development of future generations of traceability research. In Europe, the efforts of the ECMDA and research units like Sintef continue to produce high quality publications on traceability especially in the new model driven paradigm. Unquestionably, the field of traceability has emerged as a viable research domain, albeit, with many obstacles still to be overcome but with clearer and more focused objectives. This research projects attempts to build on the aforementioned tenets incorporating the lessons learned in an attempt to provide a solution framework and a body of knowledge that will continue the traceability story for years to come.

# Chapter 3 RESEARCH METHOD

## 3.1 PREAMBLE

In general a simple research process involves the steps of describing the problem and the hypothesis (Chapter 1), reviewing the literature (Chapter 2), designing the research method (Chapter 3), and systematically collecting and analysing the data. (Chapter 4 & 5 &10)

One of the problems that we observed during the literature review was the apparent lack of "tried and tested" research methods for carrying out empirical studies on traceability. Therefore, the primary objective of this chapter is to describe all the attributes in the design of the research method and to introduce the main concepts that we describe in more detail in later chapters.

In Figure 3-1 below, *Overview of Empirical Chapters*, we illustrate the chapter overview for the empirical study stages of this research. In this chapter we describe the design of the research method. In Chapter 4 we describe the initial findings of the case study, while in Chapter 5 we describe the survey. In Chapter 10 we revisit the case study and describe the changes that occurred between 2004 and 2007.

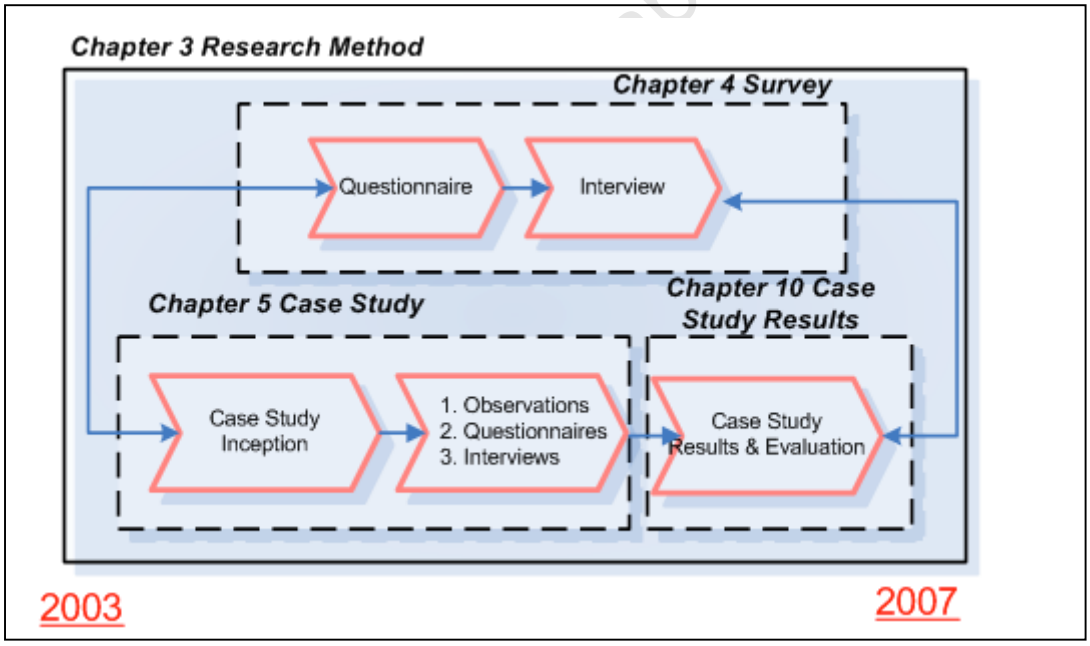



**Figure 3-1 Overview of Empirical Chapters**

## 3.2 RESEARCH METHOD CONTEXT

### 3.2.1 Introduction

One of the big challenges of this project was deciding which type of research method to use. Computer Science emerged from the field of applied mathematics. A lot of computer science research is conducted using analytical mathematical models where formal reasoning is the main approach for problem solving. Software engineering is a multi-disciplinary field within computer science. Critics of software engineering research argue that the theory should be induced from the industrial community. Software engineering deals with real-world problems involving humans, development organisations where less formal methods are used to solve the problems encountered in developing software products. Therefore the human dimension of software engineering in traditional computer science research must be augmented with additional methods. These research methods are drawn from the study of human behaviour. Each method provides a view of a phenomena and therefore using multiple methods is a good strategy.

For example, traditional scientific methods develop a theory to explain a phenomenon. A traditional engineering method observes existing solutions, proposes improvements and measures the improvement proposed. An empirical method begins with a hypothesis and the researcher designs a study, collects data and then tests the hypothesis using quantitative (statistical) methods. Unlike the scientific method there is not a formal model or theory describing the hypothesis. An analytic method takes results from a formal theory and compares the results with empirical observations.

Traceability is a software engineering sub-discipline or practice which traverses both technology and human boundaries. To understand traceability we need to investigate the theory, tools and processes but also the social and cognitive processes surrounding them. In a nutshell traceability is the study of human activities both at an individual software engineering level as well as at an organisational level. To understand traceability we need to investigate what are the attitudes to the practices, understand the what, who, when, why and how of traceability. We must identify the problems that people and organisations face when implementing traceability, understand the impact on traceability that changes to organisation structures and changes to product strategies have on the traceability practices.



### 3.2.2 Research Method Background

In this section we introduce the main components of our research method, for example, quantitative and qualitative approaches. This section merely introduces the main concepts that we develop further in later sections.

We needed methodologies that support the collection of large amounts of data from a case study and a survey, plus exploratory lab trials and field trips. Therefore we decided that mixing quantitative and qualitative techniques provided us with the best possible approach for dealing with the diversity in the data that we proposed to collect and to assist us to overcome the science and the human factors of this study.

Quantitative research emphasizes quantification in the collection and analysis of data. As a research strategy it is deductivist and objectivist and incorporates a natural science model of the research process (Bryman, 2001). Examples of quantitative methods include survey methods, laboratory experiments, formal methods and numerical methods such as mathematical modelling. In our case we use quantitative techniques to further our

understanding of traceability in a cross-sectional sample of industrial data, to better understand the attitudes of software engineers for traceability and to gain an insight into the problems encountered implementing traceability. We analysed the data gathered to assist us confirm our research questions and hence our direction.

Qualitative research methods on the other hand were developed in the social sciences to enable researchers to study social and cultural phenomena. Qualitative research emphasizes words rather than quantification in the collection and analysis of data. As a research strategy it is inductivist, constructivist and interpretivist. Examples of qualitative methods are action research and *case study* research. Qualitative data sources include observation and participant observation (fieldwork), interviews and questionnaires, documents and texts, and the researcher's impressions and reactions.

Another qualitative research method is the emergence of new theory or *grounded theory* from data systematically gathered and analysed from industry. Grounded theory is "an inductive, theory discovery methodology that allows the researcher to develop a theoretical account of the general features of a topic while simultaneously grounding the account in empirical observations or data." (Martin and Turner, 1986) The major difference between grounded theory and other methods is its specific approach to theory development; grounded theory suggests that there should be a continuous interplay between data collection and analysis.

The motivation for doing qualitative research, as opposed to quantitative research, comes from our ability to talk. (Myers, 1997) Qualitative research methods are designed to help researchers understand people and the social and cultural contexts within which they live. Qualitative research is an intensely personal and subjective style of research. On the other hand, quantitative researchers strive for testable and confirmable theories that can explain how one set of variables is related to another. Quantitative research reduces human behaviour to a set of finite characteristics that can be quantified and operationalised so that they can easily be tested.

In summary, we use both qualitative and quantitative methods in this study. Methods that are primarily qualitative include ethnographies, case studies and action research. These methods rely on fieldwork, using techniques such as participant observation and interviews. Methods that are primarily quantitative include controlled experiments and survey research. These methods require more significant time in the planning of the research than strictly qualitative methods.



## 3.3 OUR RESEARCH METHOD STRATEGY

Over the past decade, there has been an increasing trend of blending quantitative and qualitative methods and data within a study to provide a broader, deeper perspective. Mixed method research employs data collection and analysis techniques associated with both quantitative and qualitative data. This approach is called *triangulation.* Both quantitative and qualitative research designs seek reliable and valid results. Data that are consistent or stable as indicated by the researcher's ability to replicate the findings is of major concern in the quantitative arena, while validity of the qualitative findings are paramount so that data are representative of a true and full picture of constructs under investigation. By combining methods, the advantage of each methodology complements the other, leading to stronger research design which results in more valid and reliable findings. The inadequacies of individual methods are minimized and more threats to internal validity are realized and addressed.

Stake stated that the protocols that are used to ensure accuracy and alternative explanations are called triangulation. (Stake, 1995) (Tellis, 1997)The need for triangulation arises from the ethical need to confirm the validity of the processes. Denzin identified four types of triangulation: *Data source triangulation*, when the researcher looks for the data to remain the same in different contexts; *Investigator triangulation*, when several investigators examine the same phenomenon; *Theory triangulation*, when investigators with different view points interpret the same results; and *Methodological triangulation*, when one approach is followed by another, to increase confidence in the interpretation. (Denzin, 1984) (Tellis, 1997)

Our approach, as illustrated in Figure 3-2 below, *Our General Approach*, is to triangulate methods, data and theory into a *concurrent triangulation strategy*. This strategy uses different methods concurrently, in an attempt to confirm, cross-validate or corroborate findings. We collect quantitative data from a survey of software engineers to compare against the data gathered during the case study. By collecting both types of data simultaneously, rather than sequentially, analysis can be adapted to explore emerging results from the other.

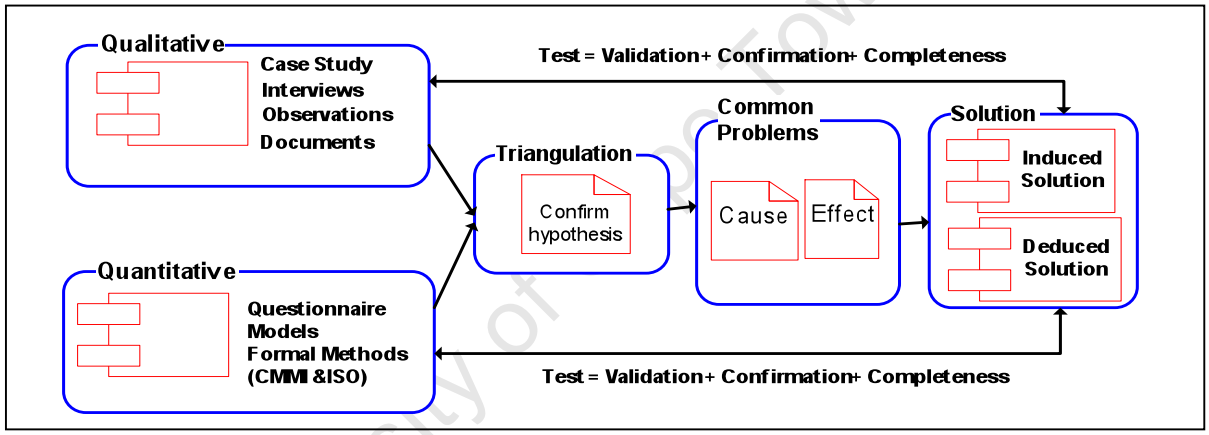



**Figure 3-2: Our General Approach**



## 3.4 RESEARCH METHOD DESIGN

Research design refers to the strategy to integrate the different components of the research project in a cohesive and coherent way. (Trochim and Land, 1982) A method is a set of organizing principles around which empirical data is collected and analyzed. The selection of the method depends on the access to the necessary resources and the alignment of the method to the questions posed.

The initial key element of successful empirical research design is to focus the research direction with a clear research question. This helps in understanding the research goals which assists with the method selection. The research method helps to decide the steps to achieve the goals of a study and decides what kinds of data to collect and how to collect it. A theory helps to explain the data and relate it to the research questions and previous studies in the literature. A proper set of criteria for assessing the validity helps improve the study and clarify the nature of the conclusions.

As shown in Figure 3-3 below, *Research Method Basic Principles*, before designing our research method we began by dividing the project into conceptual high-level activities*; explore, describe* and *explain*.

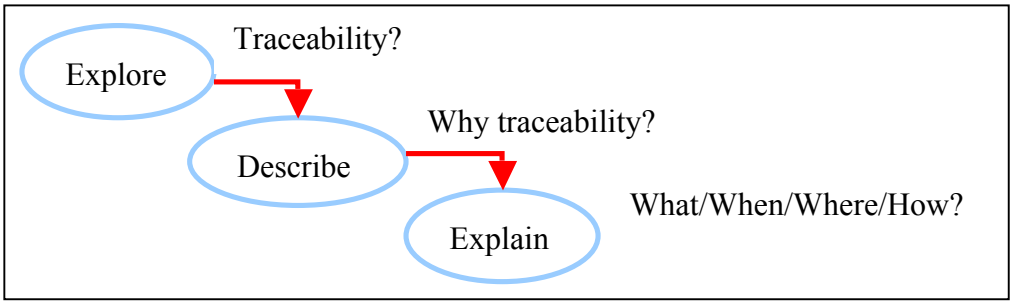



**Figure 3-3: Research Method Basic Principles**

We began by asking exploratory questions to gain a better understanding of traceability and identify focus areas of specialised interest which helped us to build tentative theories. As mentioned already in this report we had a practitioners understanding of traceability but what we needed was an insight into the research being undertaken. The next step was to understand traceability in certain situations and contexts. We observe and then describe what is observed. In simple terms this is the "why" of our research. The next stage is to explain the observations, by answering questions on what, where, when and how and the explanatory questions of why. For example, describing the problems of traceability is a descriptive activity while explaining what, when, where and how is an explanatory activity. The classifications for the questions that we asked was adapted from Meltzoff's approach as follows: (Meltzoff, 1998)

- **Existence Questions:** Is there a traceability research community? If so what research are they carrying out? What are the findings of the empirical studies? Are there processes that incorporate traceability practices? Is there a traceability research community? Are there conferences or workshops that specialise in traceability?
- **Description and Classification questions:** What is traceability? Are there different types of traceability? What are the components of traceability? What are traceability items and what is the relationship between items? What are traceability item types? Who practices traceability in the development lifecycle? What are the attitudes of the software engineers towards traceability in the development lifecycle? Is uniform traceability terminology used?
- **Descriptive-Comparative Questions:** Is there a comparison between the attitudes of different roles to traceability and the maturity of the methods and tools in place in the organisation? Is there literature on the attitudes of the roles to traceability? What is the comparison between UML and traceability? Is there a gap between the theory and the practice? We next ask base-rate questions to gain an understanding of whether a particular phenomena is normal or unusual. Some of the questions that we asked were:
- **Frequency and distribution questions:** When does traceability take place most frequently? Where in the development lifecycle is traceability practiced the most or the least? How many relationships do traceability items have?
- **Descriptive-Process questions:** What do the processes state about traceability? What is the difference between the different software processes? What are the different steps and activities to create traceability relationships? What are the traceability best practices? Are these processes supported by tools, if so what are the capabilities of the tools? What are the

factors that influence process definition for traceability? Have patterns been used to describe traceability practices?

- **Relationship questions:** Is consistent terminology used between practitioners and the research community? What is the relationship between traceability and visual modelling? Is there a relationship between traceability and models? What is the relationship between the assessments of processes?
- **Causality questions:** What are the factors that influence traceability (good and bad)? What are the problems with traceability? What causes good and bad implementation of traceability? What effect does a corporate strategy have on traceability? What effect does a tooling strategy have on traceability? What effect does resourcing or requirement managers and administrators have on traceability? Does poor communication cause problems when implementing traceability?
- **Causality-Comparative questions**: Investigate relationships between different causes. Does the lack of a common terminology cause problems when implementing traceability? What is the difference between the problems implementing traceability in large organisations compare with small to medium organisations? Do changes in the product structure have a positive or negative effect on traceability practices?
- **Causality-Comparative Interaction** questions investigate how context affects a cause-effect relationship: Is traceability more important in telecommunications projects than other projects? Does a better process lead to better traceability? Would the use of patterns lead to all round better traceability practices? Would a semantic model lead to better communication and unification of traceability practices?

On analysis of this long list of questions, we documented that themes and categories that emerged. For example, most of the questions could fall under the following headings; process, tools, people, literature and practices. As we will see in Chapters 4, the case study, and Chapter 5, the industrial survey, we used these categories and questions in the design of the questionnaires and interview scripts. For example, what is your attitude towards the importance of traceability or what are the factors that influence traceability in your organisation? Other questions from above were used in the analysis of the data. For example, is consistent terminology used between practitioners and the research community or what effect does a tooling strategy have on traceability? One of these questions is discussed in Appendix I, where we review the capabilities of a number of traceability tools. Overall, we would recommend the above approach for creating research questions. It not only helped us in the design of the questionnaires and surveys and in the analysis of the data it also helped us define the scope and objectives for this project. This was particularly useful during the case study, as on a number of occasions we returned to the above list of questions to assess if the activity we were about to undertake answered some of the questions above, if not then it was outside the scope of the project. As one would expect the causality questions helped us gain a better understanding of the problems or challenges that the practitioners faced.

In Figure 4 below, *Total Research Method*, we describe the main phases or steps in our research method as follows:

1. *Problem Analysis*: In this phase we use structured analysis of the factors that influence traceability in order to establish the cause and effects. We begin with a hypothesis and refine this hypothesis using literature, case study and an industrial survey.
2. *Technical Framework:* In this phase we generate our Traceability Framework that elaborates our hypothesis with a proposed solution framework.

3. *Application & Validation:* In this phase we apply our solution framework against the Ericsson OSS-RC domain. We validate our models and use the CMMI and ISO 15504 assessments frameworks to assess the processes defined.

This cross-sectional approach involves observations of a sample or cross section of traceability practices at one point in time. While using a cross-sectional study assisted us satisfy some of our initial objectives and gain a better understanding of the problems we combine our cross-sectional study with a longitudinal study which permitted us to observe the traceability over an extended period. Our case study uses observations, questionnaires, interviews and artefacts to study changes to traceability practices and experiences over a four year period.

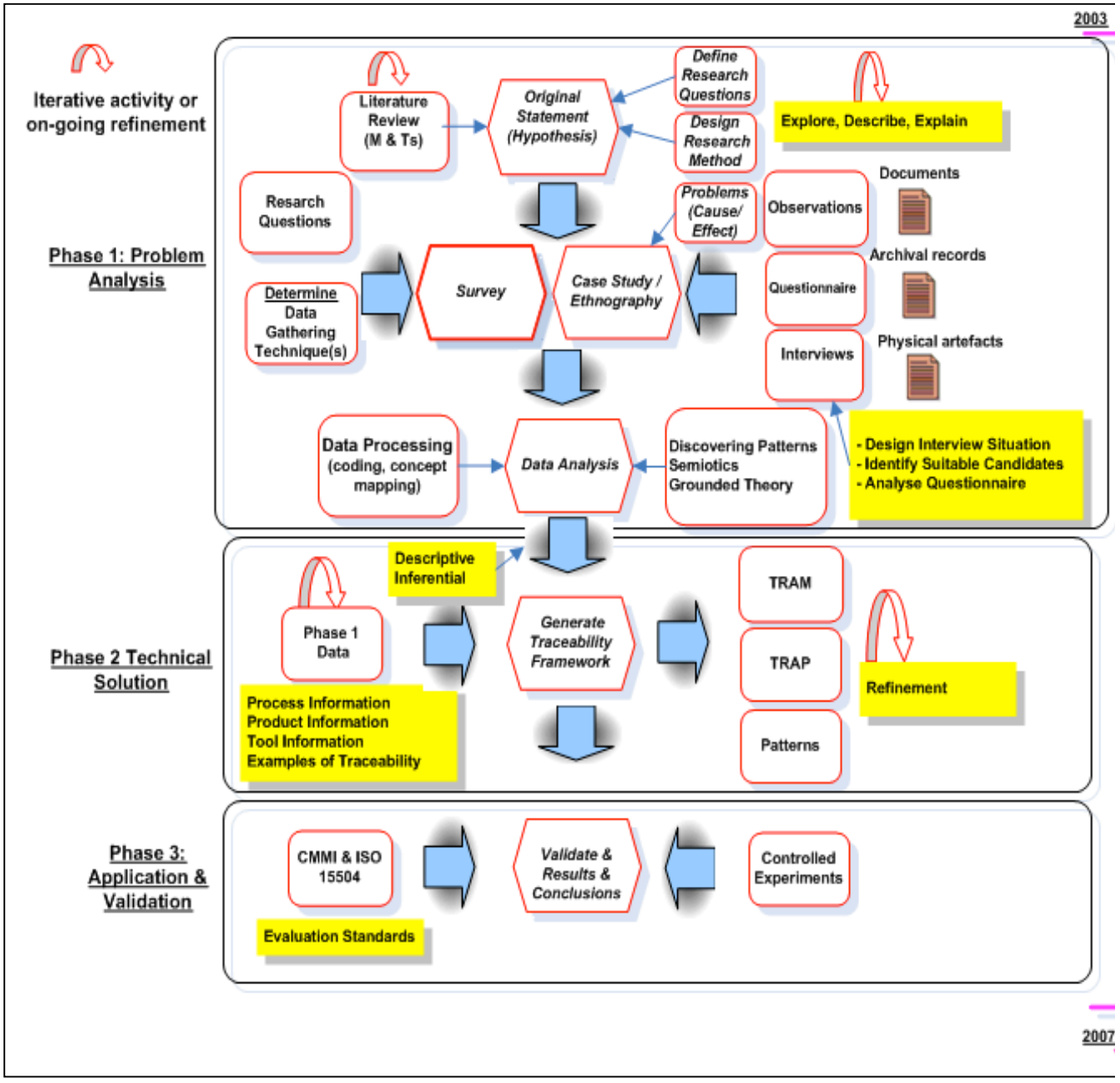



**Figure 3-4: Total Research Method**

## 3.5 MODES OF OBSERVATIONS

### 3.5.1 Case Study

In early 2004 Ericsson's agreed to participate in an extensive case study. With great enthusiasm we incorporated a four year case study into our overall research method. We defined a number of clear objectives; to document the traceability environment, the resources involved, the tools being used, the processes, the product structures, the strategies, the attitudes and also how these factors evolved or changed throughout the duration of this project.

Yin introduces the case study as "an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident." (Yin, 1994) Case studies offer in-depth understanding of how and why certain phenomena occur, and can reveal the mechanisms by which cause-effect relationships occur. Yin identified some specific types of case studies: *Exploratory*, *Explanatory*, and *Descriptive.* (Yin, 1993) Our approach combines the three case study types. Exploratory cases are sometimes considered as a prelude to research and are used as initial investigations of some phenomena to derive new hypotheses and build theories. In this study we had carried out exploratory investigations before starting this study. Explanatory case studies may be used for doing causal investigations. Descriptive cases require a descriptive theory to be developed before starting the project. Pyecha used this methodology in a special education study, using a pattern-matching procedure. (Pyecha, 1988).

Yin (1994) identified five components of research design that are important for case studies and surveys:

- A study's questions
- Its propositions, if any
- Its unit(s) of analysis
- The logic linking the data to the propositions
- The criteria for interpreting the findings (Yin, 1994, p. 20).

Stake and Yin identified at least six sources of evidence in case studies. The following is not an ordered list, but reflects the research of both Yin (1994) and Stake (1995):

- Documents
- Archival records
- Interviews
- Direct observation
- Participant-observation
- Physical artefacts

By following this process we identified the units of analysis. There is virtually no limit to what or who can be studied or what is referred to as the *unit of analysis* when

studying traceability. The unit of analysis is the major entity that is being analyzed in the study. It is the 'what' or 'whom' that is being studied. In social science research, the most typical units of analysis are individual people. Other units of analysis can be groups, social organizations and social artifacts. The main units of analysis in our project are *software engineers* and *organisations* (Babbie, 2001)

Time is an important variable in the design and execution of any research method. Apart from the obvious consideration of how much time we had for this project, we also had to decide when to design the research method, when we were ready to start data collection and when we start data analysis, when we move from one research phase to another phase, when we are ready to start the design of the solution, when we begin testing, validation and the conclusion stages.

A precondition for conducting a case study is a clear research question concerned with how or why certain phenomena occur. This is used to derive a study proposition that states precisely what the study is intended to show, and to guide the selection of cases and the types of data to collect. In our case study we started with one simple question: "what are the factors that influence traceability and how do these factors change over time?"

Although an individual case study often reveals deep insights, the validity of the results depends on a broader framework of empirical induction. A frequent criticism of case study methodology is its dependence on a single case renders it incapable of providing a generalizing conclusion. In very complex and multivariate cases, the analysis can make use of pattern-matching techniques.

The quintessential characteristic of case studies is that they strive towards a holistic understanding of cultural systems of action (Feagin, 1991) (Tellis, 1997) Cultural systems of action refer to sets of interrelated activities engaged in by the actors in a social situation. The case studies must always have boundaries (Stake, 1995) Case study research is not sampling research, which is a fact asserted by all the major researchers in the field, including Yin, Stake, Feagin and others. However, selecting cases must be done so as to maximize what can be learned, in the period of time available for the study. In our case study the boundary for the case study was the OSS-RC telecom domain and the time available was the duration of the study. While future collaborative work will be undertaken with Ericsson it was evaluated that studying an organisations and software engineers for four years was deemed sufficient.



### 3.5.2 Survey research

While the results from the case study provided us with a multitude of data it only provided us with a single source which does not provide sufficient evidence to support a complete picture of the current state of the art of traceability. An industrial survey was planned to provide extra data that we could apply triangulation techniques against our case study. As discussed in Chapter 1, a newly formed consortium of software engineers met once a month at the University of Cape Town. The Software Process Improvement Network (SPIN) was loosely amalgamated to the Carnegie Mellon, Software Engineering Institute (SEI) and provided a suitable source of software engineering practitioners from a variety of different industrial contexts.

A survey encompasses any method that measures the results from asking questions of respondents. A pre-condition of conducting a survey is a clear research question that asks about the nature of a particular target population. In our study we wanted to

understand traceability in different industrial setting or contexts. Our unit of analysis in the survey is software engineers and organisations in different industrial contexts. Questionnaires and interviews are the two broad survey categories.

The purpose of this survey, which was conducted between December 2004 and August 2005, was to establish the state of the art in the practice of requirements engineering and traceability amongst a variety of industrial contexts. Several surveys on traceability have been carried out over the past two decades. For example, as already discussed in Chapter 2, Gotel and Finkelstein's presentation of an empirical traceability study consisting of 100 samples of data. (Gotel and Finkelstein, 1994b)

In 2001, Ramesh completed a survey of 26 major software development organisations to investigate traceability practices. He captured the information needs (with respect to traceability) of different stakeholders involved in the software development process. In the first phase data collection methods included evaluation of major traceability tools[8], and structured interviews with practitioners. The second phase involved the development of reference models of traceability practice. Data from 23 focus groups was used in the development and classification of traceability links representing current practice and ideal practice. From the observations he produced a reference model comprising of the important types of traceability links for various development tasks. He validated the models in case studies and incorporated the models into a tool.

### 3.5.3 Ethnographies

Yin (1994, pp. 10-11) describes, "Ethnographies usually require long periods of time in the 'field' and emphasize detailed, observational evidence". Ethnographic research comes from the discipline of social and cultural anthropology where an ethnographer is required to spend a significant amount of time in the field. Ethnography is a form of research focusing on the sociology of meaning through field observation. The goal is to study a community of people to understand how the members of that community make sense of their social interactions (Robinson et al., 2007) (Easterbrook et al., 2007) For software engineering, ethnography can help to understand how technical communities build a culture of practices and communication strategies that enables them to perform technical work.

While there is often confusion between case study and ethnographies we believe that in this study we were involved in both. While carrying out the case study we got requested to be involved in the definition of a requirement engineering process. While this was outside the scope of the case study we agreed because it would further our understanding of the traceability context, help us gain access to further resources providing us more observational evidence in relation to the factors that influence traceability. The observations made and the experiences incurred along the way were invaluable to our understanding. Furthermore, as Ericsson was using CMM to assess the process model it gave us a grounding in the CMM assessment model, working closely with the assessors.



[8] See Appendix I for our tool evaluation.

### 3.5.4 Controlled Experiments

A controlled experiment is an investigation of a testable hypothesis where one or more independent variables are manipulated to measure their effect on one or more dependent variables. Controlled experiments allowed us to determine in precise terms how the variables under investigation were related and, specifically, whether a cause-effect relationship existed between them. Each combination of values of the independent variables is a treatment. The simplest experiment's had just two treatments representing two levels of a single independent variable. More complex experimental designs arise when there are more than two levels or more than one independent variable used. Most software engineering experiments require human subjects to perform some tasks. We measured the effect of the treatments on the subjects. For example, measuring the effect of the changes to their traceability practices. A precondition for conducting an experiment is a clear hypothesis. The hypothesis guided all steps of the experimental design, including deciding which variables to include in the study and how to measure them.

Experimental control was an important aspect of our research. Variables other than the chosen independent variables must not be allowed to affect the experiment. In our case time-series experiments were carried out, effecting the treatment of a measured in the discrete time steps over a period of time. These variations are less powerful than true experiments, and required more careful interpretation.



## 3.6 DATA PROCESSING

### 3.6.1 Coding

The key process in the analysis of our data is coding, classifying or categorising individual pieces of data. The purpose of coding was to chronicle our data in an order which could be put under investigation. We evaluated that the best way to learn how to understand and document changes to the factors that influence traceability was by learning from others who were involved in the process. We provided a patterns template to the software engineers who were involved in the practice of traceability. It was on analysis of our templates or coded data that we discovered that patterns were appearing. By keeping good pattern version control we could gain an insight into the changes that were occurring over time. For example, the pattern of tracing Application Requirement Specifications to the Use Cases in the earlier versions of the OSS-RC changed to tracing between the Main Requirement Specification to the Requirement Specifications in the later stages.

## 3.7 DATA ANALYSIS

### 3.7.1 Discovering Patterns

We followed John and Lyn Lofland (1995) suggested ways of looking for patterns in the data that we gathered. The six ways of looking for patterns are:

- Frequency: How often were certain activities or practices carried out? This information was gathered through observations, interviews and by analysis of questionnaires.
- Magnitudes: How much time was spent at each activity?
- Structures: What are the different types of traceability? For example, tracing from main requirements to application requirements.
- Process: Can we analyse the processes and identify patterns.

- Causes: What are the causes of good and bad traceability practices?
- Consequences: How do good and bad traceability practices effect the organisation?

We examined the data looking for patterns using two main approaches. In *variable – oriented analysis* we looked for interrelationships between variables, and the people observed are the carriers of these variables. The aim was to gain an overall explanation using a few relatively number of variables. This proved difficult in our study of Ericsson because there were so many variables that effected traceability. We used Ramesh conceptual framework to initially identify the factors that he discovered influenced traceability. With his work as a reference point we continued our investigations. In *case-oriented analysis* we look more closely for patterns in a particular case. For example, patterns that occurred between in the relationship between the traceability items.

### 3.7.2 Grounded Theory Analysis

Grounded theory approaches are becoming increasingly common in IS research literature because the method is extremely useful in developing context-based, process-oriented descriptions and explanations of the phenomenon

While grounded theory is a research method we utilised it as an analysis technique to seek and develop theory that is grounded in data systematically gathered. According to Martin and Turner, grounded theory is "an inductive, theory discovery methodology that allows the researcher to develop a theoretical account of the general features of a topic while simultaneously grounding the account in empirical observations or data."(Martin and Turner, 1986) The major difference between grounded theory and other methods is its specific approach to theory development - grounded theory suggests that there should be a continuous interplay between data collection and analysis. We utilise Glaser and Straus four staged approach; comparing incidents applicable to each category, integrating categories and their properties, delimiting the theory and writing the theory.(Glaser, 1967) In our research we have tight coupling between pattern discovery and grounded theory.

## 3.8 DATA ANALYSIS PROCESS

The process we followed for data analysis is described in this section.

### 3.8.1 Data Preparation

#### 3.8.1.1 Logging the Data

In our study the amount of data was small enough not to need database software to store all the data instead using well structured data archives and spreadsheets. The most critical aspect of the data analysis was that we recorded the original data records for the interviews, questionnaires, observations and so on. Strong traceability relationships were tied between the data and the output statistics.

#### 3.8.1.2 Checking the Data for Accuracy

As soon as we received the data we screened it for accuracy. In some circumstances doing this straight away allowed us to go back to the source of the sample to clarify any problems or errors. There are several questions we asked as part of this initial data screening: Are the responses legitimate? Are all important questions answered? Are the

responses complete? Is all relevant contextual information included, for example the time, place and software engineering role?

On the second screening we asked more pertinent questions: How would the current climate in the organisation have affected the answer we received? For example, organisations who were going through down-sizing may affect the answer we received? Would the number of years of experience affect the answers?

#### 3.8.1.3 Data Transformations

We transformed the data into categories and codes and scaled the data where possible.

#### 3.8.1.4 Descriptive Statistics

Descriptive statistics are used to describe the basic features of the data in a study. They provide simple summaries about the sample and the measures. Together with simple graphics analysis, they form the basis of virtually every quantitative analysis of data.

With descriptive statistics you are simply describing what is or what the data shows. Descriptive Statistics is often used to present quantitative descriptions in a manageable form. In our research study we had a large amount of data to analyse. Descriptive statistics helped us to represent large amounts of data in a sensible and understandable way.

### 3.9 CONCLUDING COMMENTS

Traceability is a practical science and a real-world practice. In our opinion the most effective way for learning about traceability is to actually be in real situations, learning from the environment, reflecting on the various pressures practitioners and managers confront in everyday organizational life, documenting the problems and understanding their consequences. Nothing will ever replace learning from experience. Our research method principle is simply based on the fact that learning occurs when we immerse ourselves into the every day world of traceability. As we learn we understand the factors influencing traceability increasing the possibility of discovering new theory that satisfies the needs of the user community. Even if we don't find the solution, the evidence that we gather provide the research community with the foundation for future developments.

In the next chapters we begin to unravel the story of traceability as it evolved in this study. We begin by describing one of the main contributions, the case study before describing the survey.

# Chapter 4 CASE STUDY: FACTORS THAT INFLUENCE TRACEABILITY IN ERICSSON'S OSS DEVELOPMENT

## 4.1 INTRODUCTION

*"In a time of universal deceit, telling the truth becomes a revolutionary act,"*

- George Orwell

In this chapter we describe the technical context in which the case study was set, the corporate strategies that were in place, the processes, the tooling situation, the supporting documentation and the educational situation with an analysis of each factor and how it impacted the implementation of traceability. In essence, we present traceability as we observed it in 2004. We analyse the attitudes that different software engineers had towards traceability and interpret the impacts this had on the overall implementation of this practice.

The significance of this case studies contribution lies in the documentation of the factors that influence traceability in a real life context over a four year period. In later Chapters (see Chapters 7,8, and 9) we build on the experience gained by designing and testing a traceability solution framework. It was not the intention of this study to institute change in any way to the traceability practices in Ericsson but rather to explore, observe, analyse, and report the findings. Our primary objective was to gain an insight into the problems, issues, successes and failures that emerged in a large telecoms organisation.

A summary of our approach is shown below in the Figure 4-1, *Overview of Chapter*. We begin with a brief review of the history of requirement engineering in Ericsson before describing the research method that we followed. In the next section we describe the technological context of the study, from the evolution of 3G standards to the Operation Support System (OSS) product line in which this study is set. We then describe the corporate strategy, the processes and the traceability practices before concluding with a summary of our findings. Where possible, we support our descriptions with examples taken from the real life case. While this chapter is useful as a standalone description of traceability in a large development organisation, however, combined with the data that we captured over the next four years we build a story of the evolutionary nature of traceability and the impacts it can have on the entire development unit.

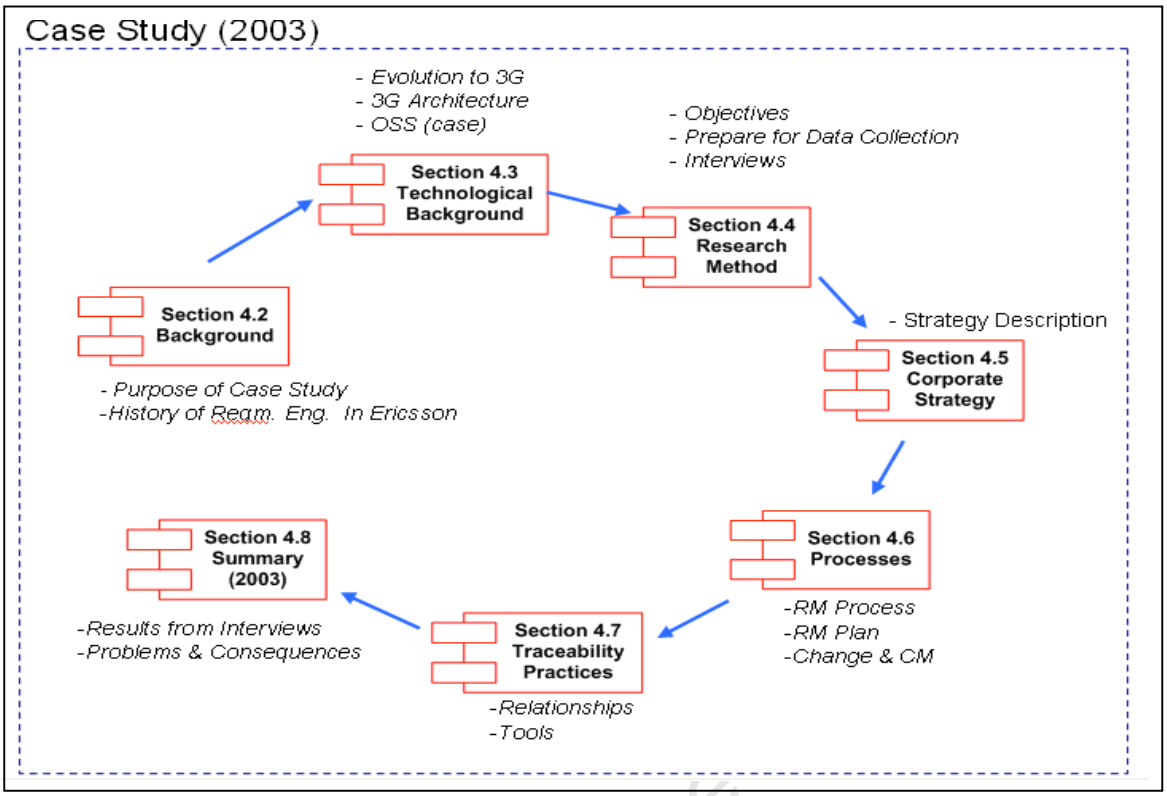



**Figure 4-1 Overview of Chapter**



## 4.2 BACKGROUND

### 4.2.1 Purpose of Case Study

With this background, the purpose of this case study was to identify the factors affecting the practice of traceability during the development of a complex software-intensive telecommunications system in the period 2004 – 2007.

Case studies, according to Stake, have become one of the most common ways to do qualitative inquiry. (Stake, 1995) Despite renewed popularity, a case study is a well used term that has many meanings. (Stablein, 2006) Today, the notion of a *case* is still open to much debate. In the context of this research effort, our *case* focuses on the factors that influence traceability in the development of Operation Support Systems for 3rd generation (3G) mobile telephone systems.

Case studies are particularly appropriate when trying to investigate practices in a field that lacks empirical data. In his classic book on case study research, Yin[1] argues that case study research is superior to survey methods at answering the "whys" and "hows" because the case analysis can delve more deeply into motivations and actions than structured surveys. Case studies are the preferred research strategy when “how”, “what” and “why” questions are being asked, when the researcher has little control over the event or when the research is being carried out in a real life context. (Yin, 1989) Stake distinguishes between two different kinds of case study: intrinsic and instrumental. (Stake, 1995) An intrinsic case study aims to provide a better understanding of one particular case. An instrumental case study is designed to provide insight into an issue and to develop generalizations applicable to other cases. The purpose of our case is both intrinsic and instrumental, because some of the factors are specifically intrinsic to the

telecommunications field and the 3G telecom domain, and others are of a more general nature. Our broader aim here, though, is to obtain an interpretive understanding which is applicable to multiple traceability contexts (Guba and Lincoln, 1994), with an eye to theory building (Eisenhardt, 1989; Yin, 2003) and theory testing (Yin, 2003).

### 4.2.2 History of Requirement Engineering in Ericsson

In 1876 Lars Magnus Ericsson, a Swedish inventor and founder of Ericsson began development of telephone equipment from a small workshop in the center of Stockholm. From these humbling beginnings, Ericsson has become one of the market leaders in telecommunication systems. To maintain this market position with production in both hardware and software many experts would agree that their success is attributed to highly skilled staff, mature processes and the application of powerful tools.

By the early 70s Ericsson's had developed its own internal propriety procedural language called PLEX (Programming Language for the AXE), which was used in the development of their main product line, the AXE.[9] Waterfall methodologies were utilised to deliver telecom products in giant monolithic projects throughout the 70s and 80s. Similarly to many software companies of this time, Ericsson's had a propriety methods and tools strategy developing all processes and tools in-house. Their project management model PROPS, described the main project activities, the milestones and the deliverables with regimental clarity and simplicity. As shown in Figure 4-2 below, *PROPS and MEDXAX Processes*, the PROP's project model was supported by a design methodology called MEDAX (Method for the AXE). The widespread use of PROPS and MEDAX gave Ericsson a common foundation in project concepts, documents, tools, and terminology in the development of one main product line, the AXE.

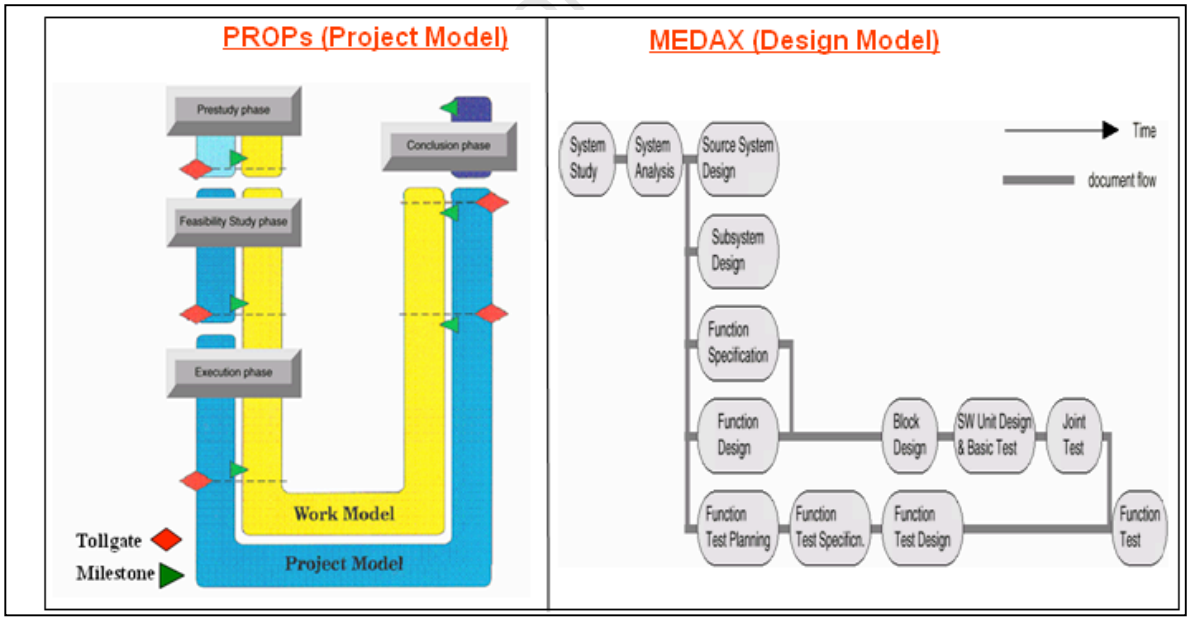



**Figure 4-2 PROPs & MEDAX Processes**

[9] It should be noted that the AXE initially was a telephone exchange. The acronym is a product classification code.

However, even this well founded approach still ran into difficulties. Because MEDAX was a document-centric approach, it utilised documents in all parts of the development lifecycle. As requirements, specifications, and descriptions were refined and clarified, new documents were derived. A fundamental problem with this approach was the lack of a single central, complete, and consistent representation of the system available to all parties involved in the development (customer, project management, systems engineers, design, and test). Each phase of development had a different set of documents as their focus. Changes to a high-level document could require changes to several lower level documents. Low-level documents thus inherited faults from higher level documents. The only check on quality was through the review and inspection processes. With little tool support, traceability had to be maintained manually.

Furthermore, the Waterfall Model of software development, which is a single iteration of sequential processes, had many inherent problems. Many software engineers argued that it was a bad idea, mainly because it was impossible to get one phase of a software product's lifecycle in a complete form before the next step started. For example, client requirements were never complete at the time of requirement specification. Each phase needed information from the following phases to be fully complete. For example, the requirements phase needed information from the design phase as to what is feasible; the design phase needs feedback from the coding phase as to which design has the best chances of succeeding and so on.

By the early 90s, new telecommunication messaging standards such as GSM were being requested by a new wave in mobility demands. With Ericsson's participation in these evolving telecommunication standards and new complexity brought about by new object oriented programming languages, new platforms with new architectures and new processes were needed. The old Waterfall Model of delivering products as one giant, monolithic project was slowly being replaced with new processes where the organization, technology, product management, developers and testers all worked in unison at the same time. The era of iterative and incremental development had begun.

In 1991, Ericsson acquired Objectory AB, which was founded by Ivar Jacobson, and was the result of 30 years experience that he gained while working with Ericsson's. Objectory, an object-oriented design method built on use-cases, was rapidly gaining recognition in the software industry. In 1990 Rational launched the development of a modelling tool called Rose which supported the graphical notation developed by Grady-Booch. Rose 1.0 was introduced at OOPSLA in 1992. Then, in 1995, Objectory AB was acquired from Ericsson by Rational Enterprises. Also in 1995, James Rumbaugh joined Rational, who by then had teamed up with Ivar Jacobsen, and the three leading object-oriented methodologists, known as the Three Amigos, worked together. Eventually, their unified efforts led to the release of the Unified Modeling Language (UML) and the release of the Rational Unified Process (RUP), a configurable process supported by an integrated tool suite. Rational then aggressively acquired a number of key software development products including Requisite Pro, a requirement management tool and ObjecTime which led to the release of Rational Rose RealTime, enabling real-time modeling.

In 1997, Rational entered into partnership with Ericsson in a multi-million dollar Joint Development Initiative (JDI) project. A number of new processes were developed during this project addressing the early stages of requirements analysis, design and test. The REME (Requirement Engineering for Ericsson) method was a general process for the early phases of software development, covering requirement analysis and early design. It was a

use-case-driven process, based on the UML industry-standard, with a primary objective of facilitating the improvement of requirement elicitation practices with the customer. It was also an iterative process, mandating new traceability practices across the development lifecycle. In contrast to MEDAX's document centric approach, REME (Requirement Engineering Method fro Ericsson) was a model driven process. As illustrated in Figure 4-3 below, *The three components of REME,* REME consisted of three components; the Specification of Functionality (SOF), the Specification of Reference Model (SORM) and the Distribution of Functionality (DOF).

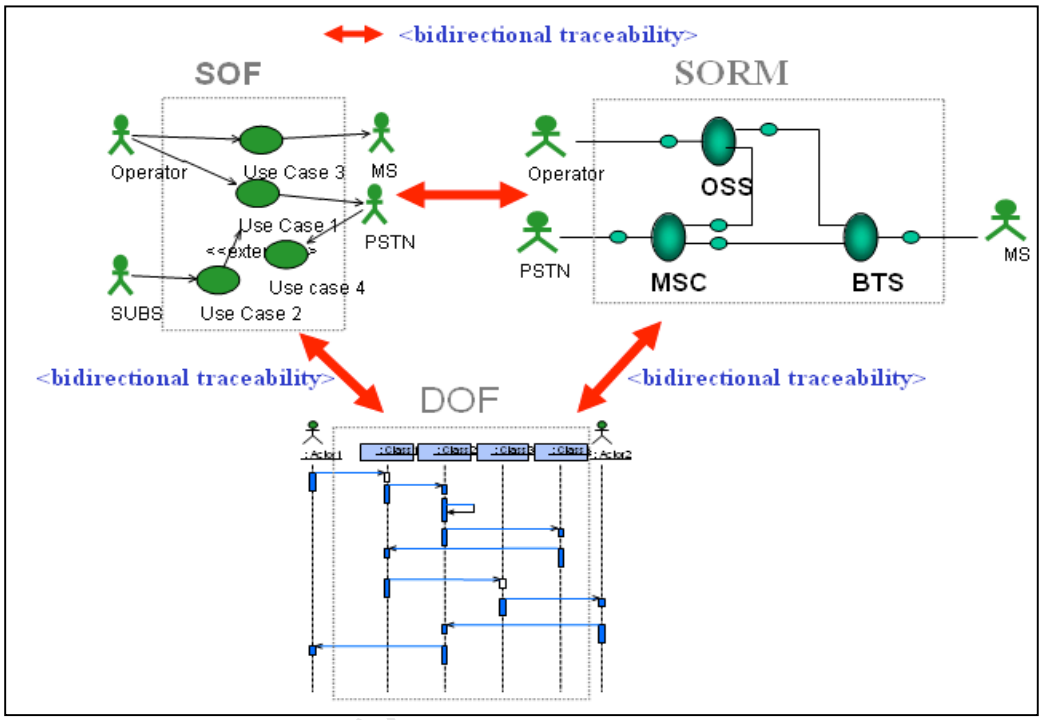



**Figure 4-3 The three components of REME**

The Specification of Functionality (SoF) involved building use-case models specifying the functionality as a gray box, that is, for developing implementation-independent specifications of the functionality. The output of SoF was a description of what the system should do. The Specification of the Reference Model (SoRM) involved building a *reference model* that identifies the internal structure of the system, or in other words the *"who"* of the system. More specifically, it identified the existing nodes in the network and the subsystems in each node. The Distribution of Functionality (DoF) involved building a set of sequence diagrams that describe how the required behaviour, specified in the use case diagram, were distributed among the internal parts of the system (from the reference model). In other words, it described how the distribution units from the reference model were affected by new or modified behaviour specified during SoF.

REME was supported by the requirement management tool RequisitePro. There were however, a number of key issues that had to be overcome to ensure its success. Using object oriented concepts, when the target language was a real-time procedural language (PLEX) caused immediate problems. For example, in one project alone 40 resources were required to "uplift" the legacy functional descriptions into use-cases before the new functionality could be defined using use-cases. This example and many other problems encountered raised many questions that had serious implications in the implementation of traceability: How do you trace to legacy documentation not described by UML? Is it

feasible to build new functionality on legacy systems using two different process paradigms? How do you integrate current OO processes with the older methods of PROPS and MEDAX? Can the new traceability tools deal with the size and scale of many of the large multi-site projects? The problems we observed during the early days of REME were as follows:

- The requirement management tool, Requisite Pro was PC based, while many of the developers worked on Unix platforms reducing the visibility of many of the software engineers to the requirements. Dedicated PCs had to be purchased, increasing the cost to the projects.

- Poor buy-in by some management leading to a lack of commitment from the development teams.

- Many developers continued to create legacy documents to ensure that information wasn't lost while using the new approach leading to a duplication of effort.

- Increased deployment of specialised requirement engineering consultants to bridge the competence gap leading to major increased project costs.

- Lack of understanding of some of the core principles of iterative and incremental development.

- Many new requirement engineers remained uncertain what their precise role and responsibilities were in relation to the management of the requirement database.

- The cost of the new tools, the consultants to support them, the development of the new processes and the need for extra resources greatly increased the cost of the development of each new feature.

- Testing was compromised due to the many design uncertainties leading to difficult test estimations and poor test results.

The projects continued and the processes evolved and by 2000, Ericsson released the Ericsson Unified Requirement Engineering Process (EUREP). EUREP encapsulated best practices from REME, RUP and internal best practices from the early waterfall models or from new "tried and tested" successful practices that emerged. EUREP was intended to provide Project Managers and Requirements Engineers with a standardised and efficient means of handling requirements through the full project development life cycle. In particular the process was intended to encourage and support the handling of requirements in an iterative approach. There were some immediate shortcoming of EUREP, in that; it didn't support traceability in the entire product development lifecycle. Once again the success of EUREP was dependent on the budget made available to projects for its implementation, the availability of essential requirement engineering resources and the willingness to embrace change by the different management teams.

During 2001 and 2002 the telecom crisis hit Ericsson hard. Staff numbers were reduced drastically from 107,000 to less than 60,000 in over a year. The new problems faced during this period included:

- Repressed IT budgets impacting the renewal of contracts for expensive third part supplier licences

- Less time and money for much needed training.

- Fewer resources for tasks like the management of the requirement databases and change management leading to a *traceability crisis* within Ericsson.

It was into this development environment that the author commenced his career at Ericsson, being involved in the definition and rollout of REME. This section sets the historical context before looking at the technological context that the case study was set.

## 4.3 TECHNOLOGICAL CONTEXT 2003

Before describing the traceability situation it is important that the reader understands the technical context of this case study. In this section we give a brief overview of the evolution to 3G standards, we briefly describe the 3G architecture and we introduce the Operation System Support (OSS). It is the OSS component that this study was based.

### 4.3.1 Evolution to 3G Standards

While the first mobile phones appeared as early as 1956, they availability did not take place until the 1980s and since then the mobile technologies have advanced beyond imagination in terms of coverage, services, technology, handsets and regulation. Perhaps the most revolutionary change is that, within 20 years, the number of mobile subscribers has surpassed that of fixed-telephone line subscribers, making mobile technology the predominant means of voice communications. (Minges, 2003) First generation (1G) mobile cellular networks employed analogue technology. Developments in digital technology led to second-generation (2G) systems. As early as 1990 in Scandinavia, 2G networks had been developed to provide better quality services, greater capacity and additional functionality over analogue systems. As illustrated in Figure 4-4 below, *International Telecommunications Union Mobile Subscription Statistics*, at the end of 2002, the world had almost completed the transition to digital cellular networks, with analogue users accounting for a mere three per cent of total mobile subscribers. (Minges, 2003)

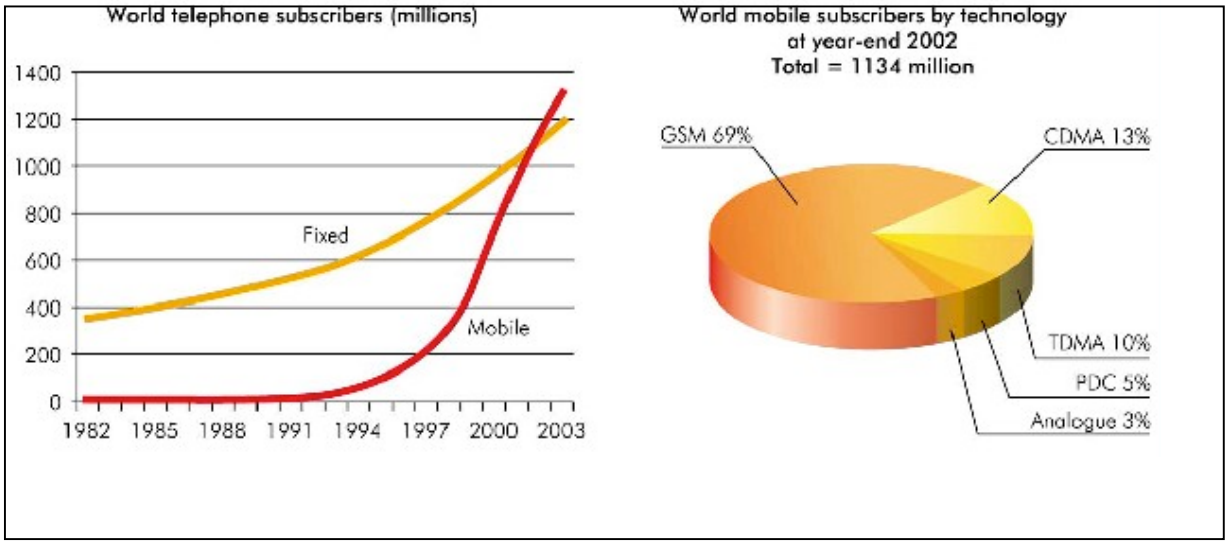



**Figure 4-4 International Telecommunications Union Mobile Subscription Statistics**

The market for mobile communications has grown explosively since the introduction of 2nd Generation (2G) digital systems. End users have become gradually used to multimedia communications with ever growing demands for more bandwidth and better Quality of Service (QoS). 2G systems evolved to satisfy the demand for wireless access to Internet applications, but did not address the demand for global access. The reason for this was that there are several second generation systems which use incompatible radio technologies, on different frequency spectra, and therefore did not support one ubiquitous standard.

What was needed, in order to support new services, was a higher capacity on the radio links, as well as compatibility between systems to provide seamless access worldwide. This led to the concept of third generation systems (WCDMA/UMTS), which expand the range of options available to users and allow communication, information and entertainment services to be delivered via wireless terminal. WCDMA/UMTS is a system where the telecom, computer and media industry converge. To support the required high bit rates for these third generation networks, a new radio technology, Wideband Code Division Multiple Access (WCDMA) is used in the WCDMA/UMTS standard. For the introduction of UMTS (Universal Mobile Telephony System), a new radio network is added to the existing GSM Core Network. The open architecture of the UMTS Core Network ensures smooth migration from the existing 2G systems to the technologies of the future. For operators that already have GSM, UMTS networks are built on top of an enhanced GSM Core Network.



### 4.3.2 3G Architecture

A UMTS network consists of three interacting domains:

1. **The Core Network** (CN), the main function of which is to provide switching, routing and transit for user traffic. The Core Network also contains the databases and network management functions. The basic Core Network architecture for UMTS is based on GSM network with GPRS.

2. **UMTS Terrestrial Radio Access Network (UTRAN)**. The UTRAN provides the air interface access method for User Equipment. All equipment has to be modified for UMTS

operation and services. The Base Station is referred as Node-B and control equipment for Node-B's is called Radio Network Controller (RNC).

3. **User Equipment (UE)**: is any device used directly by an end user to communicate.

### 4.3.3 Operation Support Systems

The above information illustrates that there was a growing network complexity introduced by, among other things, the generation shift from 2G to 3G, the increasing number of nodes and information, the high rate of change and the integration of GSM/WCDMA/UMTS. This complexity caused many problems for network operators. The growing network and service complexity made traditional operation and maintenance of the network more and more costly at a time when cost reductions was crucial for the survival of the operator. One of few existing options for an operator to solve this problem is to invest in telecom management systems, an investment that would long term reduce the cost of managing a network to a reasonable level.

By 2007, Ericsson's network management portfolio, OSS-RC (Operation Support System-Radio Controller) is today the main *product-line* for day-to-day GSM/WCDMA radio and core network management tasks. Operations Support System (OSS) which traditionally, refers to the network management product-line that handles workflows, management, inventory details, capacity planning, and repair functions for service providers using the 3G network. The network operator monitors and controls the network through the OSS, which offers cost effective support for centralized, regional and local operations and maintenance activities.

However, one OSS was not Ericsson's original strategy. As shown in Figure 4-5 below, *OSS: Radio OSS, Core Network OSS, GSM-OSS*, in 2002 Ericsson OSS product family consisted of three separate product lines namely; RANOS (Radio Access Network OSS), CN-OSS (Core Network OSS) and the GSM OSS product-lines. This was called the *"three to one"* strategy. (see Figure 4-6 below, *OSS "Three to One" Product Strategy*)

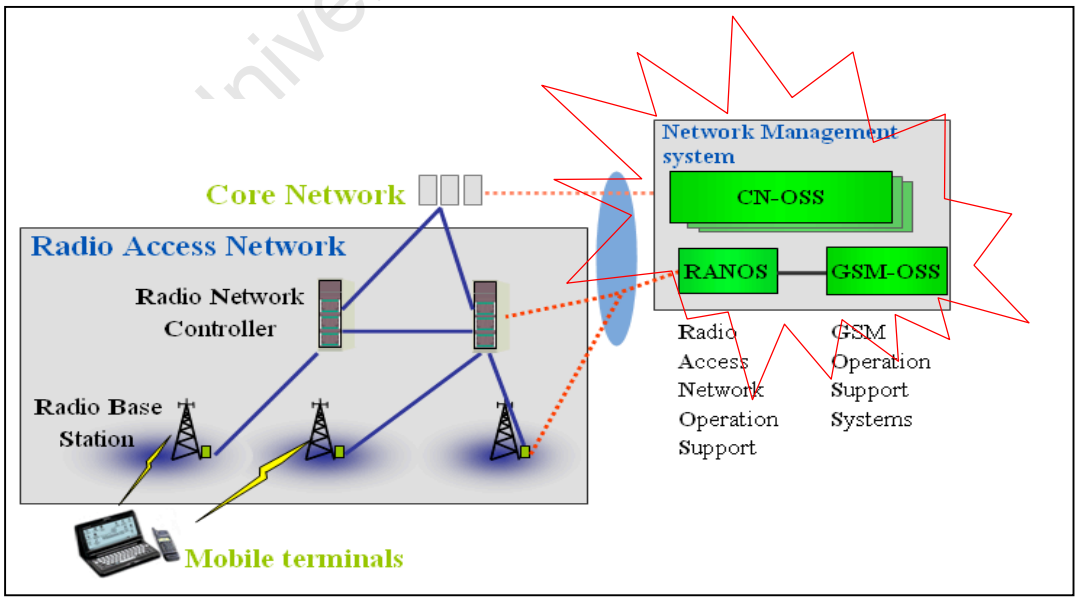



**Figure 4-5 OSS: Radio OSS, Core Network OSS, GSM-OSS**

As shown in see Figure 4-6 below, *OSS "Three to One" Product Strategy,* this strategy involved major organisational and project re-structuring. In 2002 there were sixteen design houses, with fifty subprojects in the development of the three separate OSS's product lines. In 2003, the Product Development Unit (PDU) for all OSS-RC development became the responsibility of Ericsson's LMI, Athlone, Ireland. It is not difficult to imagine the problems and complexities that such a strategy had on the implementation of traceability. All facets of the product development lifecycle were impacted causing all sorts of traceability issues to be overcome.

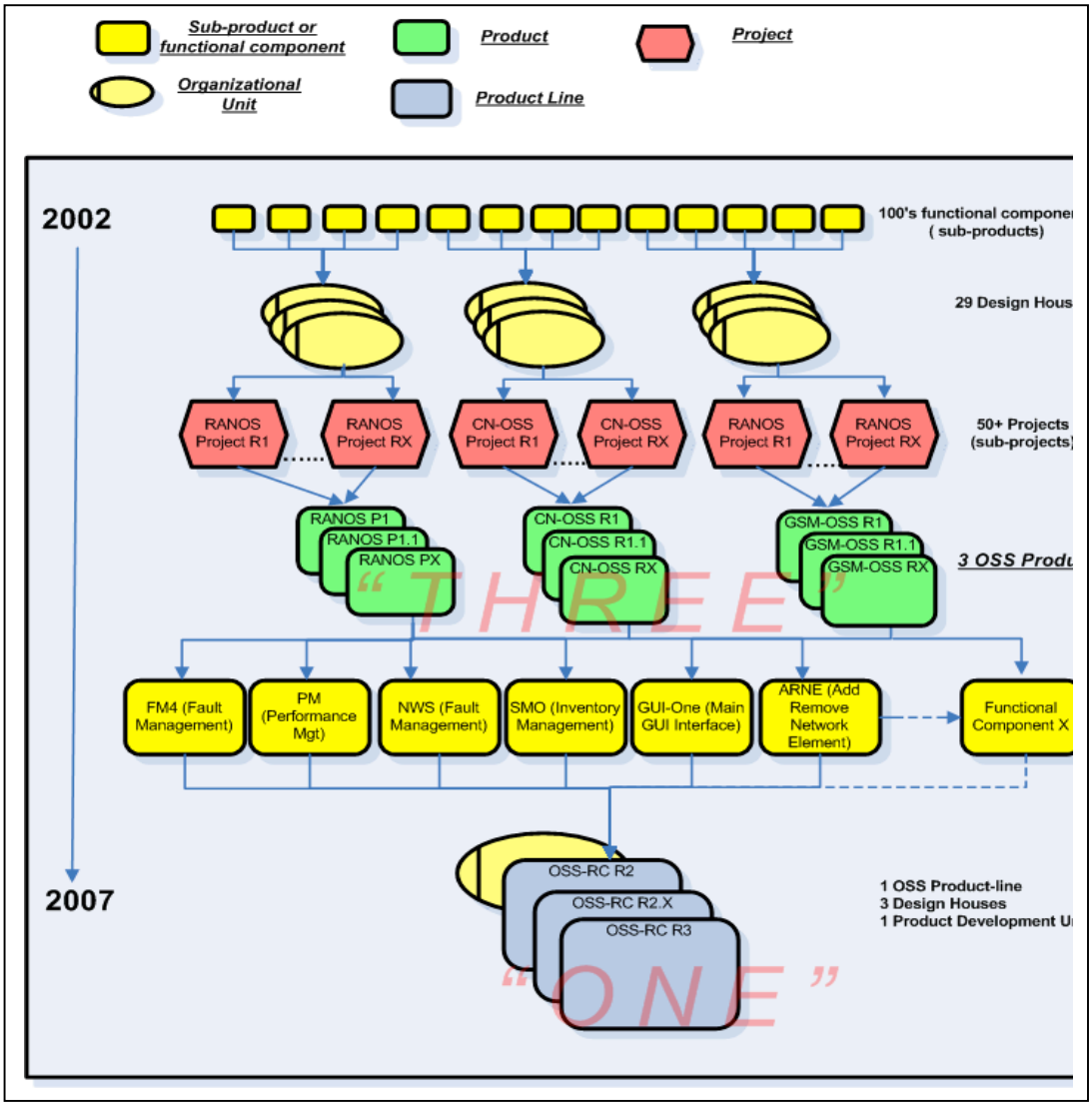



**Figure 4-6 OSS "Three to One" Product Strategy**

## 4.4 RESEARCH DESIGN CONSIDERATIONS & RESEARCH METHOD

### 4.4.1 Research Method

The research method that we utilised throughout this case study is shown in the Figure 4-7 below, *The Case Study Research Method*.

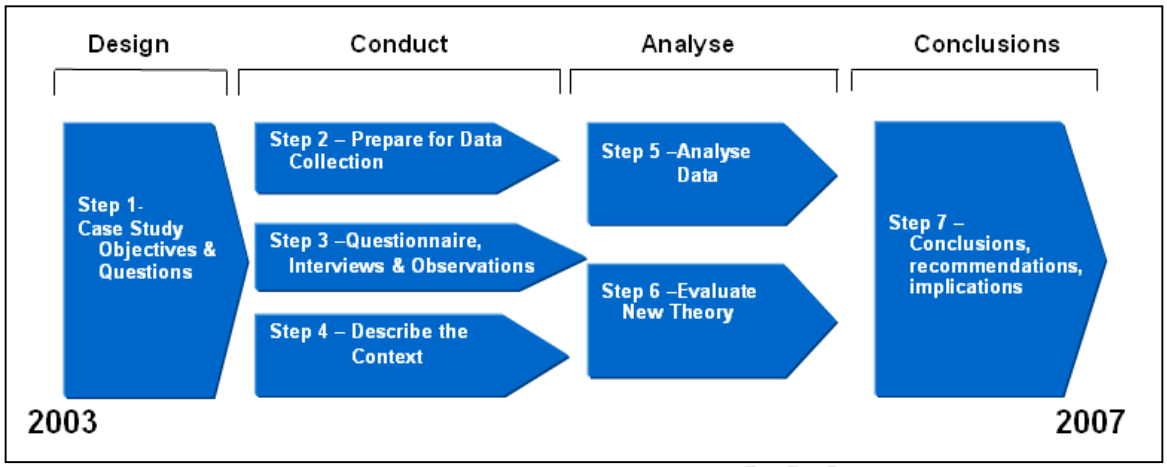



**Figure 4-7 The Case Study Research Method**

Yin suggested that the researcher must possess or acquire the following skills: the ability to ask good questions and to interpret the responses, be a good listener, be adaptive and flexible so as to react to various situations, have a firm grasp of issues being studied, and be unbiased by preconceived notions. (Yin, 1994)The investigator must be able to function as a "senior" investigator. (Feagin, 1991) With the researchers background as described in Chapter 1 we were certain that we had the required skills for this particular case.

### 4.4.2 Step 1: Objectives

As described earlier, the primary goal of the case study was to advance our understandings of traceability by documenting the factors that influence traceability from an organisational and human perspective during a four year extensive study. The objectives of this study was to accelerate the widespread use of software traceability practices by assessing and evaluating, traceability practices, to build an experimental traceability framework, and to enable software development organizations to make measured improvements in their traceability practices by working with the results and findings of this report. These objectives can be further divided into the following questions:

- Objective 1: Understand all that factors that influence traceability in the context of the development of the OSS product line and contribute to the overall traceability body of knowledge.

- Objective 2: Analyse and interpret the changes to these factors while presenting the results over a four year period. (2004-2007).

- Objective 4: Compare the results gathered in the case study with the results from the survey.

- Objective 5: Use the empirical data in the design and test of the Traceability Solution Framework.

### 4.4.3 Step 2: Prepare for Data Collection

The principal data collection methods for this case study were direct observation, participant observations, interviews (formal and informal, structured and semi-structured), questionnaires and the physical project artifacts. The researcher kept a diary of events that also included additional notes of the researcher's reflection about certain events and activities.

In order to build up a historical picture of project experiences at Ericsson, the investigator studied a number of key assessments completed by a Key Project Assessment (KPA) team. These assessments gave us a clear understanding of the common problems identified in projects and assisted us to identify *emerging patterns* with regard to requirements and traceability. Other documents accessed for information included process descriptions, requirement management plans, traceability matrixes, tooling guidelines, training material, mentoring documents, system descriptions and all artifacts produced along the product development lifecycle. We also referred to tooling reports, project metrics reports, or any other physical evidence that were provided as part of the field visits. Using triangulation techniques on the data, the documents served to corroborate the evidence from the interviews, questionnaires and observations. The investigator had to be careful while evaluating the accuracy of the records before using them. In some cases documentation that had not gone through Ericsson's review process were taken from the evidence gathered. The documentation provided a solid foundation to further the researchers understanding of the current practices and they also provided a roadmap of the changes that took place in a sequential manner. For example, for each new product release of OSS-RC, a new Requirement Management Plan was created. Each plan describes the traceability items, their attributes and relationships, the software engineering roles, their responsibilities, the necessary tooling information, the contact information of the main roles involved in the process and any other information deemed useful to describe the traceability situation. By analysing each RM Plan, from OSS-RC R2 (2003) to OSS-RC R6 (2007), we gained a chronological insight into the changing practices of each project. For example, the OSS-RC R3 Requirement Management plan uses Requisite Pro and EUREP terminology to describe the traceability situation. However, the OSS-RC R4 RM Plan mandates the use of MAR's[10] as the traceability tool with considerable changes in to the terminology. Each plan outlined the contacts of the most important roles involved with traceability which was very useful during the planning of the interviews.

The relationship between the researcher and the telecom engineers is of significance here: the question being whether the researcher was observing participants without participating, or is acting as an observer who is participating.

Since the purpose of the observation was that the researcher should learn the perspectives of the individuals being observed and the context in which their traceability activities occur, the researcher was unable to observe in an inconspicuous manner. While the behaviours which were observed required both low and high levels of inference, the majority required some level of judgment on the part of the observer.

[10] MAR's is the propriety traceability tool which was introduced in 2005. We describe MAR's in much greater detail in Chapter 10

The researcher visited the engineers regularly during the four years of the study. The researcher took a soft-line position as participant observer in that the need to be there as an observer was recognised but the researcher did not feel constrained to share in the activities in a direct and complete way (Burgess, 1985, p25). The engineers were followed during their daily activities, with notes taken of what they did, when, with whom, and under what circumstances. After each interaction the researcher queried them about the meaning of their actions. In particular the researcher shadowed the requirement engineering staff for one day a week over a four month period.

Feeding back research findings to the engineers promoted enthusiasm for the research project and the engineers came to accept the researcher as someone genuinely interested in how they functioned as scientific investigators and with a concern to identify those aspects of the task that hindered their progress. They were never put in a position of having personally identifiable data fed back to their managers, anonymity and pseudo-anonymity having been explained to them very clearly and scrupulously adhered to. The requirement engineers in the later years of the study, in particular, developed a familiar relationship with the researcher: helping to collect data when the researcher was not on site, collecting process models, relevant documents and drawing the researcher's attention to any new patterns that emerged.



### 4.4.4 Step 3: Interviews

Interviews were the most important sources of gathering "inside information" for the case study. We used several forms of interviews: Open-ended, Focused, and Structured. The interviewer began with a brief presentation of himself and the research leading to the interview. The interviewer continued with confirming the interviewee's position in the organisation and their roles and responsibilities within the organisation. The tasks were investigated with questions like: "What do you do?" and "What are your responsibilities?"

In an open-ended interview, key respondents were asked to comment about certain events, for example, what were the problems they encountered with traceability? They often proposed solutions or provided insight into events, that added value to the research effort. In some cases they corroborated evidence obtained from other sources like the RM Plan. With this type of interview we were conscious of the need to avoid becoming dependent on a single informant, and so we sought the same data from other sources to verify its authenticity.

The focused interview was used in a situation where the respondent was interviewed for a short period of time, usually answering set questions. For example: what problems have they have encountered with the traceability tool, or what do they think of the processes?

The structured interview is also one of the techniques used in the industrial survey. The questions were developed in advance. The researcher was aware of the need to avoid subjectivity and bias and often times used a system of semi-structured interviews. The interviews were set up to be an interactive dialogue, with opportunities for both the researcher and participant to seek shared understanding (Lather, 1991). Near the end of the study, interviews were more informal and unstructured. The researcher conducted all the interviews.

Altogether 27 interviews were performed. Each interview took from between 0.5 and 2 hours, depending on the particular role the informant had in relation to traceability.

| Role | Number Interviewed | Role Description |
|---|---|---|
| Product Manager | 1 | The individual within an organisation responsible for the day-to-day management and welfare of the OSS product or family of products at all stages of their product lifecycle, including their initial development. The product manager interfaces with the stakeholders and customers.<br><br>The product managers played a significant role in the definition of the product requirements. Their tasks included the creation of product requirements, participation in the change control process (attending Change Control Board meetings), ensuring that impact analysis was carried out, communicating with the stakeholders and customers in relation to any aspect related to the definition, development and rollout of the OSS-RC product. In 2004, they ensured that the requirements were entered into RequisitePro and it was their responsibility to check that these requirements were kept up to date.<br><br>It was noted in 2004 that while the product managers were responsible for the product requirements, they did not use RequisitePro relying heavily on the requirement engineer to ensure that their requirements were kept up to date. |
| Project Manager | 3 | Is responsible overall for planning and execution of all project activities, including those for requirements engineering. As OSS-RC was divided into system and sub-system there were a number of different project managers from total project manger to sub-system project manager. There were also project managers for specific groups with the software development lifecycle. For example, the systems project manager was responsible for planning, tracking and reporting on the status of the analysis of the work package of OSS-RC.<br><br>In 2003, the total project manager used a project cockpit tool to gather the latest metrics from the project. For example, a spreadsheet of test cases was provided by the testing department to the project manager. As each tester executed the test case, the tester changed the status of the test case from “not started” to “complete”. The tester would then check each test case against the compliance attribute assigned to each test case, changing it to “compliant”, “not compliant” as appropriate. Each evening the total project manager imported the test case results into the project cockpit, which would subsequently give the project metrics on the status of the project. For example, the progress of the testing effort, the number of test cases executed per day, the compliance of the product to the compliance statement and the stability of the system.<br><br>In general the project managers attended the Change Control Board meetings, and interfaced with all project team members as necessary. They did not in any way interface with the traceability tools except to extrapolate the metrics as described. It should be noted at this point that in general the status of the requirement database was not in sync with the current requirement situation. There were many reasons for this but it is suffice to say at this point, that the main reasons were the lack of resources to manage the requirement database, the magnitude of requirements, the poor performance of the traceability tool and the lack of an interface with the project cockpit tool. |
| Systems Architect | 5 | Is responsible for defining the system architecture for products being developed, while interfacing with the users and stakeholders and all other stakeholders in order to determine their (evolving) needs. They generally analyse requirements from the business or market unit generating systems requirements and their attributes like cost and schedule.<br><br>The primary role of the systems architect in the traceability process was to ensure that systems requirements were in the requirement tool, that they were traced down to the lower level requirements and to take responsibilities to carry out impact analysis. However, as we discovered very few systems engineers were interfacing |

| | | |
|---|---|---|
| | | with the requirement database in 2004. |
| Requirement Engineer | 4 | Participates in all requirements related activities and is responsible for ensuring that all traceability activities throughout OSS-RC development are completed. In 2004 this was a relatively new role in the OSS-RC domain. Their responsibilities included the creation of the RM plan, attending the Change Control Board meetings where they administered the traceability tool making changes in real-time at the meetings, ensuring the RM process matched the needs of the development organisation, ensuring that the requirement database was kept up to date and that all aspects related to the tool were kept up to date. In 2003, there was only one requirement manager for the whole Athlone site. While their role was clearly defined the magnitude of the work made it very difficult to ensure that all activities were executed properly. This resulted in the requirement database not reflecting the most recent changes to the requirements. |
| Configuration Management | 2 | One of the key purposes of configuration management is to control changes made to the software product. The Configuration Manager is responsible for the management of the configuration of the OSS product which including design configurations, ordering, shipping and customer usage to ensure conformance to the desired configuration. In our case the configuration manager controls the process of identifying and defining Configuration Items in a system, recording and reporting the status of Configuration Items and Requests for Change, and verifying the completeness and correctness of Configuration Items.<br><br>Configuration management, as a discipline for supporting software development, has been around for half a century and has evolved into standard practice within traditional software development processes.<br><br>In 2003, the configuration manager ran the Change Control Board meetings and managed the handling of the change requests in the traceability process. |
| Senior Designers | 1 | Senior designers generally have 5 years or more experience in the development of products and are adept at taking briefs and usually have a lot of client liaison experience. A senior designer should have been involved in a number of iterations of projects. In Ericsson's senior status is awarded after an interview and assessment process.<br><br>It was observed in 2004, that very few of the senior designer's accessed Requisite Pro instead using lower level requirement specification documents, architectural specifications or function descriptions in their everyday tasks. |
| Designers | 3 | The designer defines the responsibilities, operations, attributes, and relationships of one or several classes and determines how they should be adjusted to the implementation environment. In addition, the designer may have responsibility for one or more design packages or design subsystems, including any classes owned by the packages or subsystems.<br><br>In 2003, the designer's role in the traceability process was negligible. In most of the cases the designers used lower level documentation, including use cases, UML structure diagrams, low level requirement specification documents and low level function specifications. |
| (System) Testers | 3 | Write and/or executes tests of software with the intention of demonstrating that the software either functions or not. The system tester's role also includes the facilitation of the approval and certification of final functional application release into Technical Stability and Acceptance Test. They work closely with the Systems Architects and system leads from other functional areas to provide environments for execution of the test plans, scenarios, and scripts and also to review modifications, interfaces, conversions and change requests and develop reuse and execution procedures as needed for the test environments.<br><br>The testers in some (sub) projects had their test cases set up in the Requisite Pro |

| | | |
|---|---|---|
| | | Attribute Matrix, with their primary responsibility of changing the status if the test case attributes from "not started", "to complete" or to whatever status the test cases were at. As stated above the project manager in some cases took a download from the master test spreadsheet which was imported into a project cockpit for getting metrics on the status of the test effort and stability of the system. Traceability to requirements in general was carried out on spreadsheets rather than in the traceability tool. |
| Methods & Tools (M&T) | 4 | Involved in the development and evolution of the processes used during product development. In some cases the methods and tools also worked as tool-smiths building or configuring tools to meet the users needs. However, we noted that they had little competence with Requisite Pro.<br><br>The methods and tools department supported the OSS-RC projects by evolving the processes and ensuring that the tools work efficiently. During the interviews in 2004, the M&T resources encountered a lot of problems with the different versions of Requisite Pro being used by the different departments. For example, if a version of the Requisite Pro database is opened by a newer version of the tool than it was created by, then the database and structures are updated to be compatible with the new version of the tool. However, if the original owner wishes to open the database then they must be upgraded to the newer version of Requisite Pro. In 2004 version control of the traceability tool was causing major problems for the M&T engineers. |

**Table 4-1 Software Engineers Interviewed**

We will see in the concluding section the results from the interviews we carried out in 2004, using the above categories of software engineers.

## 4.5 CORPORATE GUIDANCE

In 2004 Ericsson's corporate guidance came from the corporate Methods and Tools group, which mandated the overall strategy for methods and tools to be used by all the product development units, and issued directives to be followed by each unit. Corporate Methods & Tools did recognise that each project situation is different and therefore only gave directives as a general framework to be followed.

A traceability strategy that was successfully employed in the past cannot be expected to remain valid forever. Different projects require different requirements types and/or different relationships for traceability. With such a wide variety of different organisations available to develop, manufacture and build software products, each product development unit acted as an independent site. Moving from an in-house development project to an integration project with external vendors is likely to introduce new deliverables into a system. However, whilst projects differ greatly, having a meaningful and effective traceability strategy enables development units to ensure they are not reinventing best practices or investing valuable resources into studies already completed at a corporate level.

The Ericsson traceability strategy describes the level of traceability that should be defined in the software development process. The strategy directive is sub-divided into a number of critical categories:

- *Product Numbering and Storage:* Each new product or feature must be numbered according to the corporate numbering strategy. This is an essential rule that all product development units must abide by.

- *Requirement Numbering:* Describes the requirements numbering convention that should be used.

- *Repository Structure:* Describes how the requirements should be structured in the requirements management tool or repository and the relationship, interface, or dependency on data in other tools. The repository structure is based on the traceability strategy.

- *Process Definition:* Describes what the processes of the development organisation and the expected process interfaces with third party suppliers should contain. The mandate does not specify which process to use, rather what it should contain.

The directive is two-tiered, and divided into product traceability and development unit traceability policy. The overall policy for marking and traceability systems states that Ericsson and contracted suppliers must have a system that supports Ericsson's processes for marking and traceability. A number of basic rules apply, governing: local (i.e. legislation, regulation or customer) marking and traceability requirements; cost considerations; documentation; and markings and traceability data for the Ericsson central database for traceability. In addition the policy sets out some 17 rules that should be followed which are summarised for convenience in small print in the list below. It is not necessary for the reader to scrutinise them closely; only to see the sort of information that they cover. The reader will see at once that these rules have grown out of the practice of part numbering from the days when there was a simple one-to-one relationship between analogue PCBs, each having a specific function, and their requirements.

1. All product numbers used must be centrally registered in the Ericsson Product database. An Ericsson product is an entity that has been declared as a product, e.g. identified with product number, defined by means of documents, revised according to established change rules and can be delivered to customer or used within Ericsson.

A refined general standard product must, when Ericsson unique requirements/demands on its functionality is applied on the product, be treated as an Ericsson product in terms of marking and traceability. Ericsson's traceability is based on Ericsson product identity marking. The product information in traceability systems should correlate to the information on the label used for product identity marking. To set the level of product traceability a code shall be set for the traceability requirement. The code exists in four levels, no traceability (N), batch traceability (B), individual traceability (Y) and individual traceability on included products (S).

2. Traceability requirements are applicable for products included in Ericsson product portfolio.

3. Purchased products shall follow the same rules whenever Ericsson traceability requirements apply.

4. It is the responsibility of all Ericsson supply and service units (internal and external) to insert and modify data in the central traceability database.

5. Traceability requirement on products must be set and documented, so that products can be marked accordingly, e.g. unique serial number used only when individual traceability is required.

6. Traceability information (e.g. serial number information) shall be sent, to Ericsson's database for Traceability, at latest when products are delivered.

7. Product management and development units are responsible for implementation of existing markings, e.g. to avoid creation of duplicated markings.

8. Product management is responsible for decision if different markings for different local requirements shall be used.

9. Sourcing is responsible for that document and system requirements are included in agreement with suppliers.

10. Sourcing and Supply are responsible for verification that suppliers have needed systems and documentation for manufacturing Ericsson products.

11. Sourcing contact at Ericsson is responsible for verification that product documentation exists when to order a product. If not, it shouldn't be possible to place an order of a product

12. Ericsson supply organization is responsible for their documents and system requirements for marking and traceability are included in transfer of product to external supplier projects.

13. Each Ericsson unit and external supplier is responsible for implementation and maintenance of a marking and traceability system.

14. Each Ericsson unit and external supplier is responsible for that created markings are both visual and machine-readable.

15. Each manufacturing unit delivering products must connect serial numbers to a customer or manufacturing order.

16. Each Ericsson unit and external supplier is responsible for sending in traceability data, in time, to Ericsson central database for traceability.

17. Customer order responsible must verify that information linked to generated manufacturing orders are linked to the original customer order.

While the directive gives clear guidelines on the traceability rules that should be applied, most people in the development unit were not aware of this directive. Many of these rules are so implicit to any development project that few projects needed to refer to this document at all. The directive would however, be useful to any new design house or act as a very important reference if entering into contracts with third party suppliers or out source partners. In OSS terms this directive had little or no consequence to any of the traceability practices undertaken. Many would argue that clearer or more enforced directives from corporate would lead to greater conformance to company policy.



## 4.6 REQUIREMENTS & TRACEABILITY PROCESS (2004)

This section describes the processes employed by Ericsson in 2004 under the following headings:

- Requirements Management Process
- Requirements Management Plan
- Change Control & Configuration Management Process

### 4.6.1 Requirement Management Process

In 2004 the OSS Requirements capture team used the Ericsson Unified Requirement Engineering Process, (EUREP) as its underlying process. The OSS-RC Requirement Process is intended to provide software engineers with a standardised and efficient means of handling requirements through the full project development life cycle. In particular the

process is intended to encourage and support the handling of requirements in an iterative development context. As shown in Figure 4-8, *The EUREP Workflows*, EUREP was defined with work flow diagrams, workflow detail diagrams, activity diagrams and process guidelines at different levels of granularity.

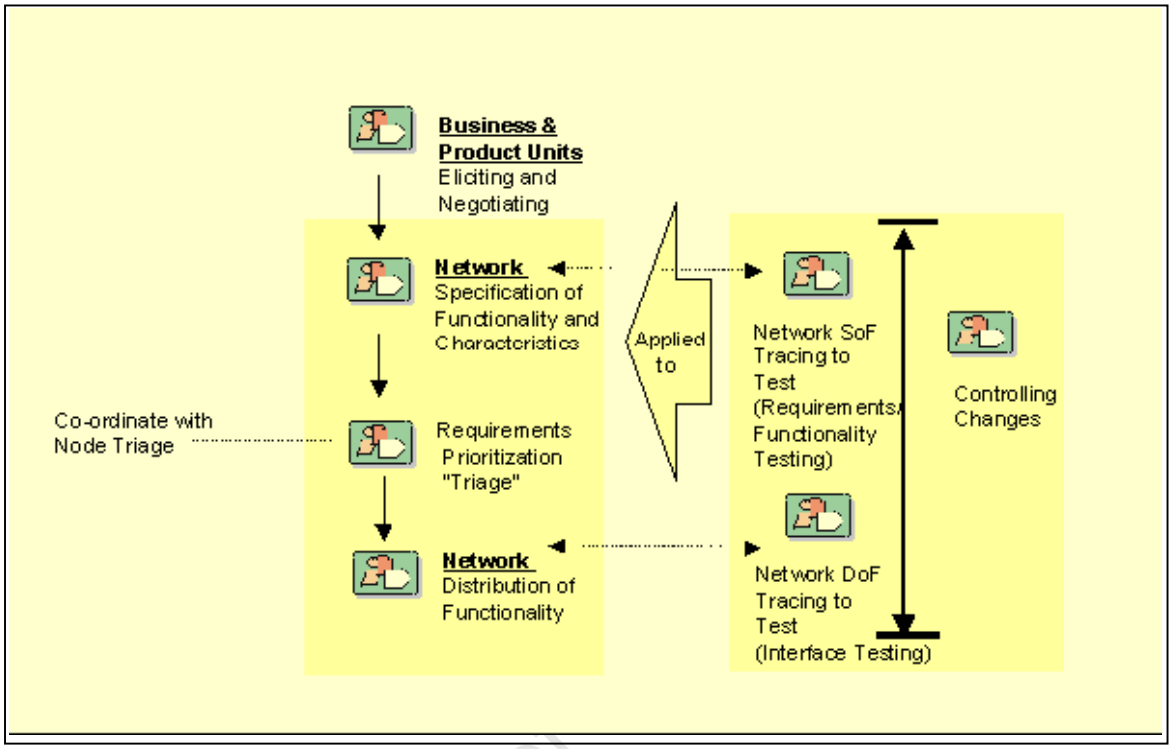



**Figure 4-8 The EUREP Workflows**

The initial start point of the requirement process is the workflow diagram shown above. The requirements flow from the business units to the product units to the network team or systems team. The systems team specify the functionality or in simple terms create use cases. They distribute the functionality across the different nodes in the network which in UML terms are the sequence diagrams. The system testers trace the test cases to the network use cases. There are several workflow diagrams at different levels of granularity in the requirement management process.

A process user should drill from the high level work diagrams to the lower level workflow detail diagrams, which show groupings of activities that should be performed. These diagrams show the roles involved, together with input and output artifacts, and activities performed. The activities of a workflow are not performed in sequence. There are several workflow detail diagrams for each discipline.

In the Figure 4-9, *Activity Diagram*, we illustrate the activity diagram for Eliciting and Negotiating Requirements.

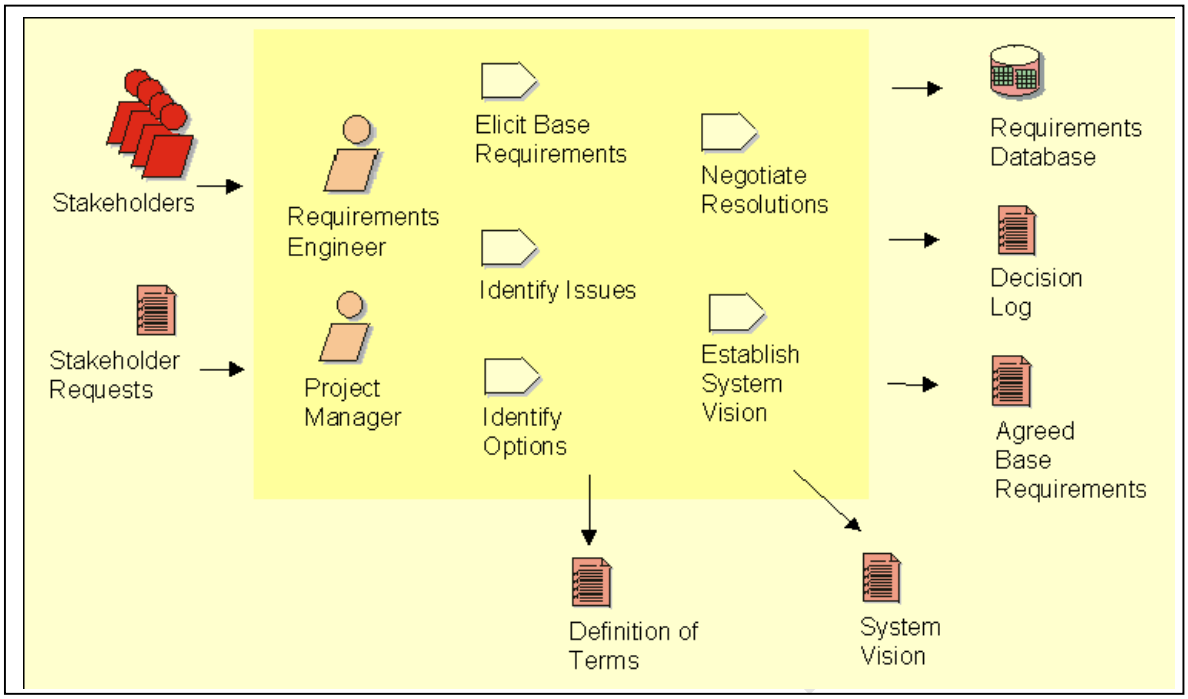



**Figure 4-9 Activity Diagram**

The activities are: to elicit base requirements, which means you need to have clear view of the stakeholders and their requirements; to identify issues means you need to have to identify conflicts, risks and uncertainties relating to specific requirements that will need to be resolved in order to establish a clear requirements baseline for the project, to identify all possible options for identified issues, to negotiate resolutions meaning to make decisions about known issues and options.

The main output artefacts are:

- the requirements database, which is the main repository of project and system requirements, their attributes, and dependencies for use throughout all requirements handling workflows;
- the decision log, which contains an audit trail of the decisions made regarding issues and options during the negotiation and resolution step the agreed base requirements contains the agreed requirements baseline for the system to be developed;
- the system vision, which contains an overall description of the system to be developed, emphasising its main functions and features, the system boundaries and the manner in which it will satisfy stakeholders' needs.

In the Figure 4-10 below, *Requirement Classification Activities*, we illustrate the workflow detail diagram describing the activities for classifying the requirements. The main objective is to define a common vocabulary in all textual descriptions of the system, especially in use-case descriptions. This allows each team member to mean the same thing when they use a given term.

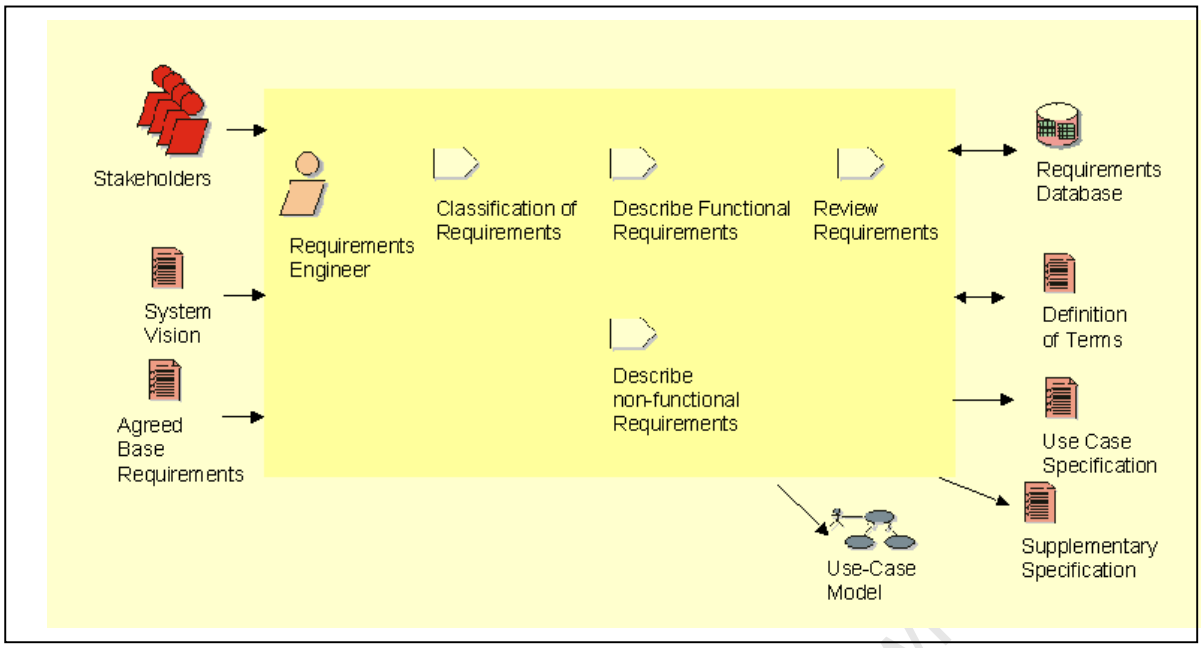



**Figure 4-10 Requirement Classification Activities**

In the Figure 4-11 below, *Trace to Functionality Activity Diagram*, we illustrate tracing the network functional requirements to the test objects providing visibility to feature coverage. It is important to trace the network level use cases to their associated test object(s) and to ensure that non functional requirements, specified in supplementary specifications, are also traced to test object(s).

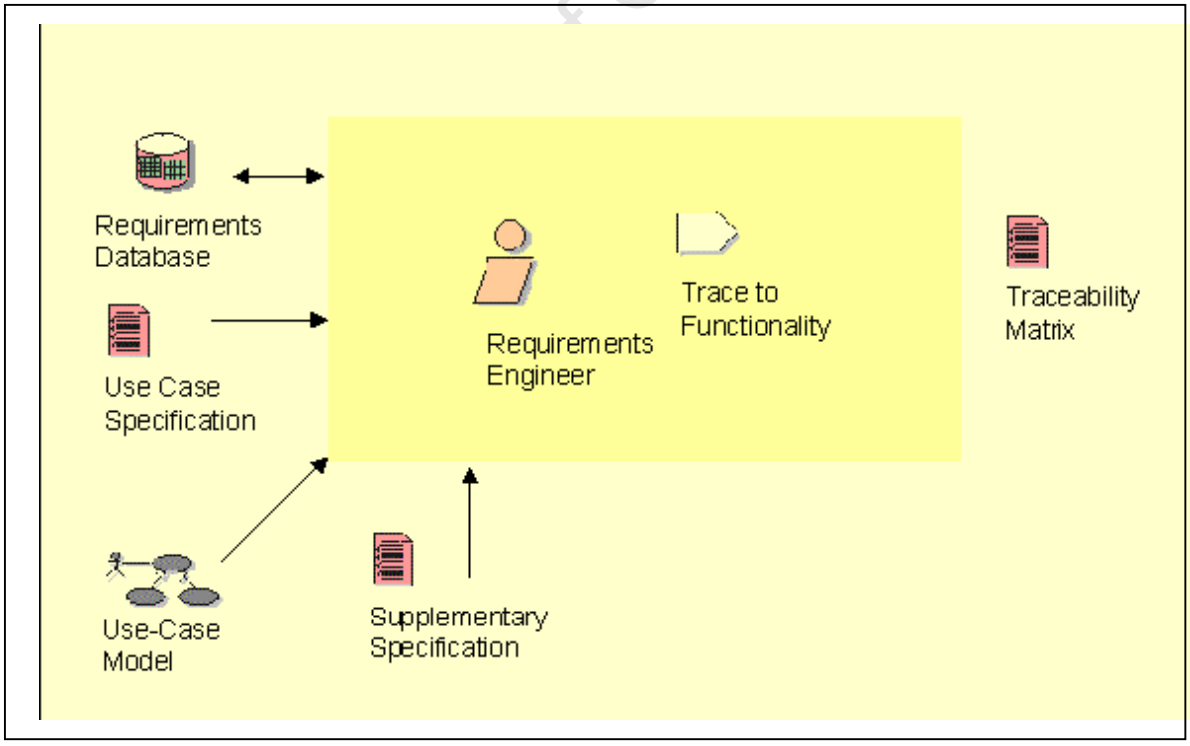



**Figure 4-11 Trace to Functionality Activity Diagram**

As stated earlier, most of the testers did not use Requisite Pro for practicing traceability. However, below is one of the few traceability matrixes that emerged in our investigations of the testing traceability effort. In the Figure 4-12 below, *Example of Tracing from Test Instructions to Test Procedures*, we illustrate tracing from the Test Instructions to the Test Procedures. We will see this particular example again in Chapter 9, the traceability patterns chapter.

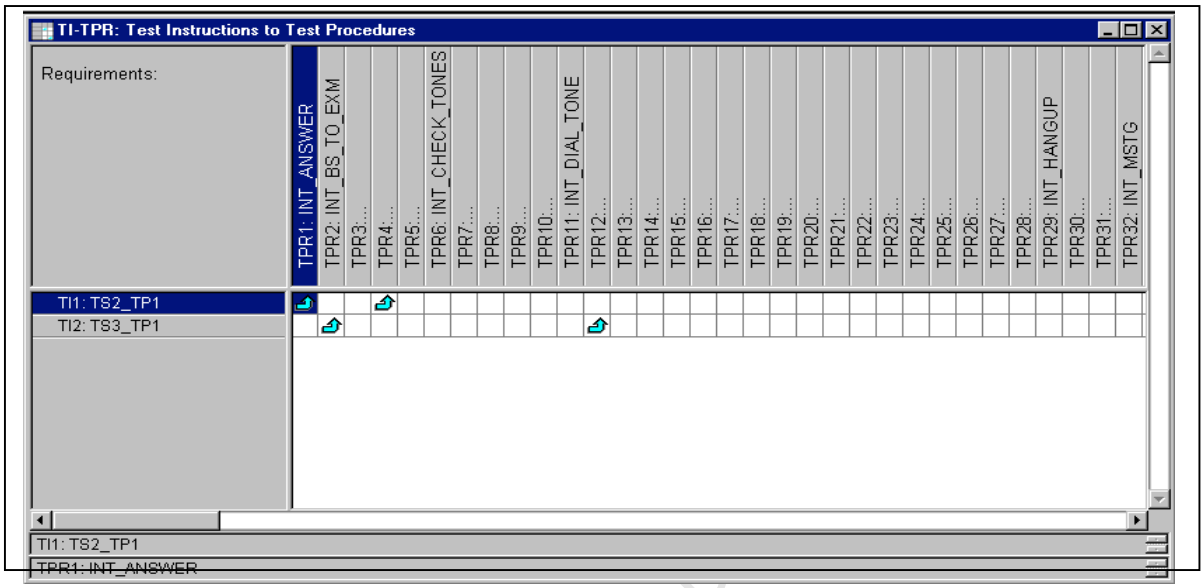



**Figure 4-12 Example of Tracing from Test Instructions to Test Procedures**

### 4.6.2 Requirement Management Plan

The Requirement Management Plan is a guideline for managing and organizing requirements for control and traceability purposes. The requirement management plan was the most important artifact in our investigations because it describes the requirement and traceability approach being followed by the OSS-RC product line. The traceability strategy is clearly described as part of the RM Plan at the start of every project. The plan addresses the needs of requirement management at both the product level and project level. The requirements evolve in both the context of product definition, and in the context of the developing project(s) of the system.

In simple terms, setting up requirements for the purposes of managing them with the help of tools, involves four main tasks:

1. Defining the different types of requirements that can exist for the system, for example the features, use cases, and characteristics.

2. Defining what additional information should be maintained about requirements of that type. For example, an attribute "Iteration" that can be assigned the values "I1, I2, E1, E2, E3, C1, C2, C3, C4, T1, and T2" and so on.

3. Filling in values for the attributes that have been defined requirements of that type.

4. Establishing the relationships between requirements of that type and other information about the system, for example the traceability of a system requirement to a product management requirement, or a test case.

In addition, in any Ericsson Requirement Management Plan, there are 3 further organisational considerations to be taken into account:

1. **Legacy Requirement Base**: In the same way that there exists a legacy design base there is also a legacy requirement base. By legacy design base we mean that there already exists a design base as a consequence of the product development in earlier projects. The legacy requirement base consists of all requirements on the product that have been implemented. The legacy requirement base is captured, managed and described in the product requirements model.

When developing a new version of the system there can be both new requirements and modifications of requirements in the legacy requirement base. These new or modified requirements are captured, managed and described in a total product requirements model.

Because of the need to maintain backward compatibility with existing systems and previous software standards, this is often a paramount consideration, which can greatly complicate things.

2. **Parallel Development**: Because of the long lead times in product development and rapid rate of technological progress in telecommunications, the development of new versions of a product often starts before the latest version of the product is implemented and delivered. This means that while there is always a single product requirements model there may be several simultaneously existing project requirements models.

3. **Distributed Development**: Because of the scale of many telecommunications projects, with several geographically separated design offices being involved in the management of all product and project requirements, which is coordinated by the System Analyst (working as part of the System Project). This implies that all projects and sub-projects involved in this process should employ the same tools, for example, the RequisitePro and Rose.



### 4.6.3 Change Control & Configuration Management Process

Once a system has passed through a development cycle and has been released to a customer(s) any further development is the process of conforming the system to the new or changed requirements. It is then said to be under Configuration Control, in which it remains throughout the system's lifecycle.

A Baseline is a version of a configuration which is established at a point in time after which only controlled changes are allowed. For example, after a Baseline has been agreed changes to an item are only allowed by using a change control process where change requests are utilized.

Configuration Control is the activity where the Configuration Manager (CM) controls changes made to a new or modified revision of a traceability item or a configuration that has been added to baseline. After a configuration has been *baselined*, changes are not only managed, they are also controlled via a formal decision procedure. For example, the level of control has been raised to project level. Configuration management is about controlling changes after establishment of a Baseline, and decisions on configuration are usually taken by a Configuration Control Board (CCB).

The Configuration Control Board, CCB, is a group of technical and administrative experts with the assigned authority and responsibility to make decisions on the

configuration and its management. The group is responsible for evaluation and approval, or rejection of proposed changes to Configuration Items. The group is also responsible for approving Configuration Baselines.

In the Figure 4-13 below, *Change Control Process*, we illustrate an extract from the Ericsson Requirement Management (OSS-RC R2) process for controlling changes. The purpose of this activity is to approve or reject change requests based on the impact they have on the project.

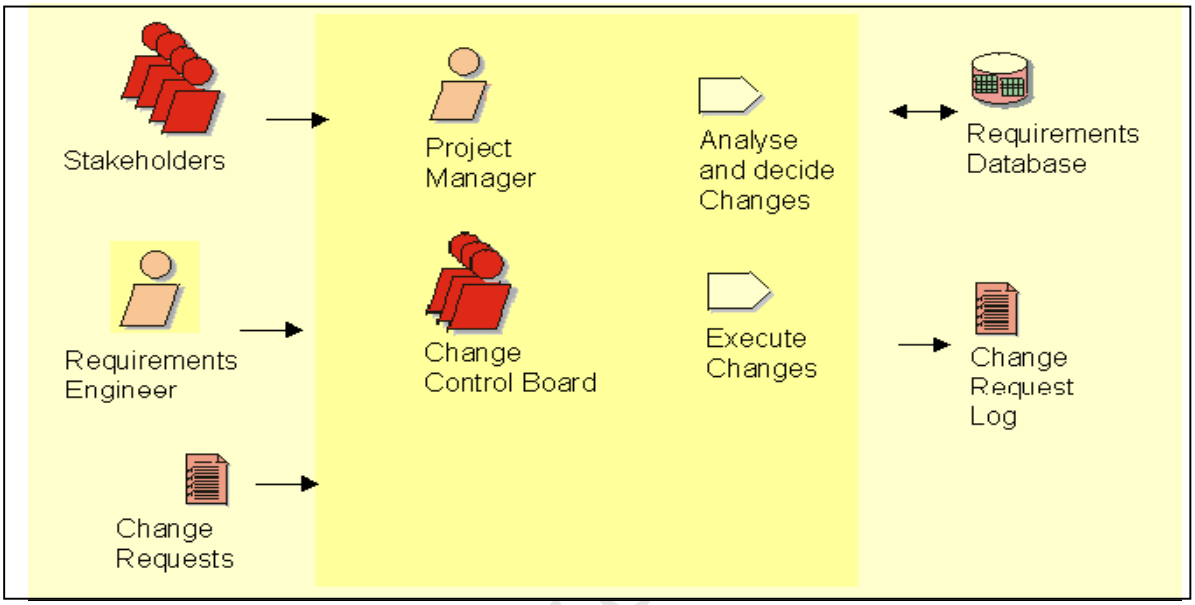



**Figure 4-13 Change Control Process**

The Requirements Engineer is the receiver of the change request and should make sure that all mandatory fields are filled out, and the text is understandable. The requirement engineer should refuse the change request if it is a duplicate, or outside the scope of the project and ensure that impact analysis is performed by an appropriate technical worker. In the development of OSS-RC, the controlling authority for changes was with the node total project, who approved changes in the following conditions:

1. There is a change to an interface or service
2. There is a change to a characteristic, for example an increased capacity.

In the Figure 4-14, *More Change Control Process,* we illustrate the flow of the change requests as documented in 2004. The total project takes change requests from the different projects (RANOS, CN-OSS, GSM-OSS and CDMA, which are then analysed by the System Product Council. Each change request is then distributed to the subproject via the System Product Council.

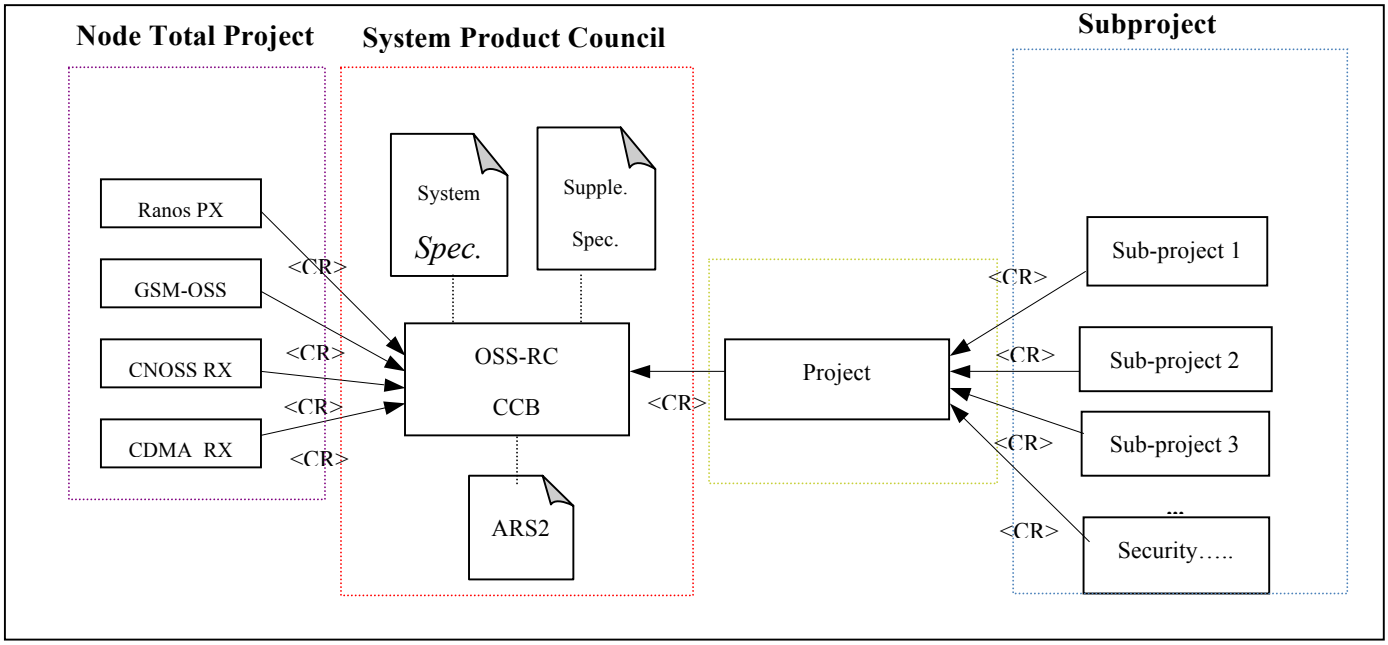



**Figure 4-14 More Change Control Process**

A Change Request (CR) should be sent to System Product Council OSS-RC in the following change conditions:

- Change spans product releases
- Issue with project budget or scope

CR's can be written on any traceability item in an approved baseline or directly on a baseline. Change requests go through a change request lifecycle:

- PREL = Draft CR
- NEW = CM reviews CR
- INVESTIGATION =CR is under investigation, see Impact Analysis
- APPR = CR is approved for implementation
- REJECTED = CR is rejected by Change Control Board
- CLOSED = CR is implemented and verified
- CANCEL = withdrawn CR

The basic change request information includes: the fields (attributes), which are project specific so the visibility, sort order and valid ranges may vary between projects. Here are however some comments on commonly used attributes:

- Title: The title of the Change Request.
- Prepared by: This is the Issuer of the Change Request.
- Proposed Change: Free text information regarding the change.
- Reason/Background: Free text information regarding the reason for the change.
- CR Category: Category of the CR

- CR Priority: Prioritisation of the CR
- URL: In this field you can enter a URL to other information relevant for the CR.
- Revision History: Revision history of the CR
- Composer: The CR writer's name will automatically be inserted by default in this field.

The change is then subjected to an *Impact Analysis*, which covers a description of the activities, cost and effort needed to implement the change in each affected artefact and product. This information is logged in the Change Request and includes at least:

- What will have to be changed?
- What alternative ways exist for making the change?
- When is the change required and whether this is feasible?
- What are the consequences of making the change (for the project, for the product)?
- What are the consequences of not making the change?
- What is the cost or saving associated with making a change?
- Will other changes supersede or invalidate this one, or does it depend on other changes?
- Are there any special test requirements?

The Change Request is then reviewed by the Change Control Board, which now has sufficient information to take an Approve or Reject decision. The decision is logged, the requirements database is updated and the Change Request status is updated to reflect the decision. Then the planned change is scheduled into the development programme. If so the Change Request is then marked as resolved, and the Change Request log is updated to reflect progress. When all activities are complete, and the changed artifacts and products are released or included in a new project baseline, the Change Request status is set to closed.

## 4.7 TRACEABILITY PRACTICE'S (2004)

### 4.7.1 Traceability Relationships

Fundamentally, traceability is about relationships between traceability items. Change requests, impact analysis and requirements are all examples of traceability items. In the Figure 4-15 below, *Tracing the Product Requirements*, we illustrate the traceability relationships that existed at the Product Level in OSS-RC R2. The stakeholders document their requirements in the Main Requirement Specification (MRS). The requirements are then distributed to the application layer, or one MRS is distributed down to a number of Application Requirement Specifications. The non-functional requirements are the documented in the Supplementary Specification which trace to the lower level use-cases.

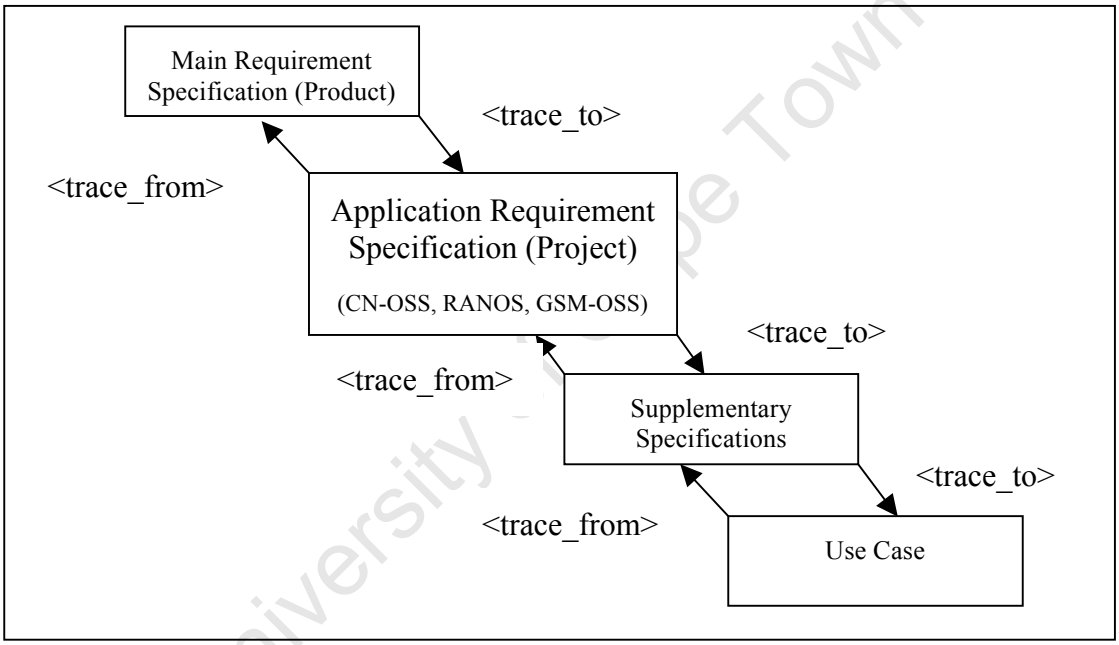



**Figure 4-15 Tracing the Product Requirements**

There are two types of trace relationships; *trace_to* and *trace_from*. For example,

- from a high level requirement to a lower level requirement, we *trace to*
- and a lower level requirement to a higher level requirement, we *trace_from*.

In addition to traceability items commonly defined as "requirements", there is a need to capture and track the traceability between many other *types* of item, with different *attributes*. These other traceability items include issues, assumptions, requests, glossary terms, test cases, change requests, impact analysis and maintenance items, to mention just a few. Capturing and tracking these other kinds of traceability item help us in effectively managing our project's requirements.

Thus, as a fundamental foundation for traceability, is the need to categorise and define the traceability items by types. A common definition of a traceability type is "A

categorization of a traceability item, for example, need, product feature, use case, software requirement, test requirement, actor, glossary term which are based on common characteristics and attributes" (Spence and Probasco, 1998)

A prerequisite for tracing requirements is a clear definition of requirement types. Requirement types are classifications of requirements at different levels of detail or abstraction with the most common being "stakeholder requests," "features," "use cases," and "supplementary requirements" Whenever you document a requirement it must belong to a specific requirement type. In order for a requirement to be traceable, it must be identified unambiguously over its entire lifecycle across different artifacts.

Below in Table 4-2, *Traceability Item Attributes*, we illustrate examples of the Attributes and associated values for the Application Requirement Specification (ARS) as defined in the OSS-RC 2.0 RM Plan. The attributes are defined and the attribute value type, for example, text or integer.

| Types & Attributes | Value Types |
|---|---|
| **Requirement Type** (ie Functional, non-functional, characteristic) | Text |
| **Requirement Identification** (ie UTRAN, Core Network) | Text, project |
| **Release Increment** (ie OSS-RC 1.0, 1.1, 1.2) | Integer |
| **Release Verified (ie 1.0, 1.1, 1.2)** | Integer |
| **Statement of Compliance** (i.e. compliant, non-compliant) | Integer |

**Table 4-2: Traceability Item Attributes**



This table describes the different attributes for the ARS, for example functional type, and the attributes that each requirement should have, for example, the project increment that it is to be developed, what release it will be verified in and the compliance attributes simply compliant or not compliant. In the second column of the table it tells us what value type is requirement characteristic should have, for example whether it is an integer or text value.

Tables like this were essential to the staff recording requirements. In order to set the requirements into the traceability tool this data was needed as a requirement creation template. In addition to the requirement type, a hierarchy or more generally, a set of valid relationships between requirement types, must be defined. This relationship defines which requirement type traces to, or is traced from, which other requirement type. The requirement types and their relationship together are sometimes referred to as the traceability strategy. The traceability strategy must be defined up-front and should be documented in the Requirements Management Plan.

There are 4 main types of requirement, with different traceability needs, as identified in OSS-RC 2.0 were:

- *Main Requirements,* which are the top level requirements for the features in the system and are defined by the Product Management.

- *Product Requirements*, based on the Main Requirements, which describe the interfaces and complete behaviour of the system from a user point of view (black box).
- *Internal Requirements*: also based on the Main Requirements and/or internal R&D improvement programs, but which are valid only during a project time frame. Such requirements often arise out of a designer's desire for flexibility and reusability of different modules.

- *Organizational Requirements*: These are non-technical requirements in the Project Specification, which cover: products to be delivered, delivery dates, projects work processes, acceptance criteria.

Product Requirements are defined by the stakeholders (UTRAN) and can comprise a number of different types:

- *Functional Requirements*, which describe all user functions and the usability of the system.

- *Performance Requirements*, which are used to specify criteria that can be used to judge the operation of a system, rather than specific behaviours.

- *Availability, Reliability* and *Maintainability Requirements*, which may be specifically related to function.

- *Environmental Requirements* that call for the system to operate in specific environmental conditions. These may be external world, such as climate, weather etc. or they may system internal or software related, such as the Operating System

- *Data Protection* and *System Security Requirements*, which are inserted at the customers request or by internal design to protect the network and software systems from interference.

- *Interface Requirements*, which in communications networks are an essential part of enabling different parts of the network and users to actually talk to each other in common message or signal formats. In telecommunications, these usually have a strong legacy inheritance.

- *Technology standards* (GSM specification etc), which may have been agreed amongst designers/industry in order to facilitate interfacing

- *Platform Requirements* that call for the system or product to interface with a host, building or vehicle of some sort.

- *Documentation and Ethnical Requirements* (Languages supported etc);

- Regulations (compliancy with country regulations and directive)

As a result of experience with the considerable functional and organisation complexity for the critical 3G projects, GSM-OSS, CNOSS, RANOS, in 2003, a Key

Project Assessment (KPA) team reaffirmed the following key processes for best traceability practice:

- Capturing requirements and finding use cases and supplementary requirements for the project.
- Detailing requirements and creating detailed specification of use cases and the numerous supplementary Product Requirements.
- Managing requirements dependencies and structuring requirements dependencies and attributes and establishing traceability links between requirements. The main purpose is for the project to be able to scope and plan the work.
- Reviewing and Prioritizing Requirements.
- Managing requirements change, for example evaluate submitting change requests and determining their impact, see that the changed requirements are handled and are consistent between product and project requirements. This has a close relation to the work performed in the configuration management activities within the project.

### 4.7.2 Traceability Tools in 2003/2004

As shown in Table 4-3, *Traceability Tools (2003/2004)*, we show the tools in use in late 2003, early 2004:

| Product | Traceability Tool |
|---|---|
| RANOS | Rational RequisitePro |
| CN-OSS | Telelogic Doors |
| GSM-OSS | Rational RequitePro |

**Table 4-3: Traceability Tools (2002/2003)**



The fact that there were two tools in use caused a number of problems, which we shall describe and discuss in more depth later on, but in addition to problems encountered with the tool suites, Requirements Engineers encountered many problems in which the tools were not to blame, although better tools might have helped. The KPA assessments found that engineers had difficulty in determining and controlling the following:

- Source of requirements.
- Ownership of requirements.
- Priority of requirements.
- Communication of requirements.
- Status of requirements.
- Impact of changing requirements on project.

Tools are not silver bullets. They do not remove the burden of having to decide on a traceability strategy, and they cannot validate whether your traces are correct and complete. However, ideally, requirements management tools allow you to:

- Establish and maintain traceability easily across your set of defined requirements in an unobtrusive way;
- Enforce the defined traceability strategy to a certain extent;
- Control the propagation of a change through your requirements hierarchy by identifying requirements that may be affected by that change based on the established traceability;
- Visualize and report on your traceability;

Such tooling capabilities, if well automated, can reduce the huge manual effort that would other wise be involved to manageable proportions and increase the reliability of your traceability. However, in 2003/2004, the traceability processes in Ericsson were not well automated.

A number of problems were identified by the KPA teams in the 2003 assessment of traceability practices on the 3G projects, but the most serious of these was that the product development cycle did not have an integrated tool suite:

- The Product Management Teams used Telelogic's Doors or Focal Point;
- The Requirements Engineers on many projects used Rational's Requisite Pro;
- Designers and testers both needed access to requirements, but because these were not integrated with each other, there were not proper linkages between the different tools into design and test.
- Integration Planning and Change Management used their own separate tools, different from those used for Requirements Management.

In addition, most of the tools within each tool-suite had a separate database, with the result that the same fact was often replicated many times. That not only led to high IT costs (hardware purchase, installation and maintenance) but made it a huge problem to ensure that consistency existed at all levels. Although this was well recognised as a problem area, the 3G projects still had 3 different relational data bases to work with. (CN-OSS, RANOS, GSM-OSS)

The problem of non-standard tools was compounded further by the fact that major problems occurred when trying to use the suite of tools across different platforms (Unix and NT). In Ericsson, many organisations develop on both NT and Unix platforms. A common problem found is that requirements may be captured in, for example, RequisitePro (Rational) on an XP platform, whilst the design is captured in Rose (Rational) on a Unix platform. In this case requirements cannot be traced through the design because of platform boundary problems.

The KPA's recommendations were:

- There should be one notation used across the traceability environment
- The traceability environment should provide complete software development life cycle coverage. For example from requirements, to modelling components to test components to build and configuration items.
- Information should be able to flow in a consistent manner between the various disciplines in the product development lifecycle. So for example, the Requirements Management tool and the Test tool should be linked, because each requirement should have at least one associated test case.
- The traceability environment should provide support for issue-tracking, to allow defect reports and change requests to be integrated into the traceability system.
- There should be facilities within the tool-suite for project management, administration activities and report generation.

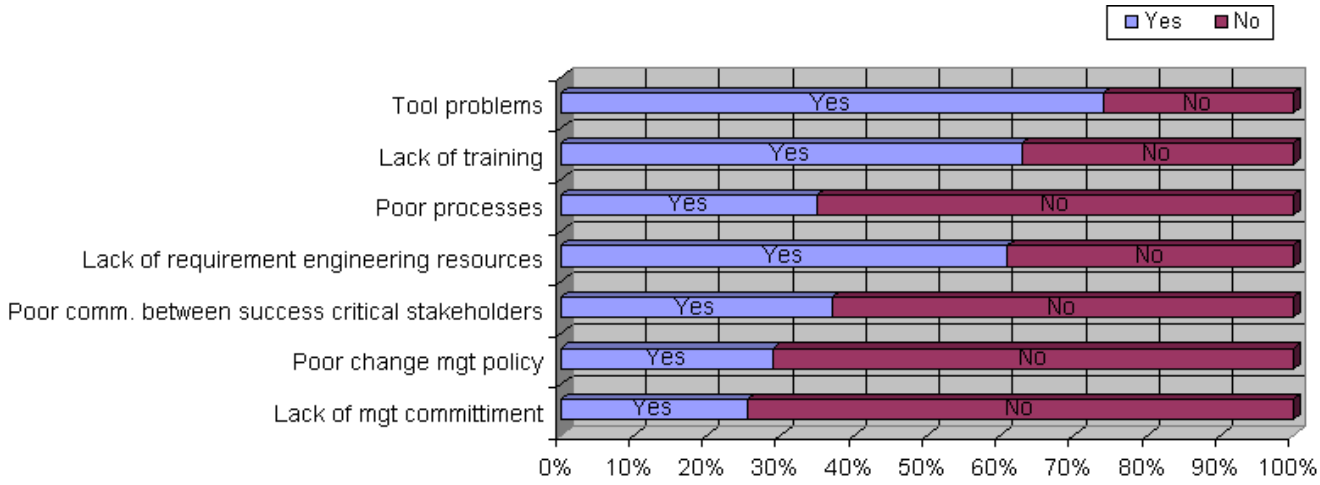
## 4.8 SUMMARY OF TRACEABILITY CONDITIONS 2004

In this section we review the main results from our analysis of 2004. We describe some of the problems that we encountered and report the main findings.

### 4.8.1 Results from Interviews

Using the classifications and the question matrix that we introduced in Chapter 3 (see Section 3.4) we set about defining a list of questions to further our understanding of the challenges that the software engineers face. Furthermore, using the results from Key Project Assessment Reports (KPA) and our assessment of the artifacts that we had reviewed (Requirement Management Plans, Training Plans and end project analysis reports) we determined that the main problems may be process, resource, tool, communication and policy related. Moreover, these questions were similar to the investigations that we were carrying out in the industrial survey; therefore we could compare the results from the case study with the survey. Therefore to determine the root cause of problems that the different software engineering practitioners faced we asked each respondent to answer yes or no to the following list of questions:

1. Have you a problem with the tooling solution?
2. Do you feel you receive adequate training to traceability?
3. Do you feel that the process describes traceability sufficiently?
4. Do you feel that there are adequate resources for requirement management tasks?
5. Do you feel that communication between the success critical resources is a problem?
6. Do you feel that the change management policy is sufficient?
7. Do you feel that a lack of management commitment is a problem?
8. Do you feel terminology is a problem when working with traceability?




| | Lack of mgt committiment | Poor change mgt policy | Poor comm. between success critical | Lack of requirement engineering | Poor processes | Lack of training | Tool problems |
|---|---|---|---|---|---|---|---|
| No | 67 | 71 | 63 | 39 | 65 | 37 | 26 |
| Yes | 23 | 29 | 37 | 61 | 35 | 63 | 74 |

**Figure 4-16 Results from Interviews**

The results shown in Figure 4-16, *Results from Interviews*, we illustrate the results from the questions. During the interviews we continued to ask the respondents what were the main problems they faced with the tool, Requisite Pro. The respondent's answers included the following:

- Lack of training. On further investigation we discovered that the Ericsson's training department were not allowed to deliver training unless their resources were certified by Rational. On inquiry with the training department, they responded that there was not a sufficient number of Requisite Pro courses on demand to merit getting a trainer certified.

- Most of the requirements were documented in Application or Main Requirement Specification documents and transferring the requirements to the tool was a major task, in addition the resources shortage lead to a lack of resources.

- Technical problems accessing the requirements database. While Requisite Web was a function of Requisite Pro, where the requirement matrixes could be accessed across the web, many software engineers responded with dissatisfaction on the performance of this web based feature.

- Requisite Pro did not perform well with requirements that were used by multiple projects.

Here is a summary of the biggest problems and that were identified by those interviewed in 2004:

*Problem 1: Complex Product Structures*

In 2003 the OSS domain was subdivided into three separate product structures, CN-OSS, RANOS and GSM-OSS. Each product had fragmented organisational structures with autonomous product strategies. GSM-OSS used different standards, architectures and platforms based on first generation technologies. While CN-OSS extracted OSS data from the core network and RANOS extrapolated data from the radio network.

*Consequence 1: Duplicated Traceability Items*

In some cases traceability items were common to all three products or an item in one product was dependent on an item in another product. Interoperability of these items was very difficult. Duplication of items is common in this scenario with the danger of duplication of development efforts.

*Consequence 2: No Common Terminology (Or Common Semantic Model)*

With three separate development organisations there was no common glossary or agreed body of knowledge causing disparity in semantics or terminology used between software engineering roles. For example, one organisation used the term work product, while another used artefact or document. One organisation refer to requirements while another call them traceability items.

*Consequence 3: Poor Communication*

Good traceability requires communication between the success critical stakeholders. The lack of a common product strategy and semantic model lead to poor communication between the success critical stakeholders.

*Consequence 4: Complex Organisation Structures*

Due to decentralized OSS product development with disparate customers, distributed market units, dispersed stakeholders, autonomous platforms, and disassociated architectures, unconnected code bases leading to complex organisational structures. This caused poor communication between critical roles leading to independent development of dependent modules. Often times, a lack of trust between organisation units lead to poor co-ordination in practices like traceability.

*Problem 2: Separate Product Development Processes (PDLC)*

The CN-OSS, RANOS, GSM-OSS product development units had separate product development lifecycle processes. Each process defines different traceability practices, different roles with inconsistent responsibilities, mismatched milestones, with incongruent change control processes, requirement management strategies, design, implementation, testing and compliance strategies. Even the process definition format varied from development process to development process. (Documents, web-pages or spreadsheets)

*Consequence 1: No Single Unifying Process Framework*

No single unified product process framework lead to poor co-ordination of practices, different document names containing the same traceability items, poor role definition leading to misunderstandings in responsibilities and so on. Overall a large number of those interviewed believed that this was one of the causes of the poor traceability practices between the product teams, project teams, business units and the outsourcing partners.

*Consequence 2: Poor Traceability Definition*

Traceability was not included as an integral part of the development lifecycle. While EUREP described it as a best practice the process did not give enough information for software engineers to complete their day-to-day tasks.

EUREP described why, when and how to trace at a very high level. The process should also define the type of links to be used, how to evaluate between good links and bad ones, how to identify traceability items, what their attributes are and how to keep items under configuration control. While much of this information was recorded in the Requirement Management Plan we observed there was disparity between the EUREP process and the RM plan. Furthermore, system descriptions were written by system engineers using a different lingo to the marketing team or the tester. A clearly defined end-to-end process would satisfy the lack of a common terminology problem.

*Problem 3: Separate Traceability Software Management Tools*

CN-OSS stored their traceability items using Telelogic DOORS while GSM-OSS and RANOS used Rational RequisitePro. This caused problems when traceability items had to be duplicated across product development units. Tracing items between products was difficult as was maintaining and extending traceability structures.

*Consequence 1: Discombobulated Tool Experience*

Traceability items that are common to multiple products must be duplicated which can lead to many problems. For example, duplicated traceability items are difficult to keep under configuration control. A change to one item may not be reflected in the related item in another product. Traceability items that are located in one tool or database are easier to maintain and extend. A trace link in one tool is different to a trace link in another tool. Furthermore, different attributes, tags, identifiers, tags were used in the different tool approaches.

*Problem 4: Difficult Training & Knowledge Transfer*

While traceability is a critical practice, most software engineers did not receive sufficient training. Many of those interviewed found it difficult to find core knowledge area in any on-line or training support manual. Because the projects had entered into third party supplier contracts with Ericsson, this meant that the organisations had to either certify trainers or use certified trainers. In 2003/2004, the training situation was very poor, with very few if any courses delivered on process or requirement engineering in this timeframe. The economic downturn and Ericsson's continued poor financial performance were the main contributing factors for the lack of training.

*Consequence 1: Poor Reusability of Good Practices*

Ericsson did not have any approach for capturing or reusing successful best practices. Also, without a common body of traceability knowledge or central repository for capturing best practices there was poor reusability of successful practices.

*Consequence 2: Loss of knowledge*

Without a mechanism for capturing or communicating traceability skills or knowledge, information is lost. Loss of knowledge of good traceability practices was a common complaint by those interviewed. This problem was further illustrated when the Requirement Manager responsible for OSS-RC left Ericsson in 2006 bringing with him much of the traceability expertise and knowledge. His primary legacy was the OSS-RC RM plans, but no best practice documents or diary of experiences was left behind by him.

In conclusion, while Ramesh contends that a corporate strategy for traceability is vital for it to succeed. While Ericsson clearly had a basic strategy in place for traceability at a corporate level, and each development unit received a directive on methods and tools to be used, however, it was limited in scope, and at too high a level to assist with the selection of the tools, the processes or with any of the day to day problems faced by the software engineers.

In the event the guidance and directives given on traceability by Ericsson's corporate HQ to the 3G projects, proved to be insufficient, for reasons that were partly due to the scale of the endeavour, which involved so many teams and sites at different locations.

The problems encountered in this study, whilst no doubt being partly magnified by the scale of the endeavour, are indicative of the sort of problems that other companies are likely to face.

The KPA recommended "one notation to be used across the whole environment"! What does this mean? Is it picking up on lack of common understanding of terms? Or is it getting at the fact that there are multiple notations in use amongst the different parts of the process and roles?

The KPA teams report highlighted the fact that traceability can be hindered to the point of complete dislocation through a multiplicity of tools that cannot co-exist on the same host machine or interface with each other. In addition the existence of multiple data bases at the very least undermines the reliability of the trace procedures. In this case too many tools are worse than too few.

From this, one could conclude that traceability from end-to-end cannot work, unless each project team has a common tool suite that can pass information from one tool to another seamlessly. That is not to say that each project team must have exactly the same tool suite, but there must be a certain element of universality to allow separate but inter-related projects to share and co-ordinate requirements and data.

The Ericsson experience provides some interesting insight into organisational attitudes to traceability in large organisations and endeavours. As one would expect from the earlier survey work by this author, there is a very positive attitude to traceability amongst requirements managers and project managers at Ericsson, despite the great difficulties encountered on the 3G projects. What is a fresh insight is that designers and testers cannot

see the benefits of such because they are cocooned from what is going up front, to a large extent, by the inability of their own tools to interface properly with tools like ReqPro. Without new and better tools, attitudes are unlikely to change.

As regards training, one could criticise Ericsson's decision not to give any organised training on the use of the various automated tools, either in-house or contracted out, as cavalier and unsystematic, but given the poor economic context of this study, these problems could probably not have been overcome.

The attitudes to training can be summarised as:

- Good attitudes to traceability at management level, less of understanding of importance at design, test and maintenance levels.
- Training is too expensive and not supported by in-house training organisations.

So the question remains how did these initial findings influence the survey, our proposed solution framework and the rest of this paper? Firstly, we were still analysing the results from the case study while we designed the industrial survey. The initial results gave us the understanding that the challenges that Ericsson's faced were; complex processes, complex organisational structures, poor communication, the lack of a common terminology, the lack of a unified tooling strategy, insufficient training, poor reusability of good practices and varied attitudes to traceability. As shown in Figure 4-18, *Impact of Findings on the Survey*, these initial findings helped us with the design of the industrial survey.



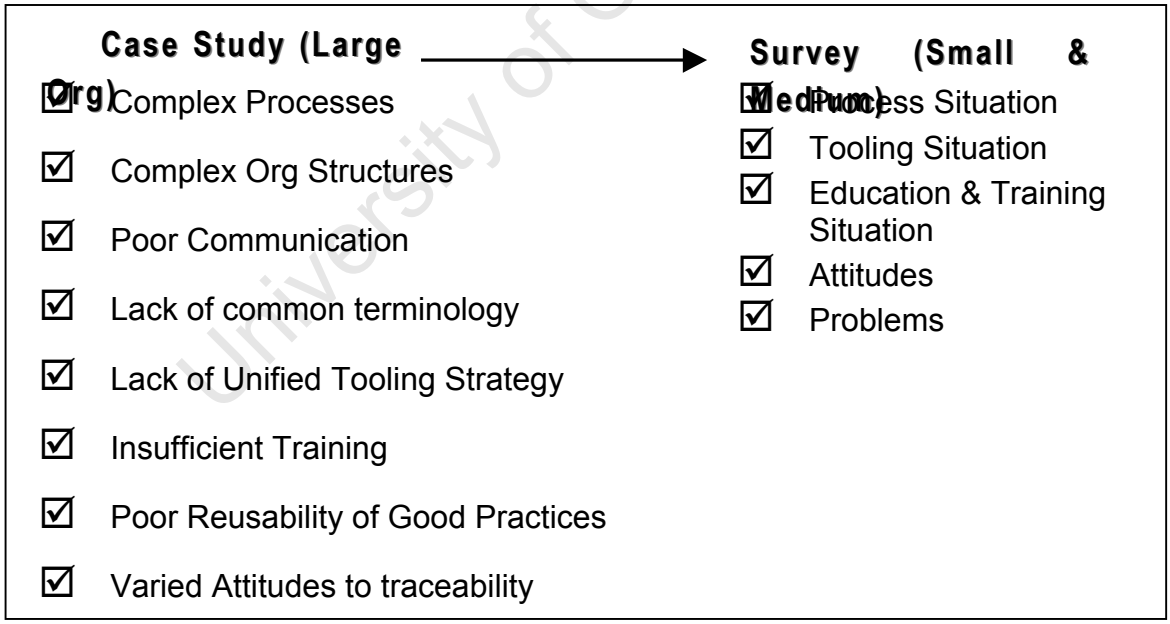



**Figure 4-17: Impact of Findings on the Survey**

The other question that remains is how did these initial findings influence the solution framework? By analysing the initial results from the case study we started a very important phase in any solution generation and that is understanding the problems. As shown in Figure 4-19, *Problems mapped to solution*, we illustrate a conceptual mapping of problem to solution, which is sufficient at this stage of the project. In Chapters 7, 8 and 9, we begin

with an analysis of each problem discovered in the case study as the starting point of our design of the components that we propose.

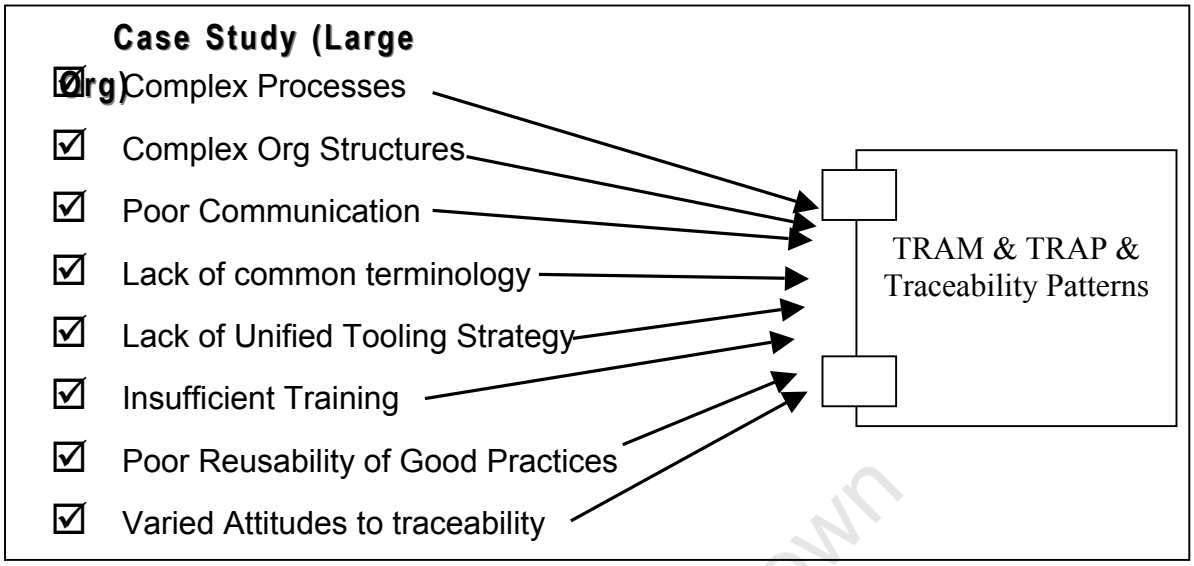



Figure 4-18: Problems Mapped to Solutions

In conclusion we gathered a lot of very useful data on traceability in 2004. The main problem in carrying out a case study of this nature was the scale or amount of information that it encompassed. However, there were a number of factors that made this study easier in this particular case. We came into this study understanding the technological context, with a good understanding of the history of requirement traceability in Ericsson; we understood the cultural context, the terminology being used by the different roles under investigation, the organisational structures and the management styles which made the execution of this study easier. A researcher who did not have this background coming into this type of study would have found it very difficult to come to grips with the factors described in this chapter. However, this has its own added risk that the researcher may be too familiar with the environment and therefore not have the distance to see all the underlying problems. To overcome this problem, we presented our findings to a number of software engineering researchers, who we included in the data analysis process. This gave us independent insights and ensured that we overcame the problem of researcher "gone native".

While this chapter sets a good context of the traceability situation the picture is incomplete without describing the changes to the traceability practices as the products evolved and the practices change. Therefore in Chapter 10, we describe the major changes that took place between 2003 and 2007 in an attempt to complete this picture.

# Chapter 5

# STATE OF THE ART INDUSTRIAL SURVEY

## 5.1 INTRODUCTION

The pressure on small to medium sized organisations to develop better quality products, with quicker turn around time, for smaller profit margins is great. This is partly due to the saturation of small software development companies from as far a field as China, India and the Middle East, and partly due to new "hostile take-over" strategies of larger companies who prefer to purchase the competition rather than compete for their business. Companies who survive have to develop products quicker, better and faster for modern customers who have higher expectations. In many cases, smaller organisations simply cannot afford to invest in new processes, new tools, training programs and extra resources if they are to compete efficiently especially within the broader trend of globalization and offshore outsourcing of software development as one way of cutting development costs. We believe that matters arising from globalization are raising the need for further empirical studies on traceability back on the agenda. (Standish_Group, 2006)

This chapter further investigates the initial findings from the case study except in small and medium sized organisations. We need to understand if the same problems are prevalent in smaller organisations. Examining in-depth the development processes, practices, tools and attitudes towards traceability in small and medium sized company, it is concluded that traceability, rather than improving is in fact not being practiced in any formal approach in many software development organisations. Moreover, we investigate the training and education situation and the impact this has on these improvements.

In 2004, we commenced a case study with Ericsson, providing data on the implementation of traceability in a large organisation. However, this single source of data only gave us one perspective from one type of industry and did not describe the state of the art in small to medium sized organisations. To overcome this problem we designed this survey, with objectives of gaining a better understanding of the status of traceability and investigating the current issues faced by the smaller corporations. In this chapter we describe our approach, the assumptions made, the data we gathered, the results correlated, the conclusions made and the implications this study had on the design of our proposed solution framework.

## 5.2 BACKGROUND TO SURVEY

### 5.2.1 Previous Surveys on Traceability

Several surveys on traceability have been completed in the past two decades. In 1994, at the 1st Conference on Requirement Traceability, Gotel and Finkelstein presented an empirical traceability study of 100 samples of data. (Gotel and Finkelstein, 1994b) They identified two problems with traceability; the lack of a *common definition of traceability* and the *conflicts in the understanding by the practitioners* of the traceability problem. Using the empirical data two new concepts emerged to the traceability domain namely *pre-*

*requirement specification* and *post- requirement specification*. Many of the problems identified were attributed to inadequate pre-requirement specifications.

In 1998, Ramesh investigating traceability across 26 software development organisations divided traceability users into two separate groups: (Ramesh, 1998)

1. Low-end users, who only practice traceability because it is mandated by project sponsors.
2. High-end users who view traceability as a major task to achieve better controlled process and improve quality systems engineering process. High-end users, employ richer traceability schemes, enabling a precise register of the reasoning of the traces followed.

Ramesh states that the "[T]he differences among the two groups of users are highlighted by the spectrum of views they have on the various factors influencing traceability practice". Ramesh provided us with an empirical framework for carrying out a survey on traceability. Furthermore, his classification of high-end and low-end users provided us with a reference point for classifying our results. In particular he describes low-end users as follows:

- Low-end users often adopt ad-hoc practices and methodologies.
- Low-end users apply simple traceability techniques.
- Low-end users see traceability practice as a cost overhead.
- Low-end users find that sponsors do not appreciate the benefits of traceability.
- Low-end users' managers view the control of traceability activity as an overhead.

Ramesh's study gave us a proven approach for investigating traceability in an industrial context. Not only does he define classifications of traceability users, he also gives us categories for investigation, namely, tools, processes and attitudes. While Ramesh utilised good empirical approaches for carrying traceability surveys he did not investigate educational or training factors or the impacts of budgetary constraints as major factors that influence traceability. In our opinion, the results he provides do not quantify the true state of the art, rather he provides us with focus areas for investigation.

On the other hand, Martin Gills in 2005 presented a survey from 32 projects from 6 IT companies. (Gills, 2005) The goals of the survey were towards possible improvements of traceability practices in IT projects. He addressed the attitudes of software developers regarding traceability, showing the trends of the culture of traceability and indicating typical traceability models in the IT corporations. In his results he states the following:

"*One of questions addressed to project members was whether the project specifically paid attention to the traceability. Positive answer was given in 53.1%, negative - in 21.9%, but medium attention was in 25.0% of cases. In most cases the observed traceability model corresponds to the latest version – 78.1%. Equal number of projects have taken the methodology from a successful earlier example and have made the model from the scratch (46.9%). Most projects will pass their experience to other new projects (71.9%). Most of projects do not maintain their traceability information by means of a special tool. Tools are used only in 8.8 % of cases, partially – in 37.5% of projects. Author did not observe a co-relation between quantitative characteristics (size, duration) and the traceability practice. One can remark that the small projects with a team of 1-4 people or those with duration over 1 year were more interested in traceability than others"*

Gills, also defines categories for investigation namely; the attitudes of small organisations towards traceability, the number of organisations that use tools, the reuse of traceability practices and the impact of the size of the project on the attitude of the

practitioners towards the practice. What is particularly interesting about Gills results is that he clearly quantifies his findings in percentages that show clearly the state of the art. Unfortunately, Gills results were not publicly available when we commenced our survey; however, it is interesting that he uses many of the same categories for investigation.

During the analysis of the available literature, we became aware of repeated words or phrases highlighting important aspects of traceability or areas the may require empirical investigation. Each word was noted and described in a short phrase. For example, attitudes to traceability, impact of processes, categories of different types of traceability users and state of the art of tools. By comparing each phrase in turn with all other phrases, we found further commonalities which formed broader categories of investigation. Glaser & Strauss described this method of continually comparing words and concepts with each other as their "constant comparative method". (Glaser, 1967) Strauss & Corbin recommended coding by "microanalysis which consists of analysing data word-by-word" and "coding the meaning found in words or groups of words". By applying the constant comparison technique we created broad categories to be investigated. (Strauss, 1998)

Using this constant comparison technique it emerged that no traceability survey investigated the education background of the software engineers in the organisations or the training they received as a factor that influences traceability practices. While, this survey was complete by the time the Center of Excellence for Traceability released their technical report on the Grand Challenges in Traceability in 2006, they did however reiterate a need for our line of inquiry by stating that: "[T]raceability is a key success factor for any software or systems project, but very few people are proficient at tracing and there are few educational programs available to impart such proficiency" They continue to state that "[T]raceability training programs must teach skills needed by traceability practitioners, however there are no standard traceability skill sets defined".



### 5.2.2 Identified Case Study Problems

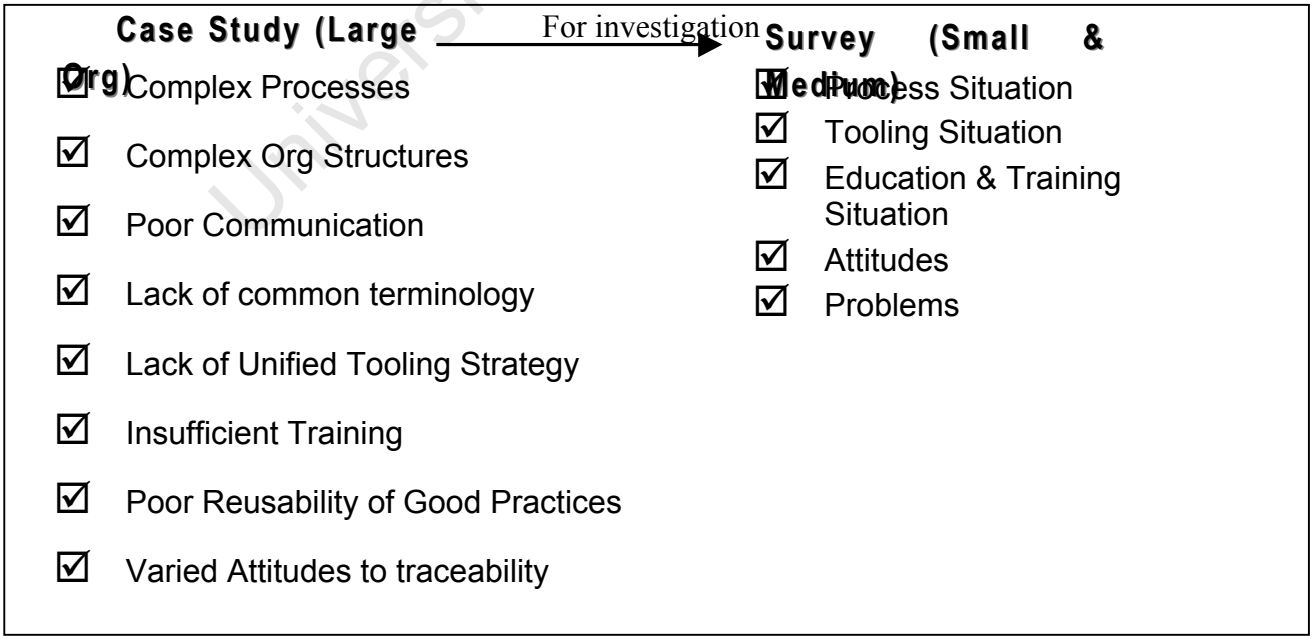



**Figure 5-1 Case Study Problems and Focus Areas for Survey.**

The problems identified during the case study are summarised in Figure 5-1, *Case Study Problems and Focus Areas for Survey.* In the figure above we also illustrate the categories for investigation, namely; process, tools, education and training, attitudes to traceability and the problems faced by small organisations implementing traceability. The relevant problems identified during the case study include the following:

- *Problem 1 Lack of a Single Unified Process:* At the beginning of the case study in Ericsson, OSS development was divided into three separate product-lines, CN-OSS, RANOS and GSM-OSS with separate product development units, different process development teams and different processes in place.
- *Problem 2 Poor communication and collaboration between the success critical roles brought about by* the dispersed geographical nature of the development.
- *Problem 3 Lack of a common terminology or glossary of terms.* A phrase or term in one development unit had a totally different meaning in another development unit.
- *Problem 4 Lack of Unified Tooling Strategy:* There was no corporate tooling strategy mandating recommended tools. Rather each product development unit could evaluate and decide which tool to use. For example, the CN-OSS used Telelogic's Doors while RANOS and GSM-OSS used Rational's Requisite Pro leading to serious problems on interoperability between tools and traceability items duplicated across different products.
- *Problem 5 Poor Training & Knowledge Transfer:* Due to the cost of using third party suppliers and the difficult economic climate many essential training programs were put on hold or did not take place. For example, very little training on process and traceability tool took place between 2001 and 2003 causing a lack of competence and poor knowledge transfer between the software engineers.

The main implications of the case study on the survey were as follows:

- We used the same categories of investigation namely process, tools, training and attitudes to traceability in this study.
- The problems that emerged during the early phases of the case study became the focus areas for the survey.

Furthermore, any new factors that arose during the survey were investigated in later stages of the case study to gain a better understanding of the problems and to see if they applied in larger traceability applications.

### 5.2.3 Assumptions made on Case Study which impacted the Survey

Due to constraints of time and resources in executing this survey a number of key assumptions had to be made as follows:

- Less complex product structures: Due to the small size of the development teams we assume that smaller organisations will not have as complex a product structure as identified in the OSS-RC product family.
- Less complex organisation structures. Due to the size of small to medium sized organisations we assumed that the smaller organisations would not have as complex organisational structures. In early 2003, Ericsson's OSS development was subdivided into 16 different design houses. We therefore make the assumption that smaller organisations do not have distributed development on any scale similar to Ericsson's.

While our case study was executed over four years it gave us a more time to assess all factors that influence traceability and how these factors changed over a period of time. This scale of investigation was not possible in a survey carried out over a couple of months. The survey is therefore not a sequential story of traceability, as is the case with our Ericsson's case study; rather it provides a snap-shot of the state of traceability in the context of a number of diverse industrial organisations.

### 5.2.4 The Purpose of Survey

Before launching into the survey, the research methods that we used and the results obtained, let us first describe the purpose of the survey: (see Figure 5-2 below, *Purpose of Survey*)

1. Gather data on the state of the art of traceability processes and tools.

2. Gain an understanding of attitudes all resources involved in software product development on the importance of traceability.

3. Describe the educational background of the different software engineers and the impact of training on the practice of traceability.

4. Identify a list of problems faced by smaller organisations in implementing traceability.

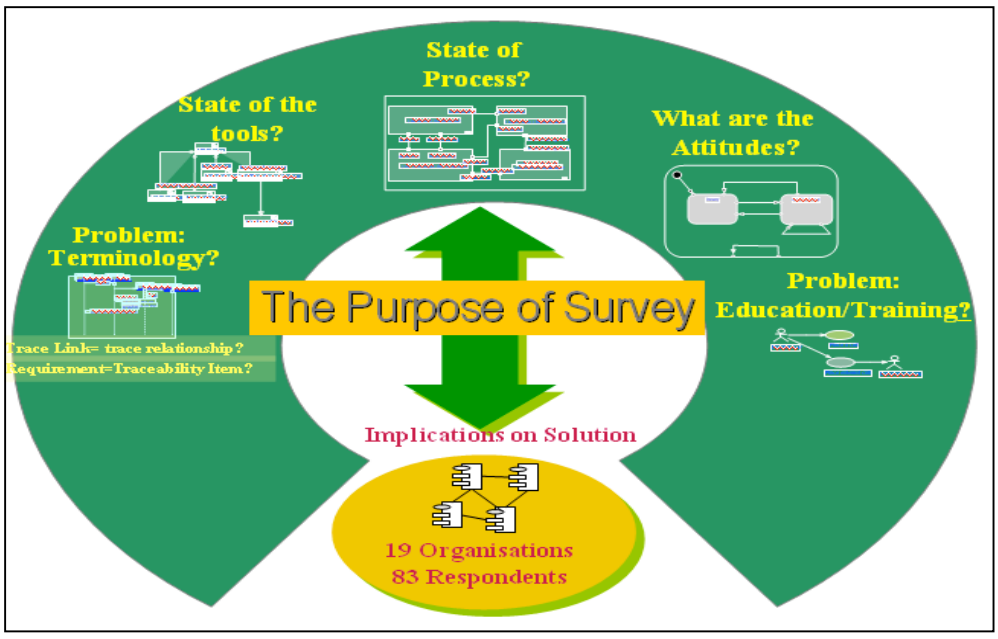



**Figure 5-2: Purpose of Survey**

## 5.3 RESEARCH METHOD

The main objectives for the survey method was to assist us to systematically collect, analyse and interpret data on traceability from a variety of industrial contexts. We illustrate our research method in Figure 5-3 below, *Research Method*. We adapt Yin's approach in the design of our research method:

- Define the Research Method (Chapter 3)

- Establish the purpose of the survey. (section 5.1.4 above)
- Determine the research questions. (see Section 5.4)
- Determine our sample, data gathering and analysis techniques
- Prepare, collect and analyze the questionnaire data.
- Define the interview questions.
- Prepare, collect and evaluate the data from the interviews.
- Analyse the data.
- Present the results and conclusions.

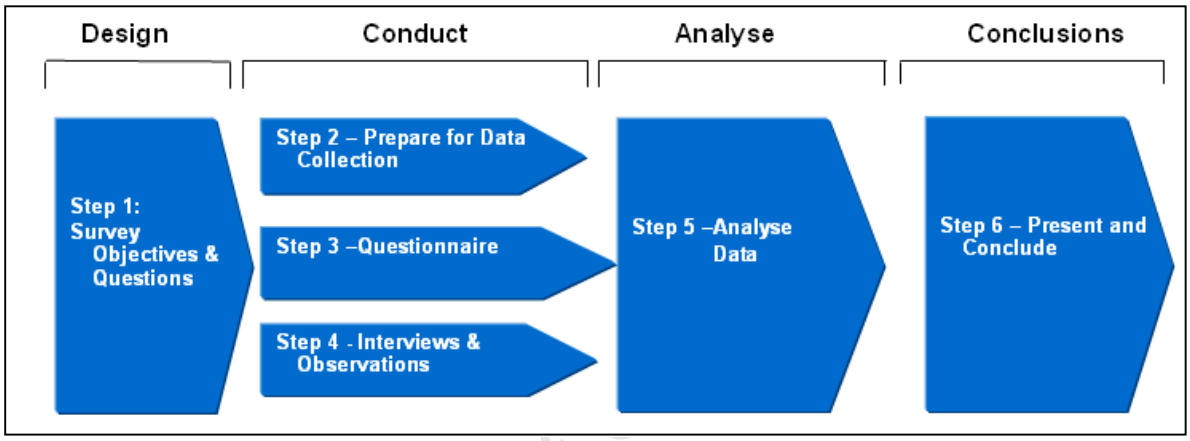



**Figure 5-3: Research Method**

### 5.3.1 Designing the Questionnaire

We designed each questionnaire against the following criteria:

- Are the questions unambiguous and easy for respondents to understand?
- Do the questions address our initial research questions?
- Do the questions assist us gain an insight for the development of a solution framework?
- How will we evaluate the results of each question and how do we output the results.

We designed the questions into two question types: Yes/ No and checkbox data gathering. (see Appendix II, for the complete questionnaire)

### 5.3.2 Collect Data in the Field. (Questionnaire)

In South Africa we focused on a software engineering forum called SPIN (Software Process Improvement Network's) which was affiliated to the SEI. This forum met once a month with approximately 20-50 members attending each meeting from a variety of industrial settings. The researcher distributed the questionnaires at the start of the SPIN meeting and briefly introducing the questionnaire and indicated that it would take twenty minutes to complete. A brief description of the doctoral research at the University of Cape

Town followed. However, we did not want to influence the responses so we did not describe in any way any of the traceability concepts or how the results would influence the rest of the research project.

The researcher was well known at the SPIN meetings which was advantageous in the return rate of the questionnaires. In all 38 questionnaires were distributed with 30 samples returned. Due to the wide spread of software engineers attending the meetings a sufficient breakdown of different roles completed the questionnaires.

In Ireland with less complex socio-demographic and technological restrictions the questionnaires were simply distributed by electronic mail to most respondents with a cover letter. The cover letter gave the name and address of the researcher, background to the research being undertaken, the aims of the study, how and why the respondents were selected, a description of the use of the data and a description of the confidentiality clause. In some cases a follow up call was given a day after the first distribution of the questionnaires.

In Ireland, 45 questionnaires were distributed with the researcher personally knowing approximately 85% of those contacted. A maximum of three reminders were sent to respondents and the data was gathered over three months. A total of 27 questionnaires were returned. Approximately 15 reminder phone calls were made, with the majority of those called returning the questionnaires. In Ireland a number of the respondents requested that the researcher sign a non-disclosure agreement of information related to specifics within their corporations when reporting the findings. A number of observations were made during the data collection in Ireland:

- Fewer responses from the large (250+) corporations.

- In general more respondents than in South Africa asked for further information on the confidentiality of the information in particular specific data related to the corporations.

- Management in micro and small corporations were more willing to get involved in surveys than in medium and large corporations. An analysis of this phenomenon was best described by one project manager who stated: “perhaps we will learn something or at least have a list of questions that we should be asking ourselves”

- Approximately 30% of those returning the questionnaires requested further information on traceability, practices, processes and tools after they completed the surveys. For example, a list of tool suppliers, simple approaches to traceability or specifically if there were “open-source” traceability tools available. In a number of cases, a generic requirement engineering and traceability presentation was given by the researcher to build up relationships for future interview sessions.

### 5.3.3 Analysis of the Questionnaire Data

We defined a scoring method for each Yes/No response. For example we designed the questionnaire so that the “yes” answers signified good practices while “no” implied a poor practice. We gave each respondent 3 points for each “yes” answer. This scoring system assisted us establish a measurement baseline that could be used for comparisons data especially data between different countries.

### 5.3.4 Collect Data (Interviews)

We wanted the interviews to be an interesting and a stress-free experience. At the start of each interview, we suggested a period of forty minutes for each session. It was important to get the respondents interested in the survey from the beginning. Therefore we started each interview with a brief introduction to the context of the research project describing the interviews purpose, the layout of the interview session and how the results would be used. We also described the incentives for completing the interview, such as the opportunity for each business to have a copy of their results and the benefit to the researcher.

We took written notes during the interview and recorded field notes after the interview was completed. While the interviews were open-ended, we structured each session around the research questions defined at the start of the survey. From the questionnaires we knew that some of the resources would be more interested in the topic than the others. For example, while the project managers were generally interested in any practice that might improve product quality the testers viewed traceability sceptically. In the cases where the topic was already of interest or importance to the respondent, we started with general questions, and then we moved to more specific questions. In the case where we evaluated the topic was of low importance to respondents, we started with specific questions about traceability in their discipline. This gave the respondents a frame of reference; before we asked broader, more general questions.

We tried to suppress their opinions and conclusions. This was the most difficult aspect of the interview sessions. It became evident that as the interviews progressed certain interviewees used the sessions to outline their grievances with their current position or management practices. This was particularly clear during the broader questions in relation to the overall business. We also observed that the resources in the later stages of the product development lifecycle, for example system testers felt that management focused more on the earlier stages of the product development than on maintenance or support issues. A number of testers believed this was due to management's lack of understanding of the testing discipline.

We interviewed 26 practitioners. In total we documented 15 interviews in South Africa and 11 in Ireland. Approximately 18 hours of interview data was documented.



### 5.3.5 Background to Sample

South Africa is often referred to as "the rainbow nation" with over 46-million people and a wide variety of cultures, 11 official languages and diverse religious beliefs. Its economy is based primarily on mining, agriculture, manufacturing and commerce. Primary exports include mined minerals and gold. For much of the 20th century, South Africa was governed under apartheid rule. However in April 1994, a major political transformation occurred when the African National Congress was elected to power. In the last decade of apartheid, the economy grew at 1.1% per annum. In the ten years 1994-2003, the growth rate averaged 2.9%; 2.7% per annum over the last five years.

On the other hand, Ireland has a booming IT sector with very few socio-economic, socio technical or infrastructure problems. The IT Sector employs 90,700 people in over 1,300 companies up from 19,000 in 1990 Over 300 overseas companies in the ICT sector have a presence in Ireland directly employing approximately 61,000. (ICT, 2006) Ireland is the largest exporter of software in the world. (ICT, 2006)

In 2002, thirty-two IDA-backed ICT companies undertook to invest €120 million in R&D Seven of the worlds top 10 leading ICT companies having a substantial base in Ireland Turnover in the ICT sector was over €51 billion in 2001 with three of Ireland's top exporters accounting for 18% of total exports between them Value added in the ICT sector accounted for 11.6% of Ireland's GDP in 2000, compared with an EU average of 5.1% (Forfás, 2005)

In compiling this survey the goal was not to judge one country over another. Rather it provides us with real data that explains the factors that influence traceability in small to medium sized organisations.

### 5.3.6 Who Was Surveyed

Some 19 companies took part in the survey. In South Africa, 12 organisations took part. South African participants consisted of all persons who attended the SPIN (Software Process Improvement Network) between the dates of June and Aug 2005. In Ireland, 7 organisations took part, the participants being mainly from the researcher's professional network. The distribution of the types of organisations involved from the two countries combined is illustrated in Figure 5-4, *Survey Sample*, below.

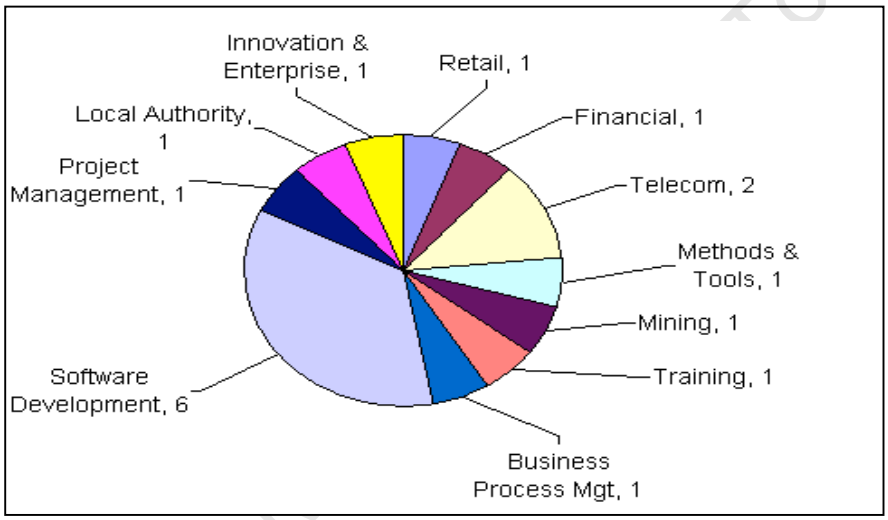



**Figure 5-4: Survey Sample**

To ensure consistency, participants had to fit one or some of the following criteria:

1. A history of working in the development of a software product or a software engineering company

2. A history of working in small or medium sized software companies.

3. Have knowledge of software development.

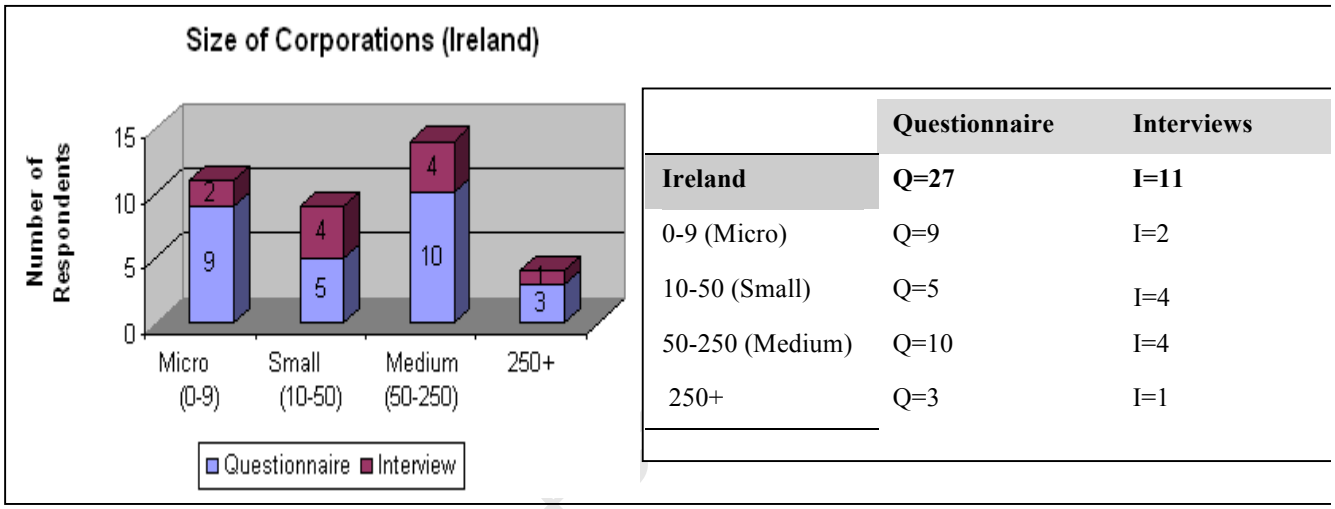
The majority of our respondents were software engineers involved with the development of software. Some respondents were consultants with a broad span of experiences from various industrial sectors.

The data was collected in first in Ireland, between Dec 2004 and June 2005, and then in South Africa, during the summer of 2005. This resulted in a sample size of 83

persons from 19 different organisations. The distribution of the interviews and questionnaires was as follows:

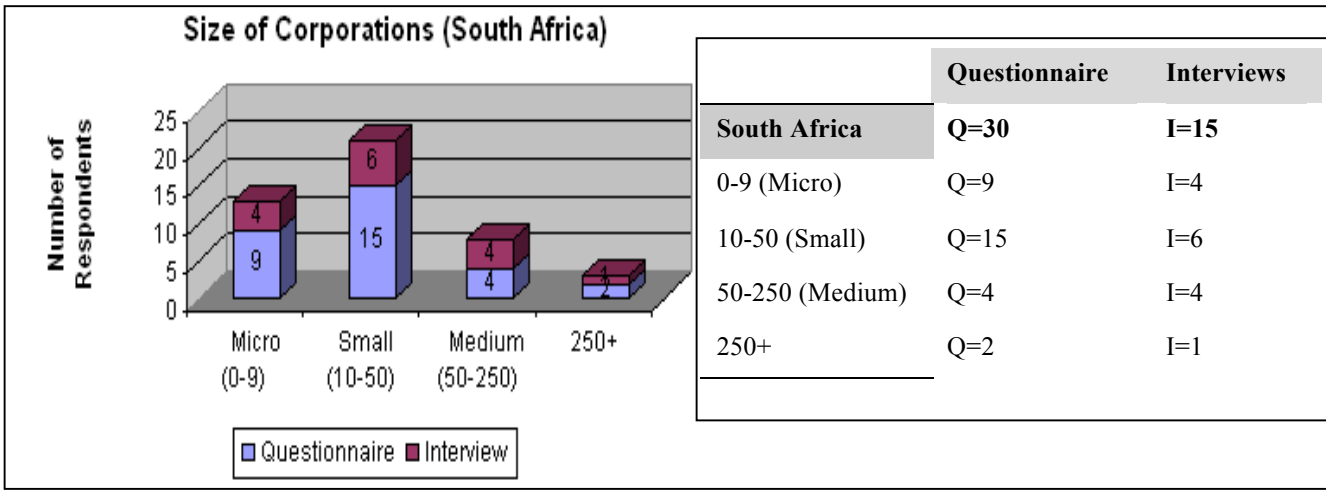
| Questionnaire | Interviews | Total No. Software Engineers | |
|---|---|---|---|
| Ireland | 27 | 11 | 38 |
| South Africa | 30 | 15 | 45 |

We classified the size of organisations into categories based on number of IT employees as: micro = 0-9; small = 10-49; medium = 50-250; large = 250 or more employees.




| | Questionnaire | Interviews |
|---|---|---|
| **Ireland** | **Q=27** | **I=11** |
| 0-9 (Micro) | Q=9 | I=2 |
| 10-50 (Small) | Q=5 | I=4 |
| 50-250 (Medium) | Q=10 | I=4 |
| 250+ | Q=3 | I=1 |

**Figure 5-5: Size of Organisations (Ireland)**




| | Questionnaire | Interviews |
|---|---|---|
| **South Africa** | **Q=30** | **I=15** |
| 0-9 (Micro) | Q=9 | I=4 |
| 10-50 (Small) | Q=15 | I=6 |
| 50-250 (Medium) | Q=4 | I=4 |
| 250+ | Q=2 | I=1 |

**Figure 5-6: Size of Organisations (South Africa)**

In Figure 5-5, *Size of Organisations (Ireland),* above we illustrate the percentage of the respondents spread across the size of the corporation were micro = 29%; small = 24%; medium = 36%; large = 11%.

In Figure 5-6, *Size of Organisations (South Africa)*, we illustrate the percentage of respondents recorded across the size of organisation were micro = 29%; small = 46%; medium = 18%; large = 13%.

The reader will note that the overall distributions for size are similar for South Africa and Ireland, but that the ratio of small/medium sized companies is greater for South Africa. The different roles that took part in the survey are shown in Table 5-1, *Dispersion of Sample across Roles*, below.

| ***Software Engineering Roles*** | **Total** |
|---|---|
| **Technical Sales** | 3 |
| **Product Managers** | 5 |
| **Systems Architects** | 8 |
| **Project Managers** | 17 |
| **Requirement Management** | 2 |
| **Configuration Managers** | 2 |
| **Methods & Tools** | 5 |
| **Senior Developers** | 10 |
| **Developers** | 16 |
| **Testers** | 16 |
| **System Testers** | 3 |

**Table 5-1 Dispersion of Sample across Roles**



## 5.4 PRESENTING THE RESULTS

In Chapter 1, we introduced patterns as a mechanism for describing recurring problems in a standardised approach ensuring the communication of the findings in a formalised way. Christopher Alexander's instructive definition of patterns that, "[e]ach pattern describes a problem that occurs over and over in our environment…." By creating a template or pre-defined form we provide a novel approach for describing the results from our survey. Patterns are a formalised abstraction that many users can understand with structured text providing an effective mechanism to capture and communicate knowledge. While the traceability pattern template has been adapted for describing the results of the survey we will see in later chapters that patterns also provide a way for describing patterns that emerge in the TRAM and the TRAP. In Table 5-2, *Pattern Template used for Presenting Results*, below we describe each heading.

| | |
|---|---|
| **Problem** | A statement of the problem which describes the goals and objectives it wants to reach within the given context and forces. |
| **Context** | The *conditions* under which the problem seem to recur. It can be thought of as the background to the problem. |
| **Questionnaire Findings** | The statistical findings from the Questionnaire. |
| **Insights from Interviews** | Further information that was captured during the interview sessions. This was especially useful to gain further insights on ambiguous questions. |
| **Implication on Solution** | A brief discussion on the implications on the solution framework that we propose. |

**Table 5-2: Pattern Template Used for Presenting Results**

## 5.5 KEY QUESTIONS & FINDINGS

The key questions asked, and the order for the presentation of the findings is shown in Figure 5-7, *Presentation of Findings* below as follows:

1. Have you practiced traceability?
2. What is the tooling situation?
3. What is the process situation?
4. What is the Requirement Engineering situation?
5. What is the education situation?
6. What is the training situation?
7. What are the attitudes to traceability?

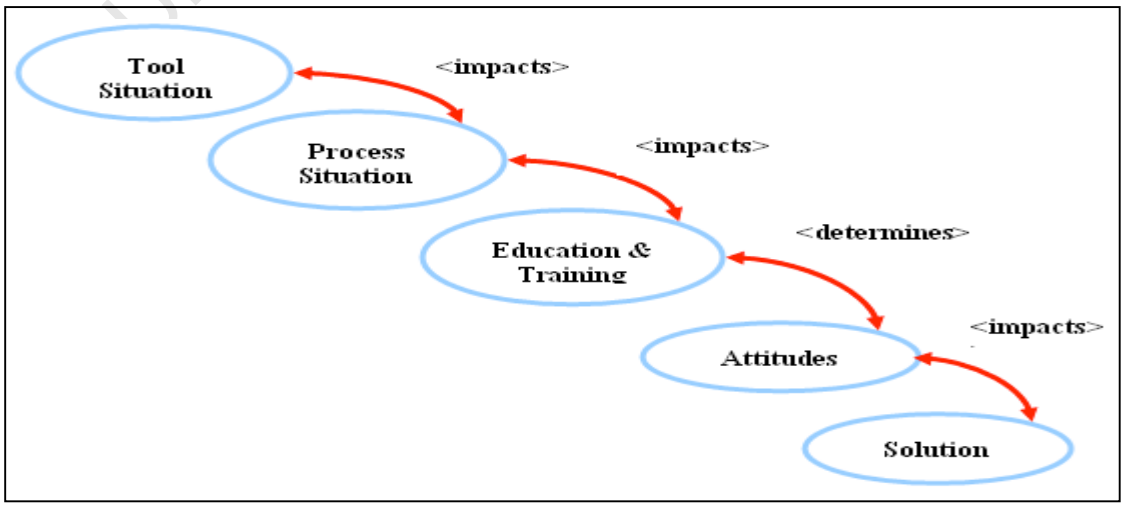



**Figure 5-7: Presentation of findings**

### 5.5.1 Practice Traceability?

Have you applied traceability in any form in your current role in any project(s)? The results are shown below in Table 5-3, *Traceability Practices.*

| *Software Engineering Roles* | YES | NO | Total Respondents | % of Total Who Practiced |
|---|---|---|---|---|
| **Technical Sales** | 0 | 3 | 3 | 0% |
| **Product Managers** | 3 | 2 | 5 | 60% |
| **Systems Architects** | 3 | 3 | 8 | 50% |
| **Project Managers** | 10 | 17 | 17 | 58% |
| **Requirement Management** | 2 | 0 | 2 | 100% |
| **Configuration Managers** | 1 | 1 | 2 | 50% |
| **Methods & Tools** | 2 | 3 | 5 | 40% |
| **Senior Developers** | 2 | 8 | 10 | 20% |
| **Developers** | 3 | 13 | 13 | 23% |
| **Testers** | 2 | 12 | 14 | 16% |
| **System Testers** | 1 | 2 | 3 | 33% |

**Table 5-3: Traceability Practices**

The overall percentage of practice of traceability dispersed across all the roles is 46%. This dispersion is also shown below in Figure 5-8, *Traceability Practices*.

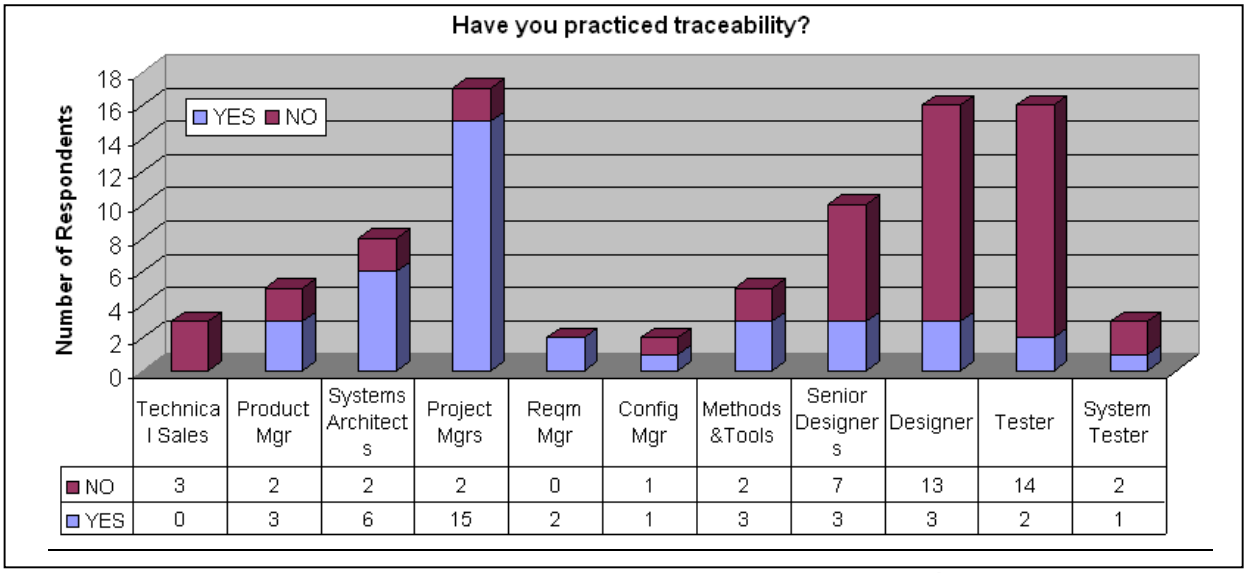



**Figure 5-8: Traceability Practices**

### 5.5.2 Tooling Situation

**Problem:**

Statistics on the usage of traceability tools is one of the most urgent facts for the traceability community. Due to the complexities of traceability, a tool helps to store the required product components into the traceability items, organize the relationship between different these items and give essential statistics on the progress, rate of change or quality of the projects. Thus, there exists a considerable need to focus on how the small and medium sized organisations manage their traceability situation and the technology that they use

**Context:**

In this set of questions we investigate the usage of traceability tools or other more informal approach across the industrial sample. Whilst we expected that many organisations would have some approach for managing the requirements, we understood that the existence of a tool (formal or informal) was a good sign that organisations are managing their development in a controlled fashion.

**Questionnaire Findings:** (see Figure 5-9, *Tooling Situation*)

- A low percentage of organisations (18%) have a dedicated requirement management and traceability tool.

A much higher percentage (43%) use some form of informal tooling approach for managing requirements and traceability items. By informal we mean Spreadsheets, or Word matrixes

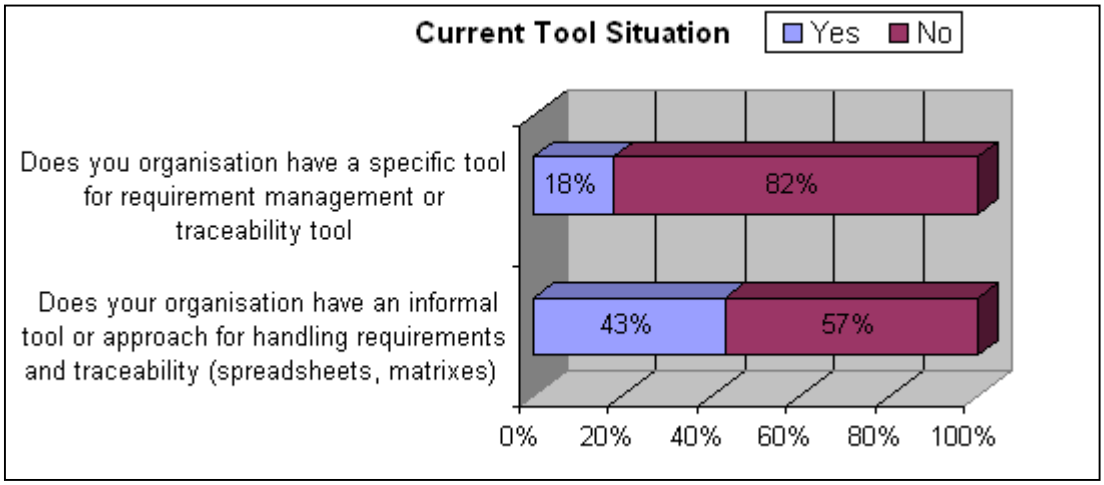



**Figure 5-9: Tooling Situation**

**Insights from Interviews**

During the interview sessions, it was evident that the single most important factor that influenced small to medium sized companies from investing in a requirement management and traceability tool was the budgetary constraints that they faced. Other factors that emerged from the respondents that influenced the purchase of tools included the lack of management commitment, a greater demand for higher priority design tools, the lack of resources to administer such a tool and the low quantity of requirements with less of a need for a dedicated tool. The majority of organisations described that they used some form of requirement specification document or spreadsheet for creating a requirement list. What was most interesting is that 77% of those interviewed believed that the requirement list was not kept up to date, with the daily changes that occurred in the projects. Many of the respondents enquired if the traceability tools had the functionality to manage changes to requirements, with a large percentage stating that change control was a much bigger problem for their projects than tracing between requirements.

**Consequences:**

Without a formal tool, there is an increased risk that traceability activities are not carried out. When asked during the interview sessions what were the main consequences that engineers believed were caused by the lack of a tool, the project managers discussed product concurrence as a difficulty and the lack of statistical data on the progress of the projects. A number of testers described that their activities are compromised because they don't have full visibility to the customer requirements. One maintenance engineer described difficulties understanding how the correction patches traced to the different versions of the products. A number of organisations especially the micro and small, described that while they had one main product they had a lot of different versions in use by different customers. Maintenance in such situations was difficult and a number of testers believed that product versioning should be included as a traceability item in any

| |
|---|
| tooling solution. |
| **Implications for Solution**<br><br>While it is clear to us that a traceability tool should be an integral part of any development process, there is currently very little data on the cost benefit of introducing traceability tool into smaller organisation. While this study does not develop a traceability tool, the fact that so few organisations used a tool these findings motivated us to carry out a tool survey (Chapter 2) and also to investigate if TRAM and TRAP could be an initial step for cost benefit analysis on the introduction of tools. |
| **Comparison with Case Study**<br><br>The lack of budget that small organisations face was not a factor in Ericsson's. In Ericsson traceability tools were always accepted as an essential tool in the development environment. The problems faced in Ericsson, however, was a lack of a mandate on which tool vendor to use leading to problems with interoperateability between tools. |

### 5.5.3 Process Situation

| |
|---|
| **Problem:**<br><br>Although many organisations have some form of a development process, often times the process is described in an ad hoc or informal approach. Here we ask two additional questions to establish whether they have a more formal process: Do the organisations have a process team? Do they model their processes? |
| **Context:**<br><br>In this set of questions we investigate the usage of a development process across the industrial settings. Whilst we expected that many organisations would have a development process, we evaluated that the existence of a process team was a sure sign of process maturity within that development organisation. The supplementary question concerning a conceptual process model is aimed at ascertaining the percentage of organisations that model their processes, because we were investigating if a Traceability Process (TRAP) model was a suitable solution in this research project.<br><br>While the answer's to these question are useful in giving us an indication of the presence processes with control, they are a precursor to more serious ones about more detailed aspects of requirement engineering and traceability, which follow. |

**Questionnaire Findings:** (see Figure 5-10, *Generic Process Questions*)

- As expected, a high percentage of organisations (87%) have a development process.
- Of the 13% of the respondents, whose organizations lacked a formal process, one was an innovation organisation only involved in promoting innovation in organisations.
- Only 42% of organisations had a process team, because many of the micro and small companies do not have the resources.
- Of the 42% who had process teams of some sort, only one in four of those, or 9% of all, modeled their process using process modeling techniques or tools, which is a truly shocking finding.

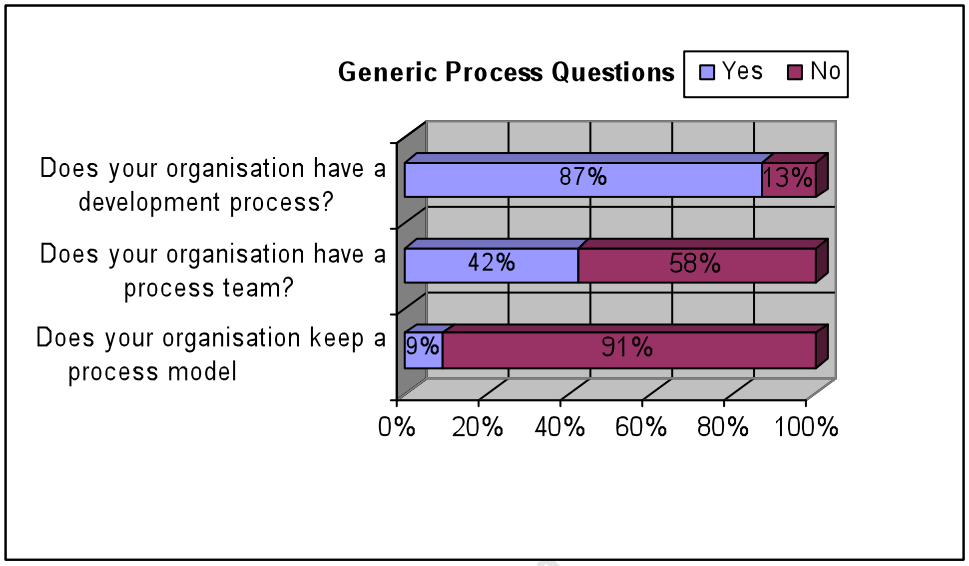



**Figure 5-10: Generic Process Questions**

## Insights from Interviews

During the interviews we further investigated in what format the processes were defined. For many smaller organisations the development process was an informal document outlining activities. One company, Cora Systems, had a novel approach for integrating process definition and traceability into their project plan. (see Figure 5-11, *Cora Systems Traceability Approach*) They described their process using a project plan, with requirements attached to the different activities outlined by the plan. Traceability was applied by the links between the different activities in the project plan.

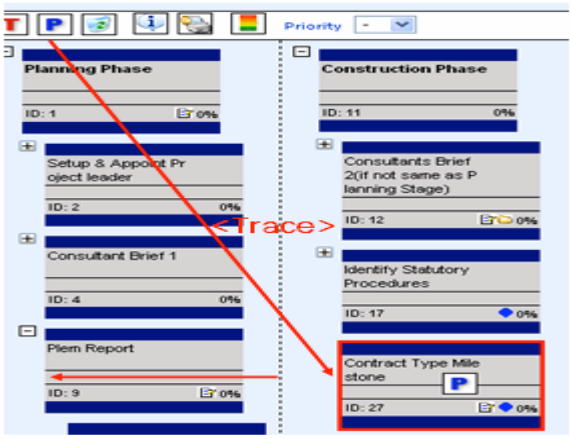



**Figure 5-11: Cora Systems Traceability Approach**

The interview sessions also revealed that in the micro, small and medium organisations the process team generally consisted of part-time project members will little time dedicated for the process definition. Furthermore, many of these teams met at irregular intervals usually between projects.

Lack of continuity in process definition was acknowledged as a problem by many respondents in these small part-time teams as a critical issue. The most common problem identified was that while management agreed that process was important they did not allocate sufficient time for any serious progress on process improvement and refinement.

No organisations had used process modelling standards like SPEM.

**Consequences:**

The consequences of not having a development process are so numerous and far reaching that we will only describe a few of the key effects on requirement engineering and traceability:

- The lack of clear definitions of the team's roles and their corresponding traceability responsibilities, leading to a lack of understanding of certain roles on the importance of traceability or the implementation of traceability in their every day tasks.
- Lack of agreement on carrying out product concurrence, lack of integration between the different disciplines of analysis, design and test and the lack of reusability of traceability practices.

The lack of even the most basic formalised process models means that even the most basic traceability principles are not applied.

**Implications for Solution**

It is clear the software processes are an integral part in the development of quality software products. Without question most organisations understand the importance of processes however it was evident from the data, in particular the interview data that there is serious lack of consensus in opinion on how this should take place. One common theme did however emerge: New Standards for process development are required. In later sections we propose the TRAP with a primary objective of assisting all organisations in the implementation of a traceability process in their organisation.

**Comparison with Case Study**

The poor definition of traceability in the development process, motivated further investigations of the OSS-RC traceability process with special regard for approaches to improve the current situation. The problem within Ericsson was not the lack of a process or even the lack of a definition of traceability rather that there was no unification of processes, with different sub-products or individual sites developing their own processes. In comparison to the poor findings in smaller organisations, Ericsson's process definition and control proved to be of a much higher quality.

### 5.5.4 Evidence of Requirement Engineering Processes

**Problem:**

Traceability is a principle of requirement engineering. Without a formal requirements engineering process, traceability in general will not occur. It is a necessary objective for our survey and research to understand the fundamental state of the art of the requirement engineering processes in place. In this section we asked 2 questions:

- Does your process describe requirements engineering sufficiently?
- Does the process describe traceability?

**Context:**

Requirement Engineering involves formal investigative and analytic processes by which it is possible to discover, document and maintain a user's requirements effectively. Requirement management is an essential foundation for both project management and software development activities, at the very least to ensure that they run smoothly.

However, the primary purpose of requirements engineering is to ensure that the quality of the software and systems products meets the customer's requirements. The primary metric that measures the success of a software system is, whether or not it meets all the requirements of its users, that is the satisfaction level of its users reflects its degree of success. For a requirements engineering process to be fully effective it must describe traceability from an end-to-end perspective. A requirement engineering process is a set of structured activities to derive, validate and maintain a systems requirements document, for common reference.

In addition, as the endeavour moves into a detailed design and engineering phase, there is a true engineering activity that develops the overall capabilities and system requirements into more detailed and lower level requirements for individual software modules, equipments and packages thereof. The engineering task then becomes one of creating cost-effective solutions to real-life requirements. Requirements engineering thus allows the appropriateness and cost effectiveness of the solution to be analysed in a systematic way, and traced back to the customers requirements, above all for the benefit of the customer.



**Questionnaire Findings:** (see Figure 5-12, *Requirements and Traceability*, below)

- Only 52% of respondents believe that requirement engineering is sufficiently defined in their organisation.
- Only 18% thought their process did describe traceability in some way.

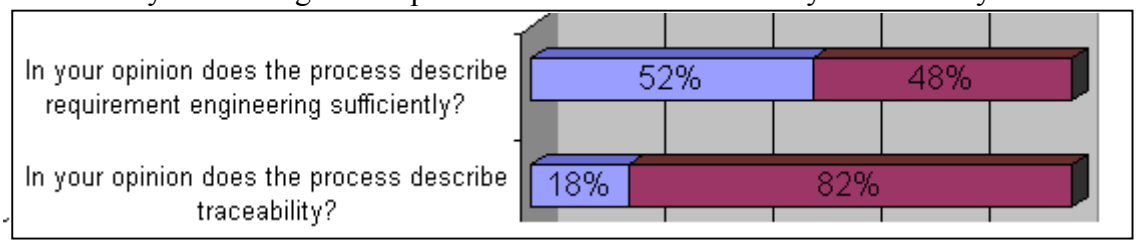



**Figure 5-12: Requirement and Traceability**

| |
|---|
| **Insights on Causes from Interviews**<br><br>During the interviews, it became clear that in some cases that, those that thought they had a requirements process sufficiently well-defined, were in fact using the most rudimentary of techniques. In fact one of the common trends that emerged during the interviews was the high number of cases (57%) who stated that they had requirement engineering defined in a spreadsheet or word document.<br><br>Furthermore, during the interviews, we discussed the use of requirement management plans. The number of organisations with a requirement management plan was very low at 19% with a large number (23%) not understanding what a requirement management plan even was.<br><br>Also during the interviews, in many organisations that state they practice sound requirement engineering principles, it became clear that the role of the requirement manager was in fact carried out by the project manager, systems engineer, product manager or senior developer. |
| **Consequence:**<br><br>The simple truth that poor requirement engineering processes will usually lead to poor traceability practices. |
| **Implications for Solution**<br><br>Our primary concern is not to develop new and improved requirements engineering methods and techniques, rather, it is to enhance the development of cost effective processes that will assist in the deployment of better requirement engineering and traceability practices. |
| **Comparison with Case study**<br><br>Ericsson has mature requirement engineering and traceability practices. The major differences were in the definition of the processes. Ericsson utilised the EUREP (Ericssons Unified Requirement Engineering Process) and process modelling using IBM's RMC process modelling tool. |



### 5.5.5 Education

In this line of inquiry we were interested to make correlations between the education qualifications of the respondents and in a later section investigate the link between education and attitudes to traceability. Type of education: 1 of 83 (7%) did not have a third level education in some form. For 49 (59%) their highest degree was a bachelors' degree; for 22 (27%) it was a masters degree, and 1 had (7%) reported having a Ph.D.

Table 5-4, *Education Dispersion*, shows the field of study of the participants. Where a participant reported more than one field of study, the table reflects only that field closest to computer science or software engineering.

| Discipline | Ireland | South Africa |
|---|---|---|
| Computer Science | 22% | 34% |
| Information Systems | 24% | 25% |
| Other Engineers (Electrical, Mechanical, Industrial, Civil) | 23% | 18% |
| Other Science (including mathematics) | 6% | 14% |

**Table 5-4: Education Dispersion**

It should be noted that we applied a random approach to data gathering therefore the respondent's education qualifications and the results are not reflective of the overall statistics of education dispersion in Ireland or South Africa. On analysis of the Irish statistics we were surprised with the low number of science graduates and the high level of engineering graduates that took part in the survey. This could be explained by the fact that during the explosion of IT in the mid to late nineties in Ireland the demand for IT graduates was more than the traditional disciplines of Computer Science and Information Systems could provide. In South Africa we were also surprised by the higher level of Computer Science and Information Systems graduates that took part in the survey. On analysis it could be explained by the close proximity of the University of Cape Town and Stellenbosch University with strong Computer Science and Information System programs.

Below we illustrate in Table 5-5, the overall results of those respondents who completed requirement engineering training in university. As expected the percentage of those who did requirement engineering in university in computer science and information systems was high. The number in the other disciplines is very low. The level of "don't know" was investigated further during the interview sessions. Many of "more senior" interviewees who had left university more than ten years beforehand understandably did not remember whether they had studied requirement engineering or traceability in university.



| Discipline | Yes | No | Don't remember |
|---|---|---|---|
| Computer Science | 79% | 12% | 9% |
| Information Systems | 74% | 15% | 11% |
| Other Engineers (Electrical, Mechanical, Industrial, Civil) | 17% | 54% | 29% |
| Other Science | 6% | 75% | 29% |
| Other Disciplines (typically arts or business) | 7% | 70% | 23% |

**Table 5-5: Did you do requirement management in university?**

On further analysis of the interview records many of the interviewees who had received training in university in requirement engineering reflected that it was a module (in some cases one lecture) in a generic software engineering course. It should be noted that

many of those interviewed also did their formal education before the UML standard and use case requirement engineering was released.

**Problem:**

As a fundamental starting point we need to establish "how much education in requirement engineering and traceability", software engineers received in university. Although, in the last two decades, software engineering courses have become widely spread in IT curricula, nevertheless, such courses focus more on architecture, coding and testing than requirements engineering practices.

**Context:**

While many universities have exemplary education programs, in many cases the course modules focus more on the design stages of development, rather than the earlier requirements analysis and management processes. In many situations, academic staff lack industrial experience, so that they don't fully appreciate that good management of product development is a crucial aspect of good design. The Grand Challenges report supports this statement by stating that: "Traceability is a key success factor for any software or systems project, but very few people are proficient at tracing and there are few educational programs available to impart such proficiency"(Hayes et al., 2006)

In general, Information Systems disciplines have a more management and requirement engineering focus. By discovering the state of education of those currently practicing software engineering, we hope to gain a better understanding of some of the factors that may impact the poor attitudes to traceability.

**Questionnaire Findings:** (see Figure 5-13, *Dispersion of Requirement Education across disciplines below*)

In the diagram below, we show the results of the survey question: "Did you receive specialised training in requirement engineering in university?"

The explanation for this is to be found in the graphic below. Whilst a high percentage of those who did Computer Science and Information Systems, did have some education in requirement engineering, other disciplines, even electrical engineering, had a very low exposure to the subject.

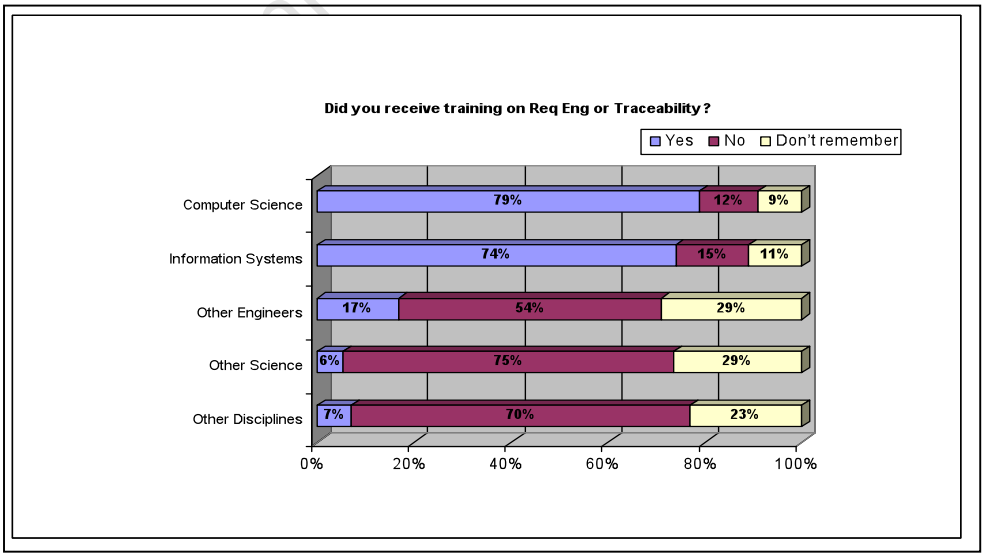



**Figure 5-13: Dispersion of Requirement Education across disciplines**

Although 79% of computer scientists did receive some form of requirement engineering education, the majority of those interviewed did not feel that this was sufficient with regard to the importance of the practice. We were surprised by the low percentage of Information Systems graduates who had studied requirement engineering but on analysis of the data it became clear that many of our interviewees had attended Information Systems courses for shorter periods than the Computer Science students.

**Insights from Interviews on Causes**

During the interviews a number of respondents responded negatively to the overall degree of software engineering training that they received in university. A common complaint was that the academic staffs had never worked in any industrial setting and were too far removed from the real issues to prepare the students for their actual work in the field. Many felt that courses run by practitioners in the field would greatly benefit the overall knowledge of the students.

The level of "don't knows" was investigated further during the interview sessions. Many of those interviewed had finished their undergraduate degrees many years before the survey. Many "more senior" interviewees who had left university more than ten years beforehand understandably did not remember whether they had studied requirement engineering or traceability in university. This helps to make the point that many practitioners currently in the field finished their educations a long time ago; long before the need to teach requirements engineering was recognised.

On the whole, it can be said that our survey showed that although many university programs have software engineering courses there is still shortfalls with regard to the quality of the requirement engineering and traceability modules. A number of the respondents described the following issues to be addressed to improve the efficiency of requirement and traceability practices:

- Incorporate requirement engineering and traceability into all course programs from OO programming to networking.
- Have guest industrial speakers emphasise the importance of requirement and traceability practices.
- Introduce more case studies illustrating project successes or failures with requirement management.
- Provide tools for implementing traceability.
- Introduce software processes as an integral part of every course.

**Consequences:**

The consequences of a lack of education in requirement engineering disciplines are:

- Far too few managers and systems architects have any grounding in requirements engineering and thus lack basic understanding of the importance of traceability, which means that many organisations are likely to have little enthusiasm for its practice.
- Furthermore, the lack of education in requirements engineering amongst electrical and engineers from other disciplines means that there is likely to be relatively little enthusiasm for traceability in the system design and development teams as well.
- University graduates with strong academic backgrounds tend to work in design and test, and so have no appreciation at all of the importance of the earlier requirements analysis stages, and the need to carry the discipline of traceability through their own activities.

One could conclude that the absence of third level education increased the possibility of poor attitudes to traceability leading to the certain software engineering roles not practicing the discipline.

**Implications for Solution**

Requirements engineering education should be based on best-practices and techniques, but it also needs to be anchored in state of the art research and the reality of industry practice. Preparing students for practice involves giving them an accurate view of reality as well as giving them the tools and the critical mindset needed to perform and improve on requirements engineering practices in industry. By utilizing experiences close to students it is possible to put abstract theory and practices in a context, as well as accomplish a deeper learning.

When analysing the data we initially identified a number of aspects that needed to be addressed. Firstly how would we bridge the gap between industry and academia? Generally the practitioners don't have the time to assist in third level education and in many cases even if they do they lack the pedagogical skills or course delivery expertise to execute a successful course. Furthermore, even if they did have the time to deliver courses they didn't have the time to run the administration aspects of correcting exam scripts, setting tutorials and so on. A mix of both academic and practitioners is a logical solution however in general it is difficult to expect an academic to take over the administration activities. Cross pollination of students with industrial projects is a good solution but generally practitioners lack the time or don't see the immediate reward of this type of approach.

Patterns capture experience and expertise in a simple format that can be used in any context. After identifying this critical problem we realised that a catalogue or encyclopaedia of software engineering patterns or requirement patterns would in fact be a good method to convey the importance and practical applications of practices like traceability. At a simple level even if a student understands that there are patterns for the creation of traceability items, the different types of traceability relationships or the importance of tracing across the entire product lifecycle would be a great benefit to all students. Of courses, design patterns are widely accepted by the academic community as an integral education tool, so why not extend this paradigm for requirement engineering and traceability?

**Comparison to Case Study**

The majority of Ericsson's personnel came from electrical engineering or computer science backgrounds. What was interesting to discover was that in the local university, Athlone Institute of Technology, Ericsson's sponsored many of their employee's to undertake postgraduate courses. They also participated on research committees and were heavily involved in further education of their staff.

In Figure 5-14, *Requirement Training*, we illustrate the results of investigations into the training situation. On analysis of the results it showed that training in requirement engineering and traceability could be the cause of poor implementation of traceability in organisations. While 30% of project managers received some form of training or mentoring, no system testers received any training. Furthermore the number of designers and testers who received training was exceptionally low.

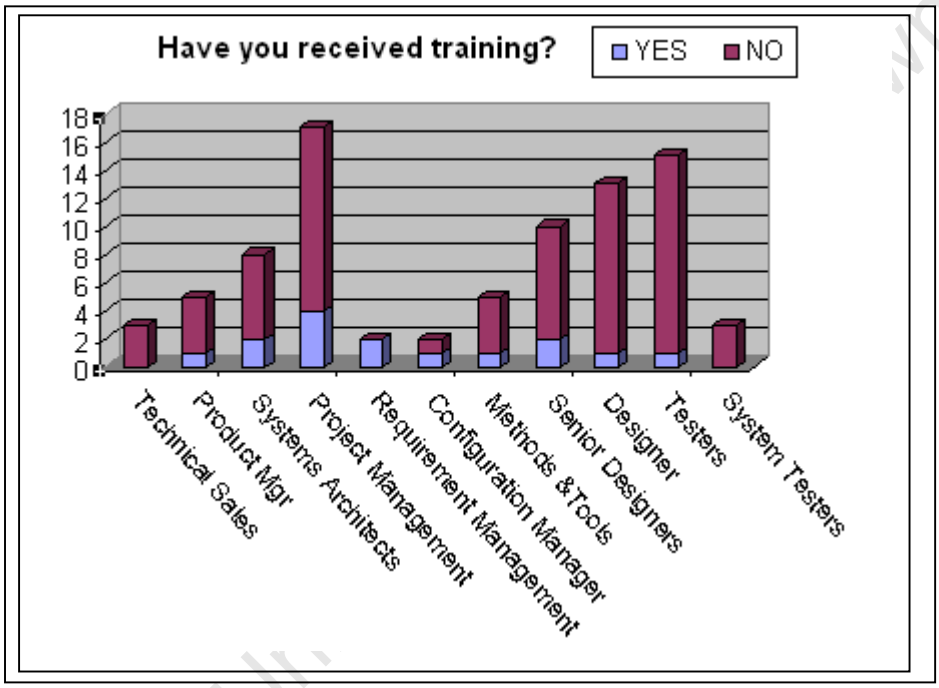



**Figure 5-14: Requirement Training?**

During the interview sessions, with careful focus on this aspect of traceability it became clear that the smaller organisations had little budget for training and almost all the budget was spent either on software development training, for example Microsoft .Net training or test management and automation. In the smaller organisation management training was usually on generic management skills training. Most project managers were self trained while a number of those interviewed had attended evening courses certified by the Project Management Institute (PMI).

Because a training program may seem expensive when compared to the amount that can be saved the cost is negligible. However, few organisations have this vision. Budgetary constraints are the single biggest factor why training in requirement management principles is not undertaken.

A small number of companies did train employees on the basic principles of requirement engineering or when a new requirement tool was installed but failed to train new employees when turnover occurred, believing that current employees can adequately train new hires. In general the interviewees described that this approach almost always failed. The people who are originally trained in the requirement tool may only remember a certain percent of its functions. However, these current employees only train new users based on what they remember about the program, and over time, many of the software's functions and features get lost and unused.

The consequences of poor of training in requirement engineering disciplines are:

- Lack of understanding of the importance of traceability causing poor implementation of the practice
- Processes lack requirement and traceability definition
- Misunderstandings of the core concepts leading to inconsistencies in terminology

In our solution, instead of using traditional training approaches we investigated another way of educating personnel on traceability. Once again, a cheap but effective approach to educating staff on the principles of traceability is to use traceability patterns. Case Study patterns can be used as a cheap but effective approach for training staff in traceability best practice. A traceability pattern is a description of how to solve a traceability problem which is sufficiently abstract that it can be understood by a wide variety of people and then can be reapplied in many different contexts. The pattern is recorded as text and often uses diagrams to get the point across. Another way of thinking of patterns is as explicit representations of experience in solving problems which recur. This explicit representation of experience is valuable in many circumstances. It allows for better communication. It also means that a best solution need not be re-discovered.



### 5.5.6 Attitudes to Traceability

**Problem:**

As with any new technology or management practice, there is usually a considerable amount of inertia, even resistance, to the need to change. Software engineers are no exception. A negative attitude to the need for traceability was clearly evident, from our own personal experiences working in requirement engineering and in the certification programs.

In the survey, in order to test attitudes, we asked participants “How would you rate the importance of traceability to software product development?”

**Context:**

The genesis of requirement engineering research began in the 90s, when researchers discussed the importance of traceability in the development of high quality software.

However by the early 2000s, due to the global economic slowdown and depressed IT budgets, software companies became increasingly reluctant to fund new developments in software best practices like requirement engineering. At the start of this research, we needed to obtain new evidence to see whether the need for traceability was more broadly accepted. We expected results would show that

requirement managers and project managers do promote traceability, but that other critical roles do not share the same opinion.

**Questionnaire Findings:**

As expected the overall attitude of the respondents varied greatly throughout the product development lifecycle. In the interviews it became clear that the requirement manager placed the greatest importance on the practice. The project manager focus on traceability was in relation to getting project statistics on the stability of requirements and progress of the overall project. 100% of requirement managers (2), 88% of project managers (15 0ut of 17), and 80% of systems architects (6 out of 8) agreed on the importance of traceability.

What is most worrying about the results are that only 30% of Senior Designers, 31% of Designers, 37% of Testers and 33% of System Testers, believe that traceability is important.

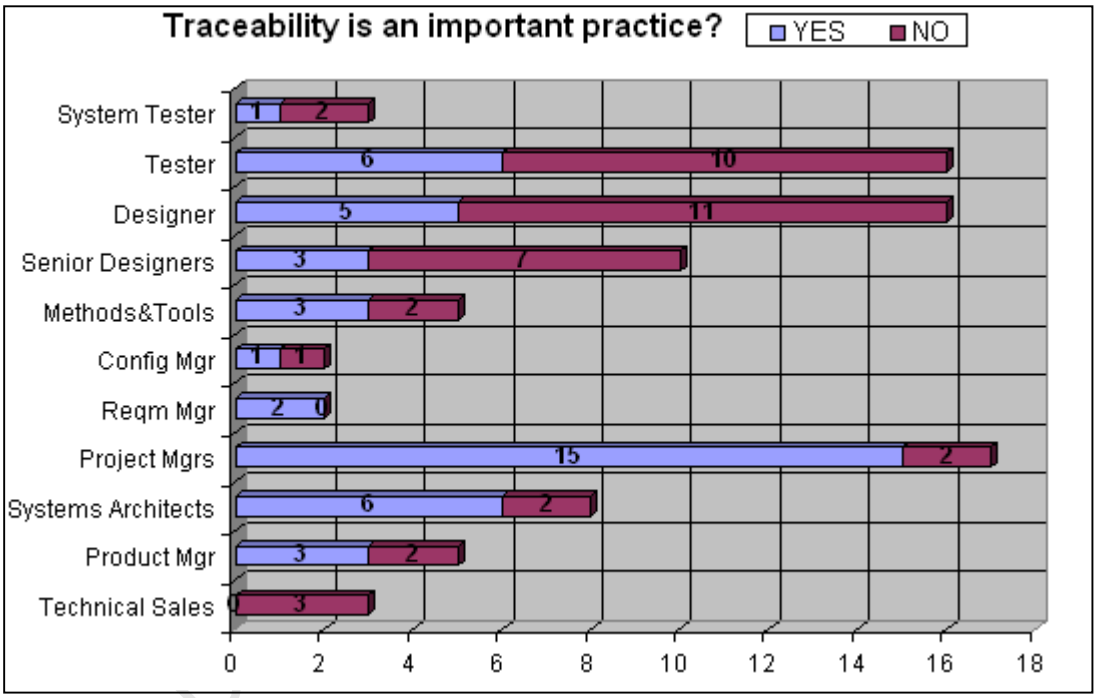



**Figure 5-15: Importance of Traceability**

**Insights From Interviews:**

The interviews provided us with a much greater understanding of the gap between the different attitudes that the software engineers have towards traceability. There is no question that the positive attitudes towards traceability are shared by the resources involved in the early phases of the development. While the most negative responses came from the testers and then the designers.

**Consequences:**

The consequence of poor attitudes are:

1. Traceability is not practiced by all roles therefore trace links do not cover the entire development cycle leading to a loss in trace data.

2. Project management is compromised as the project metrics do not include statistics from the disciplines that are not tracking their progress.
3. Requirement management can not measure the stability or progress of the requirements.

**Implications for Solutions:**

A solution that would appeal to the entire development organisation is needed. It was also important to ensure that solution was applicable to testing and design phases as it was to the early phases.

**Comparison to Case Study**

The statistics in Ericsson show a different story. In all 78% of those involved believed that traceability was an important practice in all aspects of the development lifecycle.

## 5.6 FURTHER INTERVIEW OBSERVATIONS

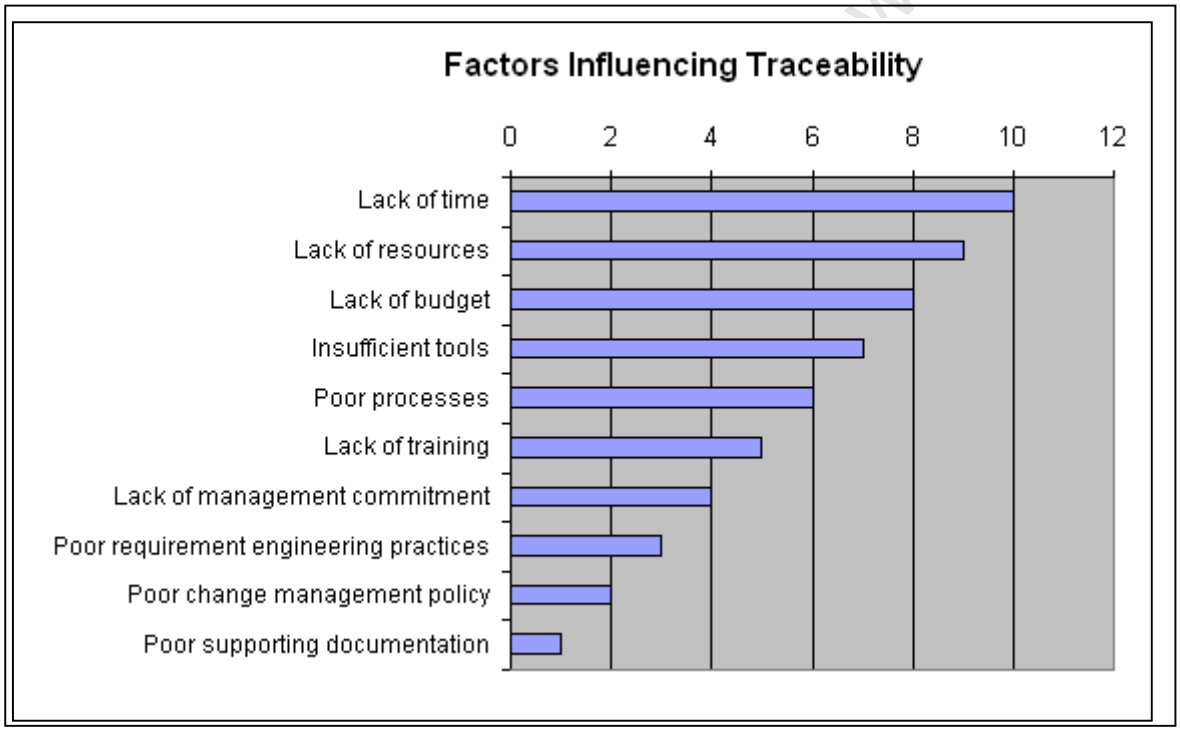



**Figure 5-16: Factors Influencing Traceability**

During the interview sessions we asked the respondents to describe the major factors that influence traceability in their organisations. In Figure 5-16, *Factors Influencing Traceability*, above, we illustrate the top ten factors as described during the interviews. Some of the results were surprising. We expected that the main factors would be tool and process related, however as one can see time, resources and restricted budget were the main factors described. Many of the interviewees admitted that they worked on a number of different projects at the same time. This was particularly true of the management groups. For example, one project manager managed five different projects at various stages, from early requirement gathering to support projects. Time and budget were the most recurring problems that organisations faced not only for implementing traceability but for many

development activities. The reality is that small to medium sized companies do not have the budget to invest in tools, processes and training. Therefore if traceability is to become a widely accepted core practice, cheap, cost effective and low labour intensive approaches to traceability are required.

During the interview sessions we asked the interviewees to grade the importance of traceability from 0 to 5. The results shown below in Figure 5-17, *Importance of Traceability*, in this case were not so surprising, with the requirement and project managers rating traceability as very important, while the software engineers in the later phases of the development rating traceability the lowest. In further investigations, it became clear that many of the designers and testers believed that the development and testing tools that they used were not sufficient to carry out there every day tasks. It was interesting, that many of the testers had the biggest grievances with the management team and they identified themselves as the most removed from the development strategy and development teams.

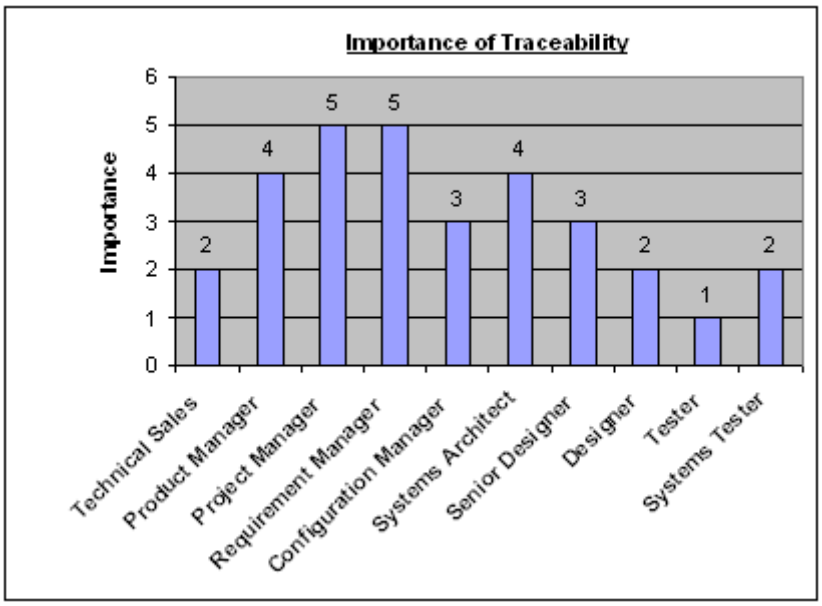



**Figure 5-17: Importance of Traceability**

## 5.7 IMPLICATIONS ON SOLUTION

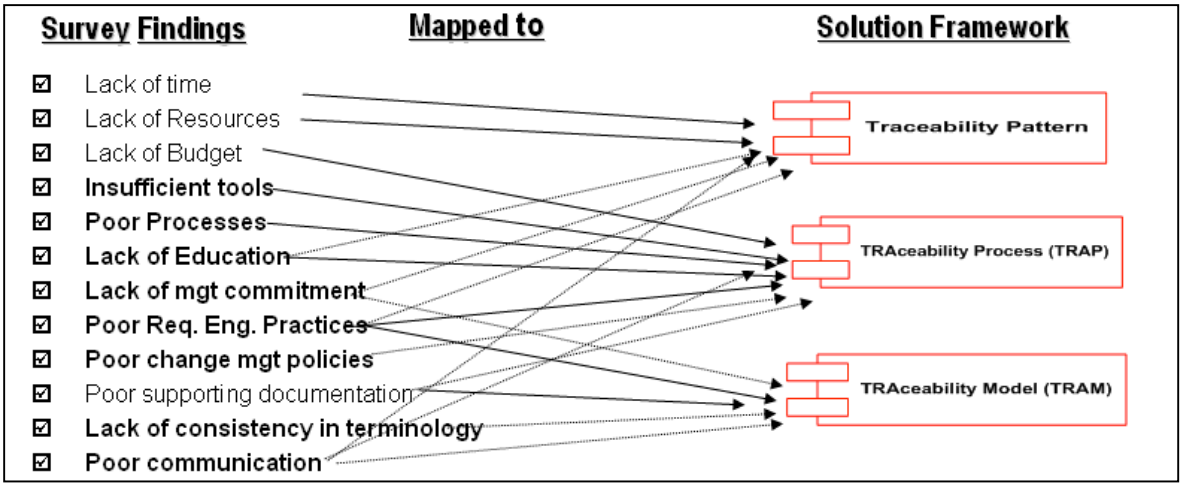



**Figure 5-18: Survey Findings Mapped to Solution Framework**

While it is difficult to execute a direct mapping between the factors that influence traceability in the above figure 18, *Survey Findings Mapped to Solution Framework*, we attempt to map the main findings to our Proposed Solution Framework. (we highlight the findings that are easily mapped to our solution)

A key part to any solution must be to provide a Generic Process Framework or Model that can be widely applied. With only 18% of respondents having traceability processes defined our solution had to address the definition of processes. The first question we asked on analysis of the data was: "how can we create a solution that could be generic enough to be used in a variety of industrial contexts?" The findings of this survey and the initial results from the case study motivated us to commence the development of the TRAP.

A second key building block in any wide ranging solution must be a semantic model of commonly accepted concepts. Many of the respondents complained of the lack in consistency in terminology used. A semantic traceability model or what we call TRAM had a primary objective of elevating this problem.

Finally, Alexander's stated that a pattern "describes a problem that occurs over and over in our environment and then describes the core of the solution to that problem in such a way that you can use this solution a millions times over without ever doing it the same way twice" In this survey we have followed a pattern approach for describing the problems and the findings of this case study. While this approach is quiet basic in this chapter we will elaborate on the power if traceability patterns in Chapter 9. For now it is sufficient to say that they provide an abstract description of the problem, while capturing experience and insights which can be reused in subsequent surveys or research efforts. The problems of poor communication and the lack of a common terminology further motivate the use of traceability patterns as a viable approach for describing traceability. Furthermore, while the patterns in this section are simply descriptive, they can also be used to describe core traceability concepts which can be used to overcome the training problems that emerged. Once we demonstrate the patterns that emerged during the TRAP and TRAM aspects of the solution the reader will further appreciate that patterns are a novel approach for training engineers on traceability principles and practices. Training courses can consist of a series of

traceability patterns leading to a better framework to describe the common problems and solutions to overcome them.

## 5.8 CONCLUSIONS & DISCUSSIONS

As discussed in Chapter 4 we commenced our case study with Ericsson Expertise Systems at the same time as we were designing the survey. One of the primary objectives of this survey was to have other sources of data to gain an understanding of the factors that influence traceability in small and medium sized companies so that the our solution framework takes into consideration data across a diverse group of industrial settings. While the case study gave an in-depth understanding of the traceability situation the industrial survey provided us with a multitude of other factors and opinions that gave us a broader understanding of the challenges that the discipline of traceability must overcome. The case studies gave us a deep applicable understanding of the factors that influenced traceability while the survey gave us an insight into the many problems faced by smaller organisations. In simple terms the case study is a more detailed assessment against one particular case while the survey gives us a more generic perspective. The combination of both survey and case study or quantitative and qualitative methods also assisted us to develop new theory from the data.

One of the challenges in studying small and medium software organisations is that there is great variation between practices, perhaps too great to be able to generalize findings and recommendations. Instead, therefore, this survey merely describes the situation as we discovered it. The sample represented in the survey offers a wide cross section of small to medium enterprises. It is not representative of the ubiquitous traceability situation, but it does nevertheless give an indication of their diversity of the state of the art for future empirical studies. It is this diversity which is most clearly demonstrated by the survey.

Even though the sample size was constrained by available time and resources, a systematic method was used and documented so that others who care to extend the sample size at a later date will be able to obtain results that are consistent with the method used in this survey. In Figure 5-19, *Taxonomy of Results from Survey*, we illustrate a conceptual map of our survey findings. The most important factors are difficult to provide a direct solution for, namely, lack of time, budget and resources. However, in our solution we address these factors as design considerations of TRAP, TRAP and traceability patterns. Factors like poor communication, lack of consistent terminology and the poor training situation are easier to address in our solution.

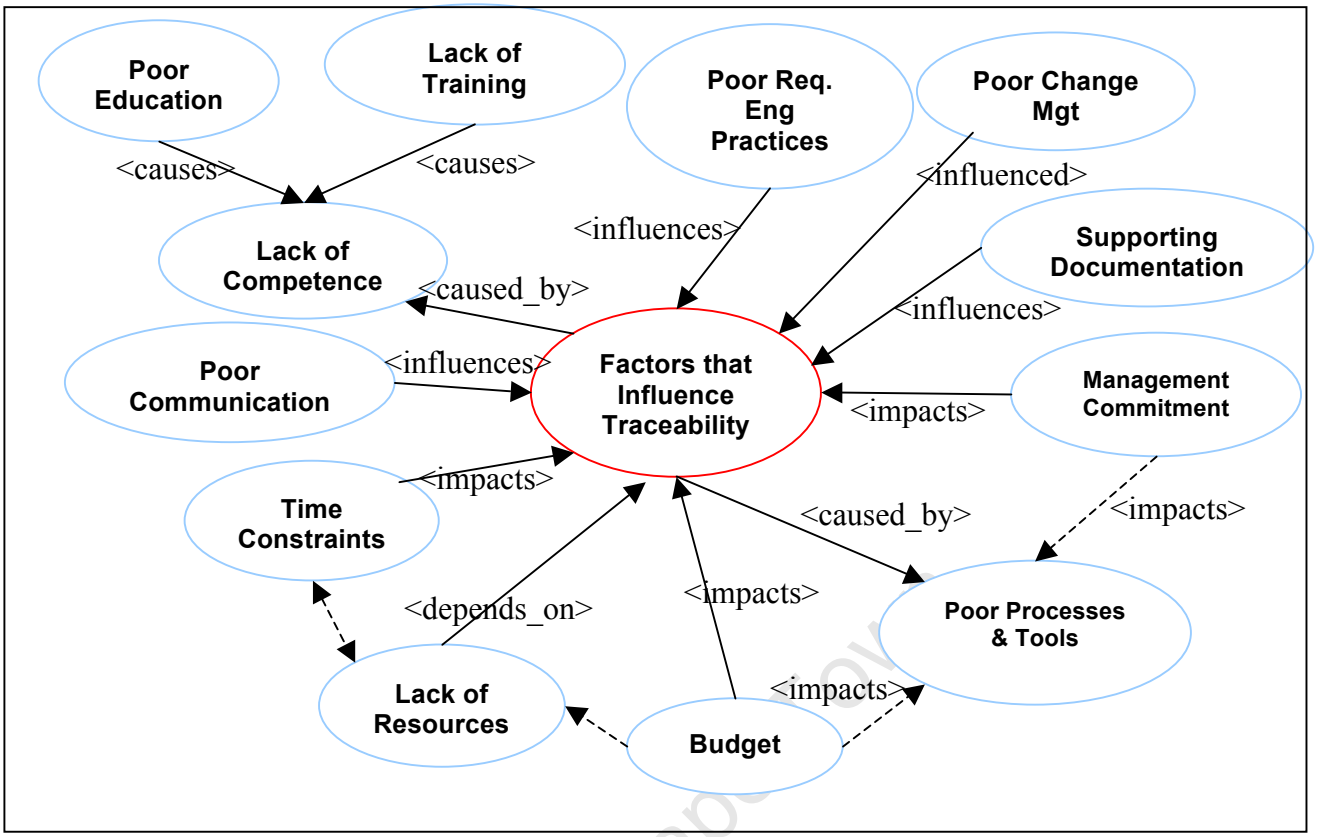



**Figure 5-19: Taxonomy of Results from Survey**

We have drawn the following conclusions about traceability practices, based upon findings from the survey:

- *Conclusion 1:* The link between education and the application of traceability was investigated. The dispersion of those who received traceability training at university level was still high. One could conclude that with the absence of dedicated training there is the increased risk of poor attitudes traceability practices.

One who receives third level, university or technical college, education on traceability, or has received specialist mentored training, has a much better attitude towards the practice and a higher probability of actually using best practise. This highlights the importance of the link between attitude, education, training, and practice.

- *Conclusion 2:* Another area impacted by these factors is the corporation's responsibility to provide training and professional certification programs to entice staff to continue employment in their organisation. During the nineties, the shortages of competent software skills meant corporations included competence development as an important aspect of any employment package. Today the availability and migration of highly educated, self motivated and highly skilled resources from the developing countries has meant that many organisations can now look for cheaper self-trained employees overseas. All this leads a crippling blow to the training of practices like traceability. The days of certification programs in requirement management for all resources in the development process are long gone. In all 22% of those interviewed had received some form of training in requirement engineering. It is interesting to note that the during the interviews 65% of those who had received training practiced traceability as opposed to the average of 46%.

- *Conclusion 3: Lack of Understanding of Terminology.* Even amongst those practising requirements engineering and traceability, a worrying finding that emerged in this survey was the lack of understanding of core concepts used to describe traceability. Overall, 38% of those interviewed admitted that they did not understand some of the terminology used during discussions with the researcher. This high proportion could best be attributed to a lack of training plus the lack of a glossary of terms.

- *Conclusion 4: Budgetary Constraints.* Most participants in the survey say that the greatest constraint on improving corporate practices in requirements engineering and traceability were budgetary; although lack of time is almost as serious. As the survey data indicates traceability is only practiced by 46% of resources in the development lifecycle, either formally or informally. In particular, small budgets adversely impacted the allocation of resources to requirement management activities, and reduced the acquisition of commercial of the shelf processes and tools.

- *Conclusion 5: Poor Processes.* Although almost 90% of participants said that their organisations had some sort of formalised development process, in most cases this proved to very rudimentary. For example, only about 40% of the participants belonged to companies that had formal process team, and even then, particularly in the micro and small companies it was no more than a part-time team. Moreover, in the small sized companies, little time was given to process review and improvement; then, only at irregular intervals, usually in between projects. Only a small minority of participants worked in organisations that had a formal process model for software development, and none of the organisations modelled their processes. Not surprisingly then, about 50% believed that Requirements Engineering was sufficiently well defined in their organisation, but often this was no more than a spreadsheet. Only 19% said that their organisations employed a Requirements Management Plan, or used some form of document to describe the traceability between the items.

In conclusion, the software industry is taking on new challenges in terms of the type of projects, products and services that they provide. This means that companies have to work more closely with customers and managers need to know a lot more about the technical challenges in their projects. In this study we illustrate that the motivation and interest in, and respect for engineering education and culture in the developed countries is rapidly decreasing. This has implications on many software engineering disciplines including traceability. Fewer people are available locally to do the technical work that is required. Moreover, a lot of the projects are only marginally profitable and therefore organization cannot afford to what would be regarded as extra non-essential roles, like requirement engineering. With naive understanding of the software development process, the organization crudely tailor now widely accepted best practices and it is from *this* that many of the 'traceability' challenges stem. An assumption by management that developers are able to take on a set of requirement specifications and then program them to meet the customer's needs is causing major problems in creating quality products. The question remains, however, are there any differences between the practices in large organisations and smaller organisations? As shown, in Figure 20, *Survey Mapped to Case Study* below, while there are many similar factors that influence traceability there are many major differences based primarily on the size of budgets and the availability of resources.

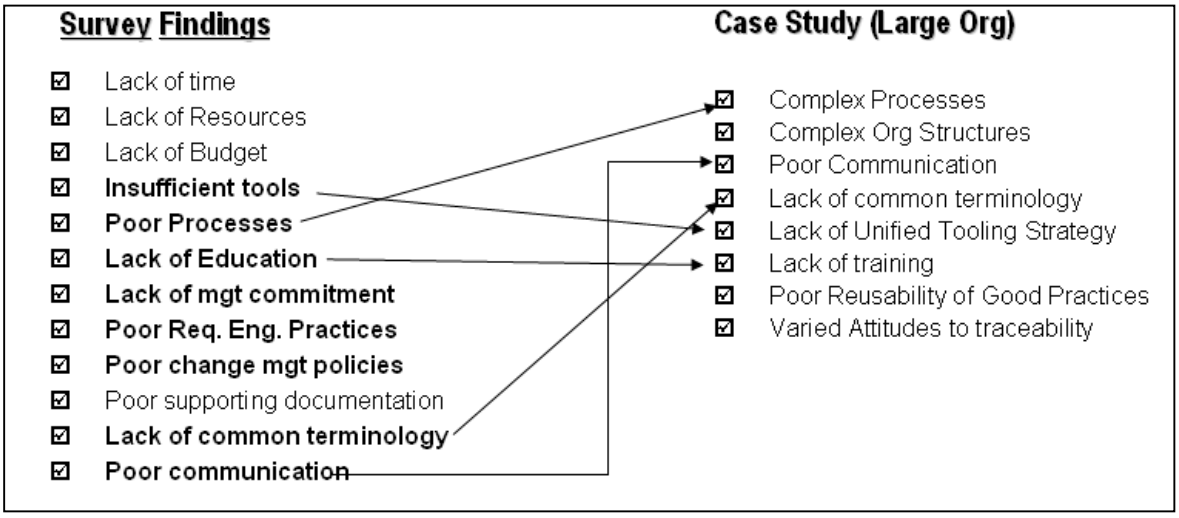



**Figure 5-20: Survey Mapped to Case Study**



And this is exactly what the most important contribution of this survey is. Not only did we gain a better understanding of traceability practices in a diverse cross section of industries, we gained a much deeper understanding of the problems that they face. Bearing in mind that although there are (always) complaints raised from designers and testers against 'management', about the expectations set by the management and the lack of processes, tools and training, it is interesting that they are still, on both sides of the 'cultural divide', comfortable with the performance of the projects. In many cases this comfortableness was the biggest factor for concern in the smaller organisations. The attitude of we don't have the time or budget or that overwhelming notion that implementing practices like traceability in later projects, is simply one of the main reasons that smaller organisations fail to practice core principles like traceability.

To generalize on the factors that influence implementing traceability is difficult, and yet there are a few clear findings that can be read from these results. In order for traceability to be practiced organisations must first understand the importance of this practice. Upper management must allocate budget for training and new traceability technologies. Project managers must allocate time to their resources for the implementation of the practice by including traceability as an integral part of development or test. All this must be supported by a simple to use and easily understood process.

In this chapter, we have reported on our experience and results from the industrial survey. Instead of focusing on one specific area we have tried to present the big picture: from processes to problems to tool usage. We feel that new approaches to implementing traceability must be provided to smaller organisations. We believe that industrial training and better university programs must be provided if this practice is to succeed in industrial contexts and research communities alike.

# Chapter 6 INTRODUCTION TO TRACEABILITY SOLUTION FRAMEWORK

## 6.1 INTRODUCTION

The purpose of this chapter is to introduce the main components of our proposed solution framework that, over the next chapters, we design, and then, in the final part of this research project, test.

Over the past decade, considerable progress has been made in the field of traceability, with: new concepts, new models, new processes, new techniques, and even new frameworks. A number of research efforts have used the term *framework* to illustrate their proposed solution.(Ramesh and Jarke, 2001, Sherba et al., 2003, Letelier, 2002) The problem of course is that the scope and scale of the problem is far beyond that which can be solved by a simple "silver bullet": so where to even start?

As illustrated in the previous chapters, many of the problems with the adoption of beneficial traceability practice arise because of:

1. Poor communication of traceability concepts between success critical roles, which arises partly from lack of common vocabulary.

2. Lack of a widely accepted body of knowledge, with training material that is standardised, leading to poor training and certification programs.

3. Lack of standards and poor documentation, which means that many implementation efforts start from scratch, thus raising the price of implementing traceability and increasing the cost of introducing good practice.

4. Poor documentation of successful practices, or even unsuccessful ones, so that there is poor reuse, and as a result, when projects end, there is a high probability that the experience and expertise gained will be lost.

In addition, there are many diverse ideas about how to effectively implement traceability. The sources of this body of knowledge is shown in Figure 6-1, *Inputs to Traceability Framework* and is described as:

1) Corporate efforts by software development, product and technology companies (tools and processes)

2) Research efforts (literature, research projects, empirical studies) by academia and software research groups;

3) Industry knowledge and a variety of consulting methodologists and companies capturing industrial best practices into various knowledge bases

4) Empirical Data

So the question becomes one of how to impose some sort of ordered approach to the field of traceability? The simple truth is it needs a framework; both a structural plan for the rest of this project, and a conceptual plan for the components of a solution.

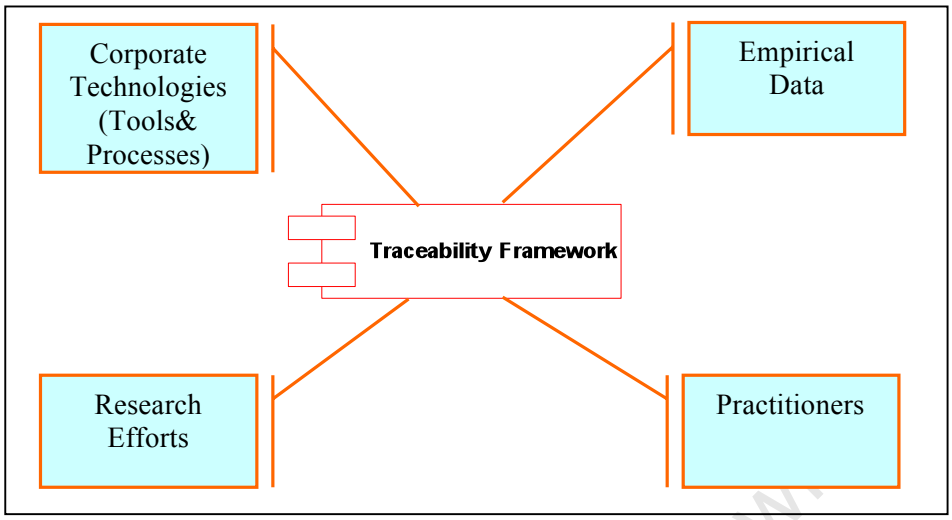



**Figure 6-1 Inputs into Traceability Framework**

While these sources continue to develop and progress the traceability discipline there is a major need to provide a framework where the different efforts can build their ideas into unified solutions or one solution framework that is reusable. That is the primary motivation for creation of the Traceability Framework.

The idea of building a Traceability Framework, however, is not in itself a new idea, and in what follows, we briefly discuss some of the previous approaches that have failed to succeed and note why.

Our approach aims to address the problem in a framework with 4 major branches:

1) the problem of poor communication we aim to overcome with a new semantic model for traceability called TRAM;

2) the problem of lack of a unified process model that enables traceability through the development life cycle, we aim to overcome with a process model called TRAP;

3) the problem of lack of a collective memory and best practice, we aim to overcome progressively with the aid of a knowledge data base called TRAPT (TRAceability Pattern Tool);

4) the problem of recognising best practice and training for it, we aim to address through the recognition of patterns and pattern-creation tools.

Our proposed solution framework seeks to combine these components, namely a semantic model, a process model, a knowledge encyclopaedia and traceability patterns into one unified solution. In it we integrate guidelines, relevant literature, semantics, processes, best practices, and empirical data into one solution framework.

This however, is not just a theoretical modeling endeavor. Later in Chapter 11 we test the framework in the field of telecommunications, in particular the Ericsson OSS domain. The purpose in applying the framework to a particular domain is to gather results, evaluate

the framework's effectiveness, and then determine its suitability as a "tried and tested approach and tool suite".

We are not suggesting that our proposed Traceability Framework should become the new traceability standard. However, in order for traceability to be widely accepted as a core practice, a framework similar to the one we propose needs to become the unifying standard for all software development projects and research efforts worldwide. Perhaps the sheer magnitude of this challenge is too much for a research degree in philosophy to accomplish but without humble beginnings there can be no unification.

In the sections that follow we briefly describe the proposed framework and 4 component branches, and the fundamental thinking and rationale behind them and their design philosophy, in order that the reader can understand why they are as they are, before confronting mass of much greater detail in later chapters.

## 6.2 TRACEABILITY FRAMEWORK ENVIRONMENT (TRAFE)

This section provides the reader with a brief conceptual overview of the whole package. In Figure 6-2, *Traceability Framework Environment (TRAFE),* we depict the 4 main components of the **TRA**ceability **F**ramework **E**nvironment (TRAFE); the TRAceability Model (TRAM), the TRAceability Process (TRAP); Traceability Patterns and the Traceability Environment Interface. The TRAFE organizes the complex components; the concepts, processes, technologies, models and patterns relevant to this research project into a single unified structure. This experimental platform integrates the components and outputs them to a web interface.

Using TRAFE, a user can navigate through the different layers of the TRAM and TRAP. Initial investigations were carried out on utilising Topic Maps (ISO/IEC, 2003) (ISO/IEC, 2006, ISO/IEC, 2007b) to represent the patterns. Topic maps are an ISO standard first published as ISO/IEC 13250 in 1999. The standard further specifies interchange syntax based on SGML and the Hypermedia Time-based Document Structuring Language (HyTime). (Charles F. Goldfarb et al., 1997) HyTime allows documents to package their information content using standard "mark-up". This mark-up provides information of the structure and notations of the document in a way that we can use in the traceability domain. After publication of ISO13250, a private consortium of independent parties, topicmaps.org, was formed to create a web-optimized topic map syntax based on XML and URIs. This syntax was published in 2001 as XTM 1.0 and was included as an annex to the second edition of ISO 13250. (Pepper and Moore, 2001) In summary, ISO/IEC 13250 is the standard and XTM is the specification. We investigated using Topic Maps to represent traceability relationships, and for navigating between traceability relationships. (Kelleher et al., 2005)

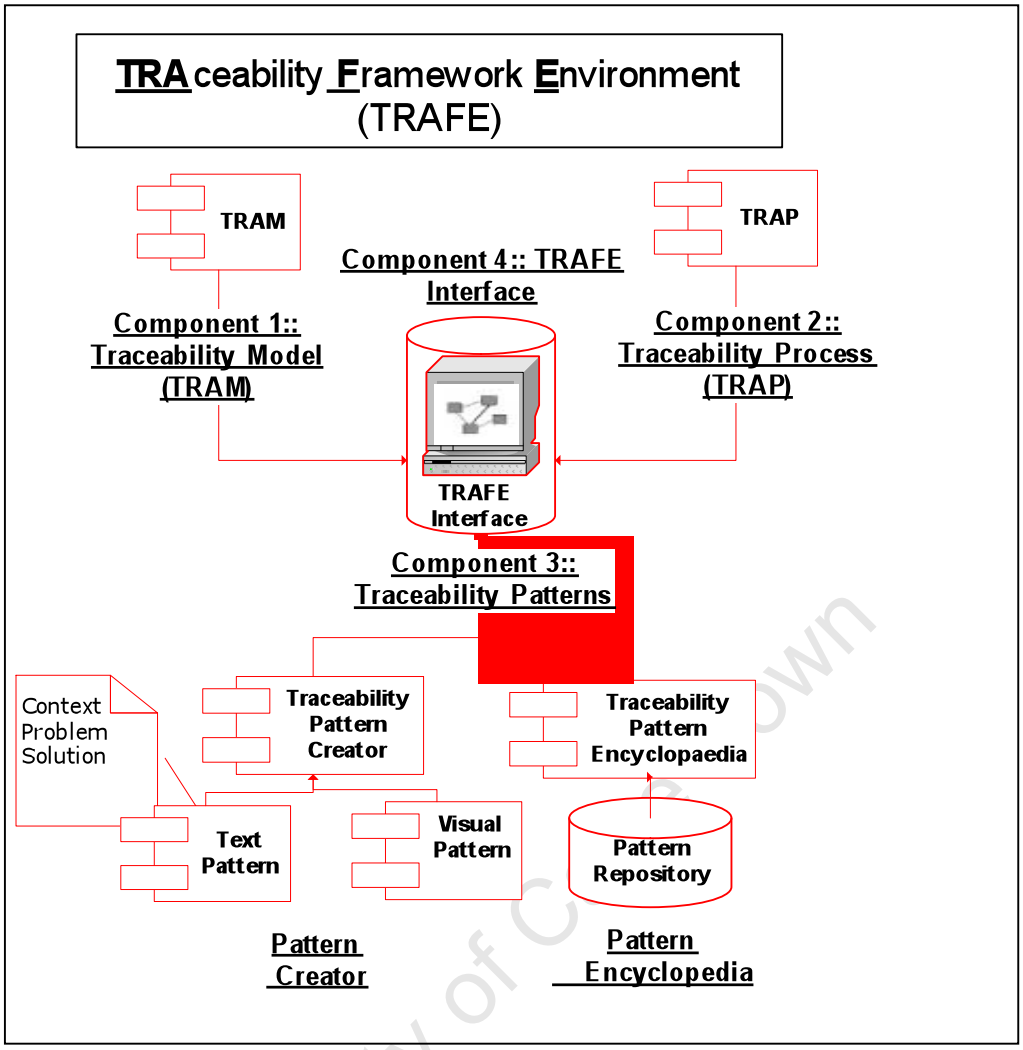



**Figure 6-2 TRAceability Framework Environment (TRAFE)**

The Traceability Pattern Tool (Kelleher et al., 2004) (Kelleher, 2005b) is a tool for the structured and collaborative creation and cataloguing of software traceability patterns. It is subdivided into two sub-components the Pattern Creation Tool and the Pattern Encyclopaedia. The Pattern Creator is divided into pattern definition and pattern management. The Pattern Creator module has pre-defined pattern forms to assist in the creation of the patterns. The Pattern Encyclopaedia module provides summaries of the latest literature as well as academic papers and articles related to patterns.

## 6.3 FUNDAMENTALS OF PATTERNS & FRAMEWORKS

### 6.3.1 Patterns & Frameworks

Today, analysis patterns, architectural patterns, design patterns and testing patterns are used by many industries in varying contexts. Each domain or technology uses patterns to capture experience and knowledge, then to impose order on a complex field, and to convey that knowledge to others.

Patterns are typical examples of concepts, designs, models, and ways of doing things that can be used repeatedly, over and over again, because they encapsulate good basic principles and allow newcomers to the field to grasp these basic ideas as common currency and to build upon them.

The renowned architect, Christopher Alexander, has offered an instructive definition of patterns: "Each pattern describes a problem that occurs over and over in our environment and then describes the core of the solution to that problem in such a way that you can use this solution a millions times over without ever doing it the same way twice". (Alexander et al., 1977). He maintains that the essence of a pattern is a sketch, and that each pattern should have a characteristic name, by means of which it is instantly recognised and visualised as an idea.

Because patterns are named, individuals can use those names to refer easily to that idea, and so the names become a compact and concise way of referring to a set of designs and design decisions, while suppressing any "details not relevant at a given level of abstraction. (Coplien, 1996)

The use of patterning is fundamental to conceptual modelling of both the development process (TRAP) and the traceability process (TRAM). In each case we make use of pre-defined *templates* called *forms*. A particular pattern is defined by inputting specific values into the form fields. For example, in formulating TRAM we discovered that each traceability item has a name, a type, attributes, link types, relationship types and a tag or unique identifier. We call this the *Traceability Item Pattern*. No matter what traceability item is being created we can reuse the Traceability Item pattern to create a traceability item. In Figure 6-4, *Traceability Patterns: Process Model & Semantics*, we illustrate that patterns emerge from the TRAM and the TRAP and are best used to communicate on matters related to traceability.

A fundamental problem arising with conceptual patterns and models is, however, the degree of abstraction. Too great a degree of abstraction renders the sketch or illustration incomprehensible without a lot more explanation; in fact a mere symbol. Christopher Coplien suggests that more specific graphical representations are by definition better. (Coplien, 1997) However, too much detail, risks losing the user in a morass of it, so that he fails to see the essentials of the concept or design. Herman et al believe that the use of conventional UML diagrams leads to such problems, with over specification and a consequent loss of the abstract nature of patterns. (Herrmann et al., 2003). Nevertheless, it is almost always necessary to explain a simple pattern with a large amount of prose, so one must endeavour to strike a succinct balance. Below we employ prose and object oriented graphical views to depict the traceability patterns for TRAM and TRAP.

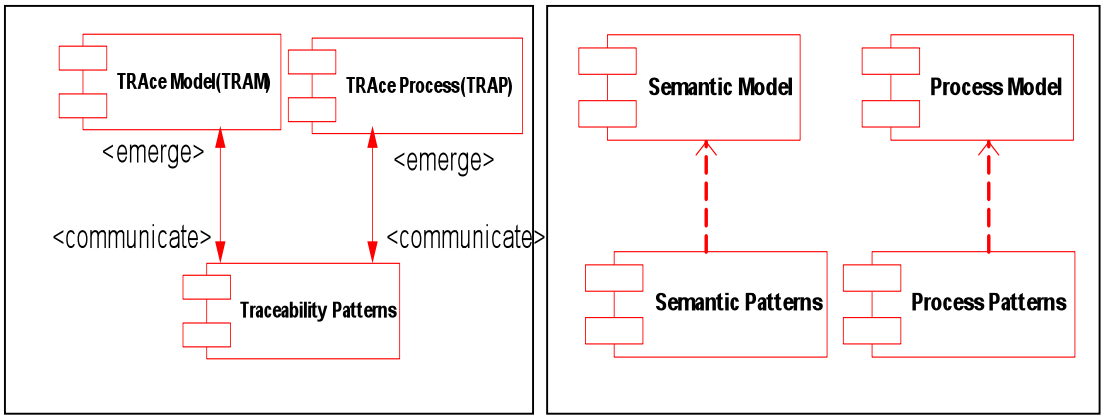



**Figure 6-3 Traceability Patterns: Process Model & Semantic Patterns**

Patterns are particularly useful in imposing structure on complex subjects, and for getting over structural concepts, command and organisational relationships and ideas about other hierarchical systems. A group of structural patterns for that very reason are often called a framework. A set of architects drawings for a new building such as a church can be thought of as a set of patterns for the planned shape of the building, its structural framework, and standard patterns for the doors, windows, roof and spire, and internal features like the sanctuary, baptistery and so on. The framework, however, holds everything together.

In the field of software engineering, Coplien has defined a structured collection of patterns that build on each other to create an architecture as a *pattern language*. (Coplien and Schmidt, 1995) The logic of this analogy follows that of language itself, in that a word is a *unique pattern of symbols in a serial sequence* and a sentence is a collection of such patterns arranged according to the grammar of the language in question. This helps to make the point that design patterns for architecture, software or language can be viewed as groups of simple patterns arranged in a logical order.

Frameworks are thus closely related to Patterns; both are essential to the philosophy of reuse. In essence, patterns may be employed both in the design and the documentation of a framework. A group or catalogue of patterns in a particular domain or context could be referred to as a framework. However, frameworks are in general more structured, more specific and tailored than patterns. Thus,

1. *Patterns are more abstract than frameworks*.

2. *Patterns are small and simple in nature while frameworks generally are larger and more complex.*

3. *Patterns are of a more logical nature, whereas frameworks are of a more physical nature.*

In the software engineering domain this has come to mean that a framework consists of components that are executable in nature, while software engineering patterns represent knowledge and experience about certain software engineering practices. Therefore frameworks are the physical realization of one or more software pattern solutions; patterns are the instructions for how to implement those solutions in code. Thus frameworks can be represented in code, while patterns are usual examples of best practices or good solutions of designs for code. Because frameworks are generally defined using

visual modelling languages like UML, which is executable, then technically the framework becomes an executable entity. In summary, frameworks can be viewed as a concrete reification of families of patterns that are targeted for a particular application-domain. These, somewhat abstract ideas, form the basis for the later section in the chapter that extends the application of patterns and frameworks to traceability practices.

### 6.3.2 What is a Framework?

Like many software engineering principles, the term *Framework* has been loosely used in the past decade. In many cases one would need to analyse the scale of the framework, the context that it is set and the components that bring it together before it should be given the title Framework. There are architectural frameworks, design frameworks, process frameworks, semantic frameworks and a plethora of domain specific frameworks from knowledge management to test management.

A software framework is a *reusable mini-architecture* that provides the generic structure and behaviour for a family of software abstractions, along with a context of metaphors which specifies their collaboration and use within a given domain(Appleton, 1998)

The term architecture is generally used both to refer to both an architecture description and an architecture implementation. An architecture description is a representation of a current or postulated real-world configuration of resources, rules, and relationships. Once the representation enters the implementation phase of the system development life-cycle process, the architecture description is then transformed into a real implementation of capabilities and assets in the field. Therefore, software architectures provide high-level abstractions for representing the structure, behaviour, and key properties of software systems. These abstractions are useful in describing to various stakeholders complex, real-world problems in an understandable manner. Software architectures are described in terms of components, connectors, and configurations. An architectural style defines a vocabulary of component and connector types and a set of constraints on how instances of these types may be combined. (Ramesh and Jarke)

With the global economic slowdown and corporation tighter budgets, frameworks are an attractive form of reuse due to their paradigmatic simplicity. Many companies are moving away from constantly building applications from scratch, and instead focus their development on building a reusable and testable framework to encapsulate their business rules. A design framework is a skeletal group of software modules that may be tailored for building domain-specific applications, typically resulting in increased productivity and faster time-to-market. A design framework is a set of common and prefabricated software building blocks that programmers can use, extend or customize for specific computing solutions. With frameworks developers do not have to start from scratch each time they write an application. Frameworks are built from collection of objects so both the design and code of the framework may be reused. This approach has lead to creation of a variety of "middleware" techniques and associated commercial technologies for component-based development. Thus design framework captures the design decisions that are common to its application domain. Frameworks thus emphasize *design reuse* over code reuse, though a framework will usually include concrete subclasses you can put to work immediately. (add in all references)

A Process Framework is a skeleton that specifies and coordinates the various processes necessary to complete a complex task. In software development the process

frameworks are primarily concerned with specifying what processes are necessary. The Rational Unified Process (RUP) is an excellent example of a Process Framework. The RUP provides a framework of rules and practices that mitigate the unpredictability of software development. RUP is an adaptable process that should be tailored by the development organisation that need to select the elements of the process that are needed to meet their projects needs. RUP consists of three central elements. Firstly, it is an underlying set of philosophies and best practices for successful software development. Secondly, it has a process model from which you create your own process configurations. Finally, it consists of an underlying process definition language or simply a process meta-model. This model provides a language of process definition elements for describing a software engineering process. This language is based on the SPEM extension to the UML for software process engineering and the Unified Process methodology (Ramesh and Jarke). The RUP Framework and especially its components had a major influence on our approach.

In computer science disciplines semantics is the field concerned with the mathematical study of the meaning of programming languages and models of computation. The semantics of a programming language is given by a mathematical model that describes the computations described by that language. The Semantic Web is an example of an evolving extension of the World Wide Web. In simple terms the semantic web comprises of a philosophy, a set of design principles, a collaborative working group, and a variety of enabling technologies. A semantic framework is a formal approach which abstracts core notions which are shared by various collaborating approaches into natural language semantics. The computational advantages of such a framework are that it enables the implementation of modules relating to the core notions, basically you can create executing code from notions.

Domain specific frameworks are now in every day use. For example, the Software Engineering Institute defines a *Framework for Software Product Line Practice* which is a Web-based, living document that aids the software community in software product line endeavors. Each new release or version of this framework represents an incremental attempt to capture the latest information about successful software product line practices. This information has been gleaned from studies of organizations that have built product lines, from direct collaborations on software product lines with customer organizations, and from leading practitioners in software product lines.

Frameworks are a valuable approach for gathering notions, processes, guidelines and novel concepts into one unified platform. Unfortunately today in many cases the size and complexity makes understanding how to use them difficult. In addition documentation to support framework reuse often lacks experimental validation and there is little understanding of how to increase their effectiveness.

In summary in this section we introduce the concepts and philosophy behind frameworks.

### 6.3.3 Previous Research Efforts in Traceability Frameworks

Several types of Traceability Frameworks have been defined in the last decade. In 2001 Jarke et al created reference models comprising of the most important kinds of traceability links for various development tasks. These reference models were based on empirical data gathered using focus groups and interviews conducted in 26 major software development organisations. The resulting models were validated using case studies. They

continue to describe how a case study used the particular models. Four different types of traceability link types were identified. (Jarke, 1998)

Letelier at ACM TEFSE 2002 described a Traceability Framework. The aim of his work was to present a framework for configuring requirements traceability by integrating textual specifications and UML model elements. His approach is generic and could be applied to any software process based on UML which could be customized according to the specific traceability needs of any project. He describes a metamodel for requirement traceability, explains how textual specifications and traceability links can be defined in the UML context and the application of this framework using RUP. (Letelier, 2002)

In the paper "PRO-ART: Enabling Requirements Pre-Traceability", PRO-ART is defined as a three-dimensional framework for requirements engineering which defines the kind of information to be recorded included; a trace-repository for structuring the trace information and enabling selective trace retrieval; a novel tool interoperability approach which enables (almost) automated trace capture (Pohl, 1996)

### 6.3.4 Framework Design Considerations & Principles

A Traceability Framework must encapsulate and integrate a set of key concepts, knowledge, models, processes, structures, practices and guidelines into one unified workspace.

A prime goal of this framework is to promote better definition of traceability concept hence simplifying complex concepts, promote better communication of new techniques. It must therefore:

- unite the semantics, terminology and notions into one single traceability portal.

A second important goal is to have as wide an application as possible for the models of the development and traceability processes, so that any organisation, small or large, can work with them and tailor them to their own processes, depth of detail and desired level of capability in traceability. The models must therefore must:

- allow a simple mapping of relationships and dependencies between different components;
- be simple and easy to use;
- modular and easily configured and tailored to meet the needs of every software development project at a level of detail appropriate to the needs of the user;
- re-useable, extendable and adaptable, so that maximum use can be made of the best efforts of others, and the models can be extended and adapted readily to meet changing practice from project to project, product to product, standard to standard.

Thirdly, the framework has to play and important educational and training role, and so it must provide:

- a repository for describing the problems that may be encountered during product development implementing traceability, and for describing suitable solutions or even unsuitable solutions to these problems.
- a tool for capturing traceability knowledge, techniques or actual practices in a simple, easy to use environment.

Finally, it must provide a management user friendly interface and reporting system so that the traceability process tool is not just a "magic black box". Thus it must be:

- observable and readily support monitoring and analysis, when applied in a particular domain, in particular for tracking the frameworks performance and correctness, and for anticipating and detecting faults that can be corrected.

- able to provide accurate and comprehensible documentation, making maximum use of patterned templates for presenting information and reports.

## 6.4 TRACEABILITY FRAMEWORK COMPONENTS

In this section we introduce the three main research components and their underlying concepts that encapsulate the Traceability Framework that we propose. As illustrated in figure, *Traceability Framework Components*, below the components are, a **TRA**ceability **M**odels (TRAM), a **TRA**ceability **P**rocesses (TRAP) and Traceability Patterns.

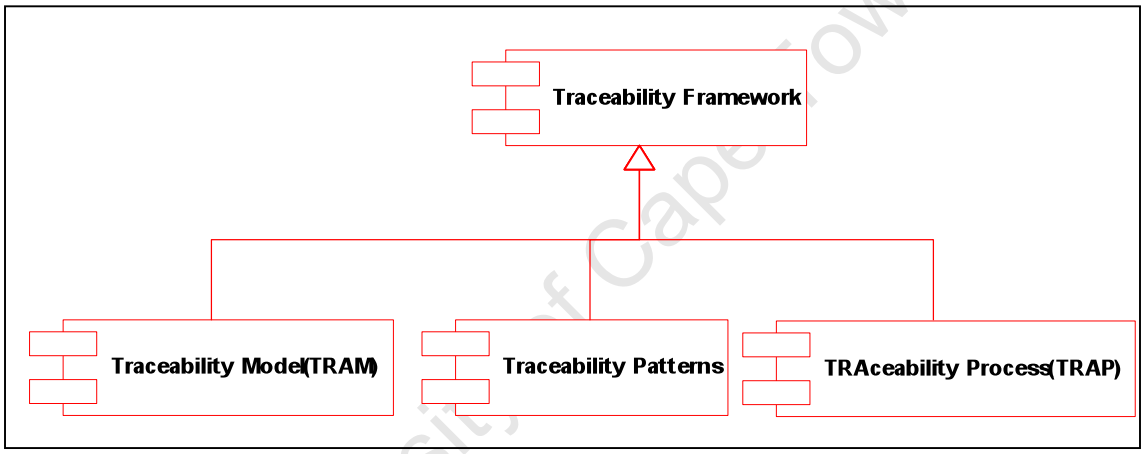



**Figure 6-4 Traceability Framework Components**



### 6.4.1 Component 1: TRAceability Model (TRAM)

*Model* is the generic term used for visual diagrams that describe problem domains regardless of their use or level of detail. Models add meaning to concepts, capture data, encapsulate knowledge, depict scenarios, events, actions or activities, act as blue prints for software design, illustrate real-world supply chains, describe the test cases and can be used to replace legal contracts between clients. There are process models that describe product life-cycle processes, business processes, requirement management processes, analysis processes, design process, test process and supply chain management processes. A *process model* identifies the common, generic features of processes and represents them as a system of concepts. (Rolland, 1998)

Models have layers or levels, each layer describing the problem domain at a different level of abstraction. Often the topmost layers are called *semantic models*. Semantic models offer a higher layer of abstraction where we can add meaning or define concepts to a particular domain.

The name for each layer, its objective or its structure is often pre-defined by a modelling standard or user community. With the formation of the Object Management

Group (OMG) the standardization of the layers, their semantics and model concepts has taken place. The OMG describes a Model Driven Architectures (MDA) to provide a conceptual framework and a set of standards to express models, model relationships, and model-to-model transformations in a platform independent manner. The MDA as defined by the OMG is a four-layer architecture which provides a solid basis for defining a model of any problem domain. Domain Specific Model Driven Architecture (DSMDA) is the term used to describe the use of MDA for a particular problem domain. For example, we utilise DSDMA for building our models of traceability in the telecommunications domain.

In the 1970s Data Flow Diagrams, Entity Relationship Diagrams and Structured Analysis Diagram were being used to model systems. During the 1980s object orientation was introduced. By 1990, more than 50 object oriented modelling languages were in common usage culminating with the standardization of the Unified Modelling Language (UML) as the de facto modelling standard.

In Figure 4 below we illustrate the four layered architecture which we used in our research. The diagram accentuates some of the basic modelling concepts that are pivotal to the rest of this paper. We call the unified four layers the **TRA**ceability **M**odel (***TRAM)***.

The primary objective of the TRAM architecture is to simplifying the concepts of traceability in the OSS-RC domain. Each layer adds semantics to traceability at different levels of abstraction. For example we model the traceability items, the relationships, the traceability tool, the attributes and the traceability views or matrixes. In the context of this study we use the terms "metamodel" and "model". A metamodel is a model that explains a set of related models. A metamodel in any domain is a model which describes a language with which models can be expressed. It is a precise definition of the constructs and rules needed for creating semantic model elements at a high level of abstraction. A metamodel serves as a template for a model or in other words a model is an instance of a metamodel.(Colin Atkinson et al., 2001) The metamodel describes the set of modelling elements which are represented as metaclasses with relationships described using meta-association. In simple terms the metamodel gives us the rules and semantics for modelling traceability in the OSS-RC product development lifecycle. A model is an instance of a metamodel, while a metamodel is an instance of a meta-metamodel. (OMG, 2005)

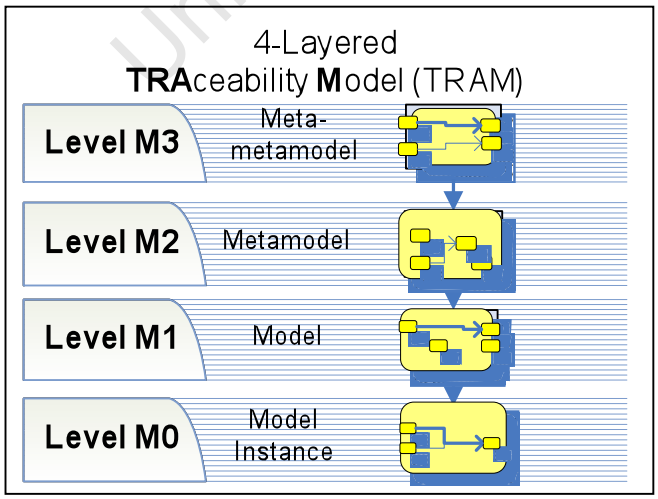



**Figure 6-5 TRAceability Metamodel (TRAM)**

The Object Management Group consortium defines the Meta-Object Facility (MOF) standard which is designed as a four-layered architecture and is as an abstract language and framework for specifying, constructing and managing technology neutral metamodels. (OMG, 2005) It is the meta-metamodel (called M-3 models) that is the foundation for defining any modelling language, and is typically more compact than the metamodel that it describes. The meta-metamodel layer forms the infrastructure for the metamodeling layer while the metamodel layer forms the infrastructure for the model layer and so on. The primary responsibility of the MOF is to define the language for specifying a metamodel. There are similarities between the MOF M3-model and the UML structure models therefore MOF metamodels are usually modelled as UML class diagrams. In our research we begin by creating M-2 models using UML Class diagrams. Our M-2 metamodel is a precise definition of the constructs and rules needed for creating semantic model elements at a high level of abstraction. The M-2 models will use UML class diagrams to define all the traceability concepts in a technology neutral way. This means that it describes all the traceability concepts without describing the OSS-RC technology. It is technology neutral and can be reused for any domain. The M-1 and M-0 models capture technology specific information from the OSS-RC domain.

A similar framework was proposed by Mason called MATrA (Meta-modelling Approach to Traceability for Avionics) which enabled traceability links to be established and consistency maintained across data from potentially disjoint tools. (Paul Mason et al, 2003)

### 6.4.2 Component 2: Traceability Process

The Software Engineering Institute (SEI) states that "An essential aspect of software engineering is the discipline it requires for a group of people to work together cooperatively to solve a common problem. (SEI, 2007b) Defined processes set the bounds for each person's roles and responsibilities so that the collaboration is a successful and efficient one" Rational defined a process as "a set of partially ordered steps intended to reach a goal. In software engineering, the goal is to build a software product or to enhance an existing one". (Kruchten, 2003)

The field of traceability traverses a wide range of software engineering disciplines including requirements, analysis, design, implementation and test. The marketplace is in a constant state of flux, the customer's needs change, the stakeholders are dispersed across complex organizational structures, the development organisation is fragmented across multiple sites, the project budgets change impacting the project structure, communication between varying roles is difficult, documentation is incomplete and so on. Processes must be in place to manage this complexity. A software process is composed of phases, activities, artefacts and roles.(Kruchten, 2003)

In this study we define a Traceability Process Framework which we call the **TRA**ceability **P**rocess (**TRAP**). TRAP has the fundamental objective of describing how to manage traceability. It describes the roles and their corresponding responsibilities, the artefacts that contain the traceability items, the change control process, the product development lifecycle model, the impact analysis process, the baselining process and the guidelines for implementing the practices.

Lee Osterweil wrote in 1987: “Software processes are software, too”. (Osterweil, 1987) Processes can also be modelled. To promote reuse of processes a process metamodel identifies “the common, generic features of process models and represents them in a system of concepts” (Rolland, 1998)

As with any successful technology there are many new specifications outlining how to create a process metamodel. For example the OMG presents the Software Process Engineering Metamodel (SPEM).(OMG, 2005) SPEM is object-oriented specification which describes how to model a software process. The SPEM is a metamodel for defining processes and their components. The SPEM specification states “This metamodel is used to describe a concrete software development process or a family of related software development processes…… taking an object-oriented approach to modelling a family of related software processes and we use the UML as a notation”

TRAP is also a four layered architecture. Similarly to the TRAM model at level M3, the MOF is an abstract language and framework for specifying, constructing and managing technology neutral metamodels. In Figure 6.7*, Example of TRAP,* below we illustrate a simple conceptual example of TRAP. We begin with an M-2 Metamodel of the generic process “concepts”. At Level M-1 we create a number of models that can instantiated from the M-2 process metamodel. These models describe traceability from an OSS-RC product process perspective. The models contain classes, logical packages, objects, operations, component packages, components, processors, devices, and their corresponding relationships. Each of the model elements possess model properties, which identify and characterize them.



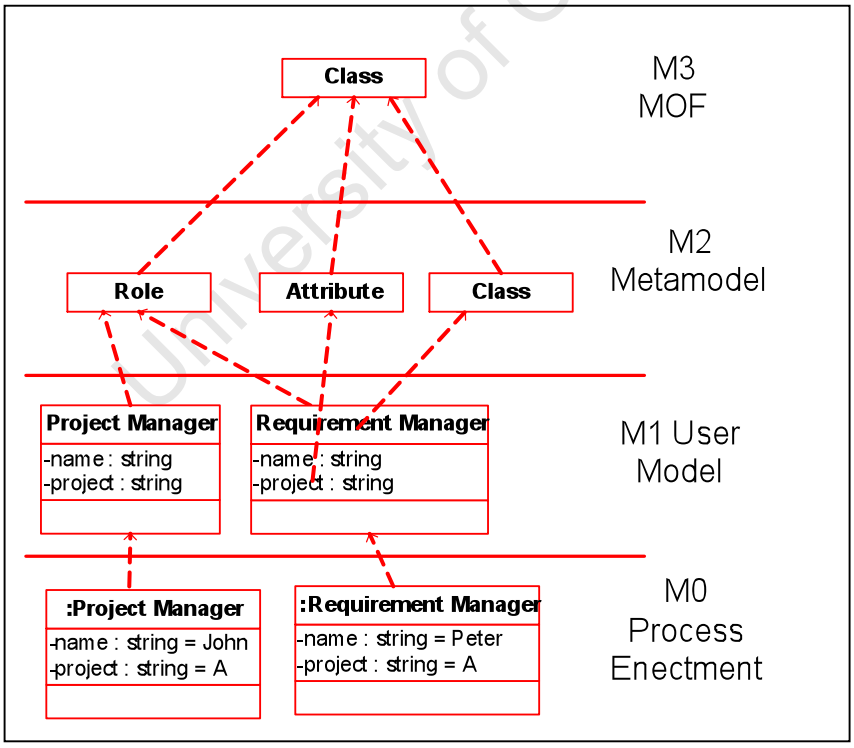



**Figure 6-6 Example of TRAP**

The models also contain diagrams and specifications, which provide a means of visualizing and manipulating the model's elements and their model properties. Since diagrams are used to illustrate multiple views of a model, icons representing a model element can appear in none, one, or several of a model's diagrams. This enables you to control which elements and model properties appear on each diagram.

The Level M-0 is an object model of the process as enacted by a specific project (OSS-RC R6). At M-0 we address the reasoning behind certain activities taken as a user works through each traceability task. We apply task analysis techniques to identify problems or successful solutions as they emerge in the application domain. For example we develop profiles of users such as their level of expertise (e.g. novice, intermediate, expert) which is known as *user modelling*. These models were constructed from user interviews and observations made during the case study. Modelling the user in this way helps determine the level of traceability expertise of the roles. For example, the senior architects have a better understanding of the importance of tracing to customer requirements than a junior designer with an implementation view.

TRAP describes the work products, the roles involved in creating the work products and the responsibilities in terms of activities that each role must perform to implement traceability in the product development lifecycle. The TRAP contains workflows conveying the development time for the product as a sequence, the traceability best practices and traceability guidelines (or literature) and traceability process patterns.

There are several reasons why using a formal process framework is beneficial and intuitive approach to traceability implementation. Some of the more obvious benefits include:

- Effective ‘tried and tested’ techniques can be used rather than inappropriate or untested techniques. This will lead to a more effective and more usable traceability approach.

- The roles involved in traceability can focus on the content of the traceability technique rather than trying to determine what needs to be done at each stage because the key tasks at each stage will have already been identified.

- The organisation is provided with a complete picture of the full traceability process, which can then be used to assess progress.



### 6.4.3 Component 3: Traceability Patterns

It is not uncommon for practitioners to have little or no experience in traceability practices. Traceability patterns aim to make traceability concepts and practices understandable to project team members in the product development cycle.

A Traceability Pattern Catalogue is a collection of traceability patterns. Traceability anti-patterns represent a "lesson learned." We argue that there are two types of traceability anti-patterns. There are patterns that describe a bad solution to a problem. These documented patterns will act as a “stop” sign for proceeding in a certain situation. While there are patterns that describe how to get out of a bad situation and how to proceed from there to a good solution. As we dedicate an entire chapter to patterns, this is sufficient information on traceability patterns at this point.

## 6.5 VALIDATION CRITERIA FOR FRAMEWORK

### 6.5.1 Benefits of Traceability Framework

The Traceability Framework is a collected body of traceability practices, processes, tooling guidelines and best practices. The benefits to the practitioner includes that it provides an understanding to individual responsibilities with regard to the implementation of traceability techniques. It provides a glossary of terminology and an encyclopaedia of knowledge to help you communicate traceability effectively with the project team members. The process component provides a central, common process definition that all organisations can share, helping to ensure clear and unambiguous communication of traceability principles. It provides a wealth of guidance on traceability practices that novice and experienced practitioners alike will find valuable. Even if you are a lone traceability practitioner, you would be able to take this framework and find it a useful mentor in helping you to build traceability solutions. For process engineers, this framework provides you with a good architectural foundation and wealth of material from which you can construct your process models, enabling you to configure and extend the framework as desired. This will save you enormous amounts of time and effort that would otherwise be required to create such a process definition from scratch

Other benefits include:

- Because it is based on proven object oriented technologies, it enables us to build a common platform for communication on traceability.

- Componentization is built into the traceability framework. This inherently supports reuse of processes, models, practices and guidelines for future projects.

- Effective 'tried and tested' techniques are used rather than inappropriate or untested techniques. Those using the framework are guaranteed that the approach they are reusing has been tested with results.

- The framework can be used as a training framework. Novices to traceability can use the glossary and semantic aspects of the framework while the more senior user can refresh their knowledge with the different models and configure them to meet their specific needs.

- The roles involved in traceability understand their responsibilities with regard to the implementation of traceability.

- Structured: The traceability framework of the models provides an excellent structure that organizations can follow. Furthermore, the structure helps everyone be on the same page because they can see what is expected.

- Auditable: Without having a standard framework, it becomes difficult for auditors, to effectively assess improvements in traceability practices. The goal must be to at least certify the organization against at least one base standard and then make recommendations over and above the standard(s), where appropriate.

- A framework also has the added benefit for comparing and integrating different frameworks against. Furthermore, other frameworks can be checked for completeness against this framework.

## 6.6 CONCLUSIONS

As discussed in the previous in this chapter, there are lots of specific approaches in the development of different types of frameworks. In essence, we are setting the context for the Traceability Solution Framework, which we design, build and test over the coming chapters. The main objective of this chapter is to orient the reader to the main concepts, the underlying architecture that the framework is built on, the design principles and the main components that encapsulate the framework.

# Chapter 7 TRACEABILITY MODEL (TRAM)

## 7.1 INTRODUCTION

In the past two decades two major paradigms have contributed to the discipline of traceability: the emerge of traceability research communities and new commercial tools offering better traceability functionality supported by commercial-of-the shelf processes. However, as the results of our case study and in particular the survey illustrates there are still many problems that must be overcome for continued success and improvements into the future. For example, the findings from the survey illustrated that inconsistencies in the terminology used to describe traceability often leads to misunderstandings and miscommunication between success critical roles. A traceability item to one person is a requirement to another or configuration item to another. Furthermore, organisations, especially smaller organisations lack the budget to invest in expensive traceability tools or even to undertake training courses on requirement engineering and traceability for their staff. Moreover, budgetary constraints can cause a lack of resource allocation for traceability activities or process definition. One of the problems that emerged during the survey was a lack of knowledge sharing or general agreement on the best approach for implementing traceability often leading to poor attitudes by certain team members on the importance of traceability.

MDE (Model Driven Engineering) is a new approach of software design where the whole process of design and implementation is worked out around models. With MDE, systems, processes and software engineering concepts are described as a set of models at different levels of abstraction. In a model, real-world objects are replaced by simpler objects, usually carrying the same names. Knowledge about a particular domain is structured according to modelling standards and their usefulness is best realised when they simplify complex software engineering problems. Furthermore, these models promote better communication for gaining agreement between different roles, to make predictions, to depict the results of empirical studies, to describe organisation structures, to describe system descriptions, to simulate code or to help make decisions on any domain.

Traceability has complex concepts that impact the entire software product development lifecycle. A common understanding and co-ordinated commitment must exist from early product inception right through the development stages if the practice is to yield the best results. This is no simple task. Building models assists in gaining agreement between all the success critical resources involved in the traceability process. The usefulness of these abstractions can only be fully realised by applying the models against real-world data.

In this chapter we propose a model-driven approach for describing traceability semantics using different layers of abstraction to gain agreement within a community on all matters related to the creation and manipulation of traceability data. We base our approach on the OMG's MDA specifications and we apply the models in Ericsson's OSS-RC telecom domain. We conclude with a discussion on the lessons learned and the challenges that were overcome in this endeavour.

### 7.1.1 What is a Semantic Model?

A model is an abstraction of phenomena in the real world. The study of semantics is the study of meaning, the meaning behind words. Semantic models allow users to work at a higher level of abstraction than the real world, abstracting the information and logic and adding meaning to concepts in a particular domain. A semantic model expresses meaning using a language such as UML or XML and attempts to define *data* from a user's perspective. The models show relationships between the different model elements. When agreement has been reached on the model elements the models can then be used to communicate the agreed semantics to a wider group of people, perhaps an entire organisation. The models provide an effective approach for presenting education material on a particular domain. In some cases the semantic models provide the foundation from which standards are built.

### 7.1.2 What is a Traceability Semantic Model?

In the context of this study we use a semantic model to add meaning to traceability concepts. In a general sense, we provide meaning to a number of core traceability concepts. The models provide a higher layer of abstraction for developing a consistent perspective, for example between the researcher and Ericsson. The model is layered and can be extended to create powerful, intuitive representations of the traceability domain. While the models were not used during the deployment of training it is our recommendation that in the future models similar to the ones produced by this study act as the foundation for courses on traceability.

The models also take into consideration many relevant research concepts. The semantic model we propose facilitates better communication among the success-critical stakeholders, promotes reusability of concepts and creates a consistent terminology of traceability concepts. We call the semantic model, the TRAceability Model or TRAM. The TRAM consists of a conceptual structure which acts as a schema for representing traceability, giving us a means to focus on structure and organize our perception of traceability.

The TRAM describes the *traceability data concepts* and the *relationship between the data*. For example, when talking about requirements do we refer to a product requirement or a project requirement? The answer to this question is likely to be "it depends." And that is the correct answer, because it does depend. It depends on who is asking and why. If a Product Manager is asked for a list of their requirements they will produce a list of the product requirements. If a project manager was asked the exact same question, they will produce a list of the project requirements. Are the two lists the same? It depends, on whether there is one project developing the entire product. In the case with Ericsson's, in particular the OSS-RC domain, the product requirement list would be much larger than the OSS-RC R5 project list. Something as deceptively simple as "Requirement" can cause much confusion. It is exactly this confusion that the semantic model strives to both reveal and resolve

At the higher levels of abstraction we attempt to make the semantic models independent of any particular domain or organisations. At the lower levels of abstraction we create semantic models of the traceability situation from the perspective of the OSS-RC project domain. At the topmost layer we strive to create a model with a goal of providing one unified perspective of the traceability data. All the lower levels provide models from a

number of different perspectives that can be understood in its individual context and its relation to other contexts.

In Figure 7-1 below, *TRAM Constructs*, we illustrate that TRAM consists of concepts (classes), relationships (properties of objects and the data), rules (axioms and constraints) and instances of concepts (objects consisting of data and facts)

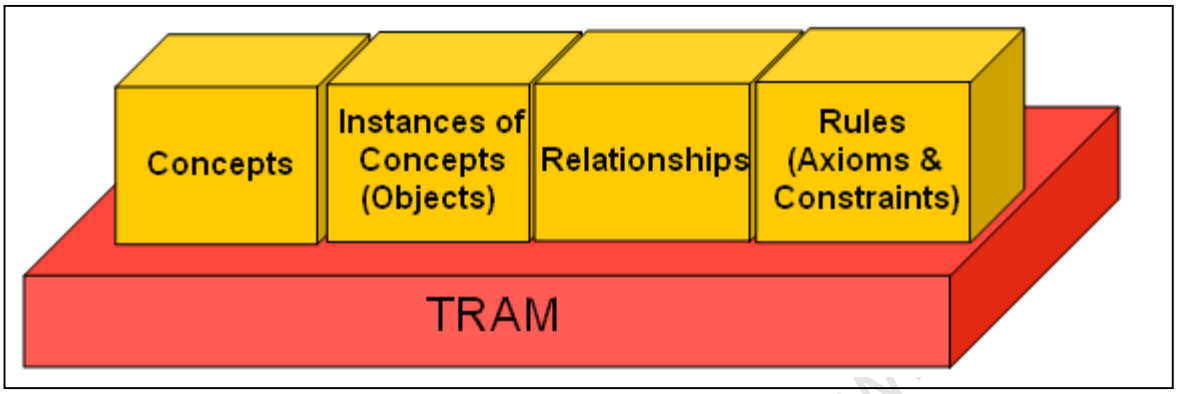



**Figure 7-1 TRAM Constructs**



### 7.1.3 The Elements Defined in the Semantic Model

The main elements that we describe in the semantic model are:

- *Traceability Item*: A traceability item is any item that impacts the quality of a software product in any way. Spence describes a traceability item as "Any textual or model item, which needs to be explicitly traced from another textual or model item, in order to keep track of the dependencies between them". (Spence and Probasco, 1998) A general definition of a traceability item would be a "requirement". Other examples of traceability items include textual descriptions, model elements, design elements, source code and test artefacts. A traceability item could also include items like assumptions, issues, change requests or impact analysis. Each item can be placed under configuration control.
- *Traceability Link:* A traceability link is a relationship between traceability items. Traceability link help stakeholders understand the many associations and dependencies that exist between software artifacts created during a software development project. (Sherba et al., 2003) The links are cross-references between the items connecting for example a requirement to a design element or to a source code element. Understanding how changes to a traceability item and the impact of changes to others is difficult unless there are links between the items. For example, in the Figure 7-2, *Requirement Associations*, we illustrate using UML that Requirement B is linked to Requirement A. Therefore if there is a change to Requirement A then Requirement B may be impacted by this change.

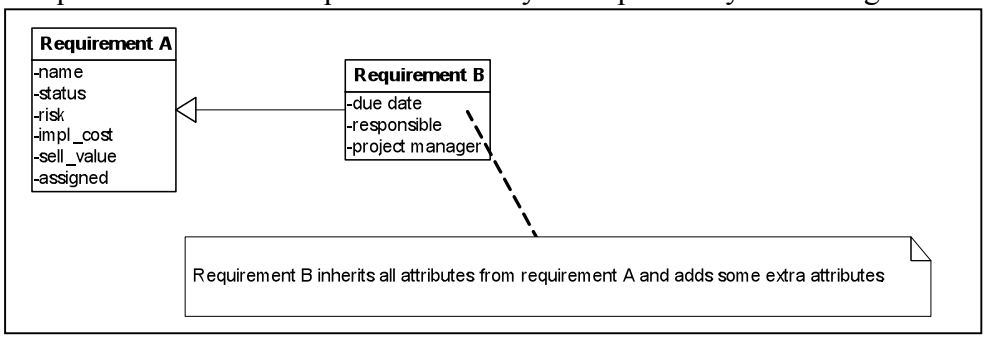



**Figure 7-2 Requirement Associations**

- *Traceability Types*: Programming languages have data types which is a name or label for a set of values and some operations which can be performed on that set of values. Similarly traceability items have a type which groups the items using a set of predefined values and primitive operations. For example a non-functional requirement is a traceability type category for requirements that describe non-functional qualities like performance, reliability, and design and implementation constraints. These types may act as statically or dynamically checked constraint on the programs that can be written in a given language.

## 7.2 RELATED RESEARCH

Letelier's paper on building a requirement traceability framework was a major input to this paper. He described the use of UML to establish a common traceability framework. He establishes a UML reference metamodel for requirements traceability, representing all the software development artifacts and traceability links among them. He further used the UML extension mechanisms, for adapting UML for every projects needs. (Letelier, 2002)In 2005 at a traceability workshop as part of the European Conference on Model Driven Architecture , Aizenbud-Reshef established an approach for defining operational semantics for traceability using UML. (Aizenbud-Reshef et al., 2005) Limon et al analyzed current traceability schemes, for instance link types, in order to obtain relevant features and identify overlaps and inconsistencies among the approaches. (Limón and Garbajosa, 2005)

Related research also considers new approaches for improving requirement modeling practices. Shrotri et al at the 2003 IEEE Software Engineering and Formal Methods conference proposed using UML object diagrams for specifying pre- and post-conditions for use cases. (Shrotri et al., 2003) At the 2005 ACM symposium on Software Visualization, Kholkar proposed to bridge the gap between the natural language used in use cases by extending the set of UML diagrams with three new diagrams that would enable rigorous specification, analysis and simulation of requirements.(Kholkar et al., 2005)

However as the Semantic Modelling of Model Driven Architectures workshop observes "semantic modelling in the context of model-driven development approaches is an area that is still in its infancy, but that has the potential to address recent issues in software engineering technology" (SMMDA, 2006)

### 7.2.1 What is a Metamodel/Model in Context of TRAM

The *TRAceability **Model** (TRAM) consists of metamodels and models which* specifying the key concepts and characteristics in the traceability domain in a standard implementation-independent way. As described in Chapter 6, a metamodel is a model that explains a set of related models. A metamodel is a precise definition of the constructs and rules needed for creating semantic model elements at a high level of abstraction. It serves as a template for a model or in other words a model is an instance of a metamodel. (Kelleher, 2006b) The model encapsulates a view of traceability at the application level or in our case the data at the OSS-RC projects. It is an abstraction of the metamodel, for a specific domain or purpose. With the formation of the OMG the exploitation of metamodel and model technologies has been revolutionized. The OMG defines UML metamodel profile specifications in many diverse fields from Quality of Service and Fault-Tolerance Characteristics and Performance Management.

### 7.2.2 What is a Profile?

*"Profiles are a UML extension mechanism. A profile applies to a language specification, specifying a new modeling language by adding new kinds of language elements or restricting the language"*

*-UML 2.0*

There are situations when the UML does not have all the constructs and elements for a particular domain or application. UML provides an extension mechanism, the *UML Profile* mechanism, to tailor the language to specific application areas. A profile is described in UML 2.0 as a stereotyped package that contains model elements that have been customized for a specific domain or purpose using extension mechanisms, such as stereotypes, tagged definitions and constraints. In principle, profiles merely refine the standard semantics of UML by adding further constraints and interpretations that capture domain-specific semantics. They do not add any new fundamental concepts. If a concept is not present in the UML, you may describe it by defining a stereotype. Profiles represent an agreement within a community from which practitioners can then draw the particular model of any educational situation they want to describe. In this study we create a profile which represents an agreement between the researcher and Ericsson. In Figure 7-3 below, *UML Profile*, we illustrate that a profile is an extension mechanism to a UML metamodel.

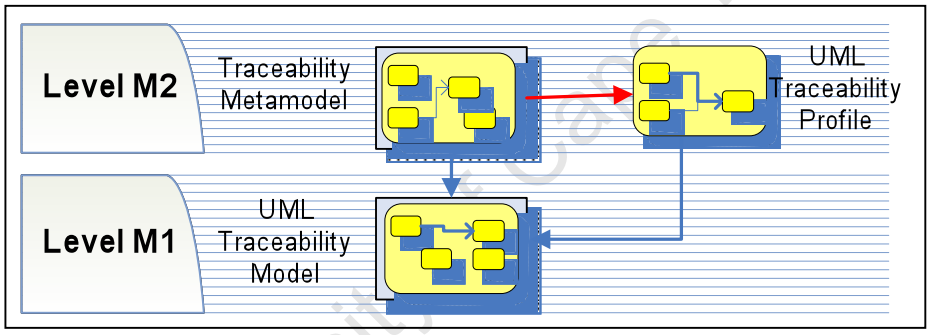



Figure 7-3: UML Profile

### 7.2.3 What is the Difference between TRAP and TRAM?

Semantic data elements are deceptively similar to the entities and attributes we defined in the TRAceability Process (TRAP) model. As illustrated in Figure 7-4 below, *Tram Models the "What"*, the TRAM defines the traceability data while the TRAP describes how and who processes that data. *The TRAM models represent meanings and associations of all aspects related to the traceability data, while the TRAP represents the human factor involved in the implementation.* In simple terms the TRAM defines the "what" and the TRAP describes the "how" and the "who".

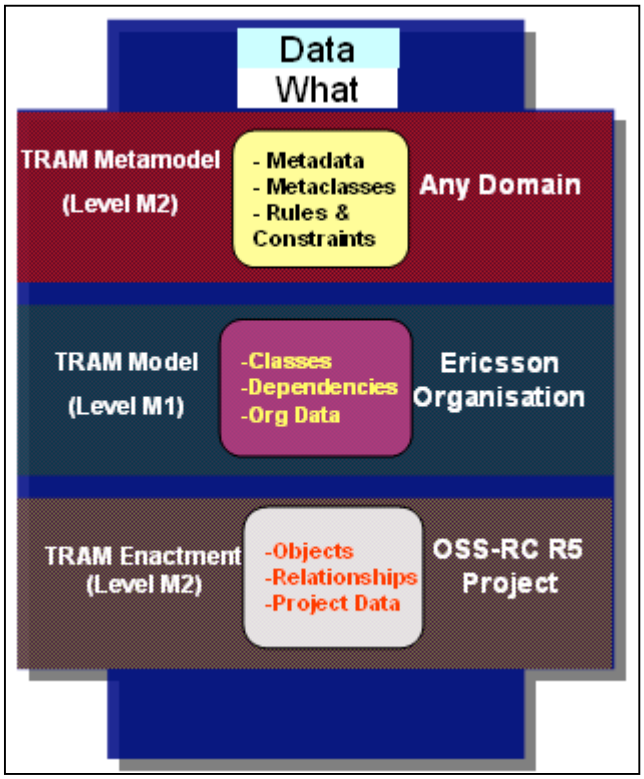



Figure 7-4: TRAM models the "What"

Some would argue that process elements and data elements should be incorporated into the same model. A simple question that helped us decide to create two separate models was: What is the difference between the process and the tool? The answer lies in the fact that the semantic model or TRAM would act as a blueprint of the logic for a tool, while the TRAP model describes how the tool interfaces with the development organisation. Another way of looking at it is TRAP describes the human factor while the TRAM model describes the traceability data that the humans manipulate. TRAP has process elements like, Lifecycle, Phases, Roles and Iterations, while TRAM has data elements like the Traceability Items, the Types, their Relationships, and the Tool. In the early stages of this project we combined the process and the data into one model. When demonstrating the models to different users we observed that the models were in fact overly complex leading to difficulties with communicating even the simplest concepts.

Furthermore, the inputs to the TRAM and TRAP came from different project members. For example, the OSS-RC process team had the biggest input into the TRAP models while the OSS-RC Requirement Manager and the Project Managers had the most input into the TRAM model. Moreover, if the models were used for training purposes the TRAP would be an integral part of a process course, while the TRAM would be used predominately in a Requirement Engineering course. Finally, updating and maintaining models is an important part of any modelling effort. In our experience it was the OSS-RC process team who maintain and update TRAP while it was the Requirement Manager who took ownership of the TRAM models.

In Figure 7-5 below, *TRAM and TRAP utilise the same 4 Layered Architecture*, illustrates that they both utilise the Four Layered Architecture recommended by the OMG. Layer M3 Meta-Object Facility (MOF), Level M2 Traceability Metamodel, Level M1 Traceability Model, and Level M0 Traceability as enacted in a real life application. It is the meta-metamodel (called Layer M-3 models) that is the foundation for defining any modelling language, and is typically more compact than the metamodel that it describes. The meta-metamodel layer forms the infrastructure for the metamodeling layer while the metamodel layer forms the infrastructure for the model layer and so on. The responsibility of the MOF is to define the language for specifying a metamodel. At Layer M2 we define a traceability metamodel which is a precise definition of the constructs and rules needed for creating a semantic model of the traceability domain. Level M1 consists of UML models which represents the traceability data at the Ericsson's organisation level. While, the Layer M0 consists of object or project instantiations, which in this study is the OSS-RC R5 project.

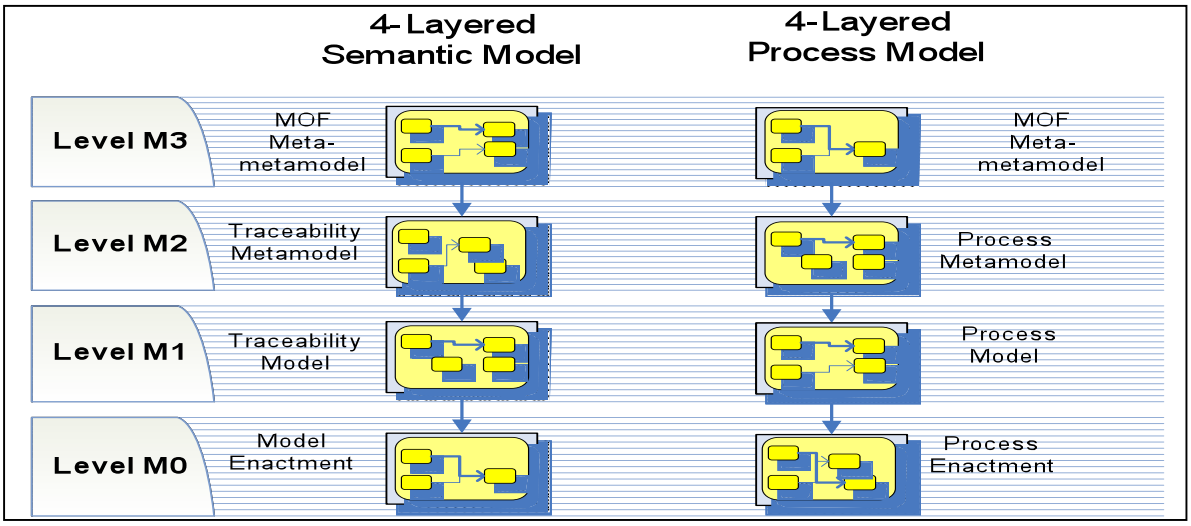



**Figure 7-5: TRAM and TRAP utilise the same 4 Layered Architecture**

In data terms, at the lowest layer we have objects and their relationships. At the middle layer we have object types and relationship types. The highest layer is the meta-object types and meta-relationship types.

At the lowest layer a physical traceability system provides a list of traceability items, their storage, configuration management, and all the functionality that can be implemented on the traceability items.

At the middle layer the traceability model offers a set of modelling concepts for designing a conceptual traceability suite. This includes the schema constructs for defining the properties of the object types and relationship types for particular applications. Different models offer a variety of modelling concepts. Thus, this set of modelling concepts characterises the semantics of a specific data model.

## 7.3 MOTIVATION

The Center of Excellence for Traceability technical report helps motivate the TRAM by describing the following problems and challenges that need to be overcome in the field of traceability:(Hayes et al., 2006)

> "Tracing requires communication between stakeholders, but semantic mismatches and disparate use of terminology across various stakeholder groups create communication barriers"

The Center of Excellence for Traceability describes the following challenges that exist in the traceability domain:

- *Challenge:* Create a body of knowledge that reflects best practices of traceability experts and practitioners, standard terminology, and additional information such as case studies on traceability.
- *Challenge:* Develop effective educational components on the practice of traceability that can be integrated into university, industrial, or certification curriculum.

"For traceability links to be useful, they must reflect current dependencies between artifacts. Traceability links need to synchronously evolve with their related artifacts, however, current change management systems and link semantics are not sufficiently sophisticated to support effective evolution of traceability links"

- *Challenge:* Develop change management systems that effectively support the evolution of traceability links across multiple artifact types.

- *Challenge:* Develop techniques for reusing traceability work products.

- *Challenge:* Develop techniques for maximizing reuse of traceability links when existing code is reused in a new product.

"In order to effectively utilize links and understand the underlying traceability relationships, it is necessary to define the semantics (e.g., type) of a link, however defining a formalism to represent the semantics is a non-trivial task and may be domain-specific"

- *Challenge:* Define a meta-model to represent semantic information of traceability links and provide examples of instantiation to specific domains.

- *Challenge:* Establish a communication mechanism to make the traceability community aware of standards related to traceability.

### 7.3.1 Objectives of TRAM

Modelling a semantic model process supports the goal of improving traceability by providing a mechanism that:

- Builds a framework that satisfies some of the problems identified during the case study and survey.

- Defines and explains the key concepts and relationships required for the traceability domain as well as their relationship.

- That serves as a logical architecture which can be consumed or used by a traceability tool.

- By recording reasoning on the traceability data elements.

- By communicating and promulgating all aspects involved in traceability both to the user community and the academicians.

At a more detailed level, we have identified five primary objectives for the development of the TRAM:

1. Facilitate our understanding of traceability by abstracting from the real world at one level and applying the abstractions at lower levels.

2. Enable effective communications regarding traceability data.

3. Facilitate reuse of traceability data structures.

4. Support evolution of the models for future research and developments by the user community.

5. Provide a framework for *capturing, analyzing*, *assessing and improving* the implementation of traceability that can be used large and small organisations.

As Figure 7-6, *TRAP and TRAM mapped to Empirical Data*, below illustrates, while TRAM and TRAP address similar problems that we discovered during the case study they also address some different issues. Undoubtedly, the biggest issue that TRAM addresses is the lack of a common terminology for traceability concepts.

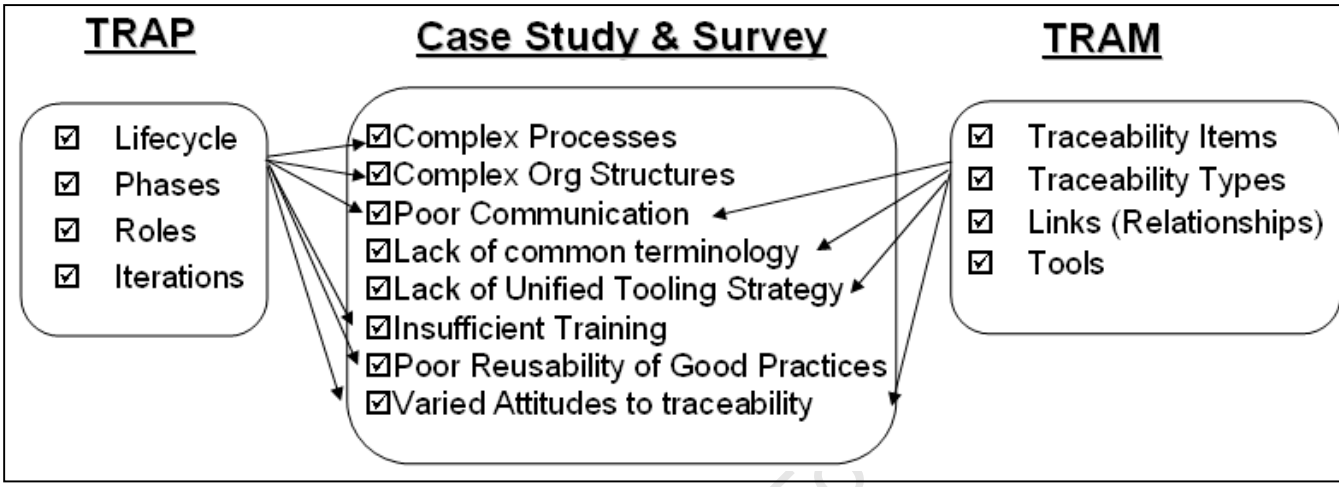



**Figure 7-6 TRAP and TRAM mapped to Empirical Problems**



## 7.4 OUR APPROACH

Achievement of the above objectives can make possible using a good methodology.

### 7.4.1 The Research Inputs & Method

The inputs to our process framework are shown in Figure 6 below, Inputs to TRAM, include:

1) Software modelling standards, best practices, like OMG's MDA, MOF and UML.

2) Research efforts (literature, research projects, empirical studies) by academia and software research groups;

3) Industry knowledge and modelling expertise

4) Empirical data gathered from the case study and survey.

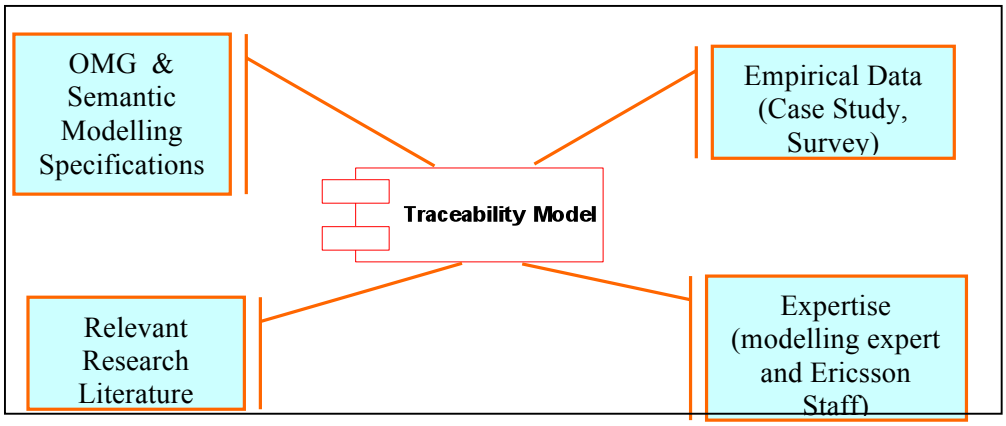



**Figure 7-7: Inputs to TRAM**

Table 7-1 summarizes the various steps we followed in the creation of the Traceability Model.

| Stage | Activity | Data Source | Outputs |
|---|---|---|---|
| TRAM Prototype (2005) | Develop M2 Metamodel | Current Literature (SPEM, IEEE, MOF etc. | Metamodel Layer M2 |
| Desk-check and model check (lab trials) | Complete model checking activity with process modeling expert. Model walkthroughs, scenario tests etc. | Expert Modeler | M2 Process Metamodel assessed by process modeling expert |
| Prototype introduction (2005) | A series of workshops carried out with the OSS-RC Requirement Management Team | Input & Agreement from participants. | M2 metamodel refined for OSS-RC domain. |
| Development Unit Layer (M1) (2006) | During participation workshops, we designed and analysed the models. We assessed the models against the objectives particularly against the criteria for making improvements to traceability | Participation input from OSS-RC(RM, CM, Project Mgt Assessment by OSS-RC modeling expert. | M1 Layer Models |
| Project Layer or Enactment Layer (M0) 2006-2007 | Models Instantiated | Project representatives Project Manager, RM, CM | M0 Layer Models |
| Assessment 2007 | Analyse and assess modelling effort | Modelling expert, Requirement Manager, Project Manager OSS-RC | Assessment |

**Table 7-1 TRAM Method**

In Chapter 11, the validation chapter we describe the validation approach and the results.

In order to ease the readability of the remaining sections, Table 7-2, *Model Overview*, we briefly describe the upcoming sections for the reader.

| (Sub) Section | Brief Description |
|---|---|
| Section 7 | Layer M2 Metamodel |
| 7.5.1 | Design considerations and inputs from MoF. |
| 7.5.2 | Packages are a way of grouping model elements together. We introduce the package concept and illustrate the package structure |
| 7.5.3 | The M2 Metamodel and all the components. After illustrating the M2 metamodel we describe each element and the attributes and operations where appropriate. |
| Section 7 | Layer M1 Model (Ericsson) |
| 7.6.1 | Layer M1: Traceability Item. Model created with the Ericsson organisation wide requirement engineer. This model describes all the elements involved in the creation of a traceability item. |
| 7.6.2 | Layer M1: Configuration Control. Model created with the Ericsson organisation OSS Configuration Manager. |
| 7.6.3 | Layer M2: Product Management. Model created with the Ericsson Product Management team. |
| 7.6.4 | Layer M1: Report. Model created with the Methods and Tools department describing a report generation model |
| Section 8 | Layer M0. A few simple examples documented from the OSS-RC R5 project. |

**Table 7-2: Model Overview**



## 7.5 LAYER M2 METAMODEL

### 7.5.1 Design Considerations

The three main metadata modelling constructs provided by the MOF are the Class, Association, and Package. These are similar to their counterparts in UML, with some simplifications. Classes can have Attributes and Operations at both “object” and “class” level. Attributes have the obvious usage; that is, representation of metadata. Operations are provided to support metamodel specific functions on the metadata. Classes may inherit

from other Classes. Associations support binary links between Class "instances." Each Association has two Association Ends that may specify "ordering" or "aggregation" semantics, and structural constraints on cardinality or uniqueness. When a Class is the type of an Association End, the Class may contain a Reference that allows navigability of the association's links from a Class "instance." Packages are collections of related Classes and Associations. Packages can be composed by importing other Packages or by inheriting from them. Packages can also be nested, though this provides a form of information hiding rather than reuse

Objects correspond with practical elements such as traceability items, relationships and so on but also with arbitrary compositions of interactive elements. From our early efforts we discovered that we can generate arbitrarily complex models which do not simplify traceability but rather make the concepts more complex. Therefore in the models we suppress the Attributes and Operations where possible.

User manipulations of objects are modelled as editing operations on the state of these objects. This provides us with an object style semantics, in which user actions update the state of an existing object rather than creating a new object (a functional style semantics). User actions create and interconnect objects.

### 7.5.2 TRAM Package Structure

The UML 2.0 Infrastructure defines the foundational language constructs required for UML 2.0. It is complemented by UML 2.0 Superstructure, which defines the user level constructs required for UML 2.0.

A package is used to group elements, and provides a namespace for the grouped elements. A package is a namespace for its members, and may contain other packages. Only packageable elements can be owned members of a package. By virtue of being a namespace, a package can import either individual members of other packages, or all the members of other packages.

The basic units of compliance for UML are the packages which define the UML metamodel. All metamodels that reuse the Infrastructure Library should clearly specify which packages they reuse, and further clarify which packages are imported without change, and which packages are imported and extended via specialization.

In the UML metamodel, Package is a subclass of Namespace. A Package contains Model Elements such as Packages and Classifiers. A Package may also contain Constraints and Dependencies between Model Elements of the Package. The purpose of the package construct is to provide a general grouping mechanism. In fact, its only semantics is to define a namespace for its contents. The package construct can be used for organizing elements for any purpose; the criteria to use for grouping elements together into one package are not defined. A package owns a set of model elements, with the implication that if the package is removed from the model, so are the elements owned by the package. Elements with names, such as classifiers, that are owned by the same package must have unique names within the package, although elements in different packages may have the same name.

In UML the Core package is the highest level package not depending on any other package. As shown in the Figure 7-8 below, *TRAM extends the UML Core package*, we illustrate that before starting modelling our process we must extend the *UML Core*

*Package*. The Core package provides basic constructs for creating and describing meta-model classes.

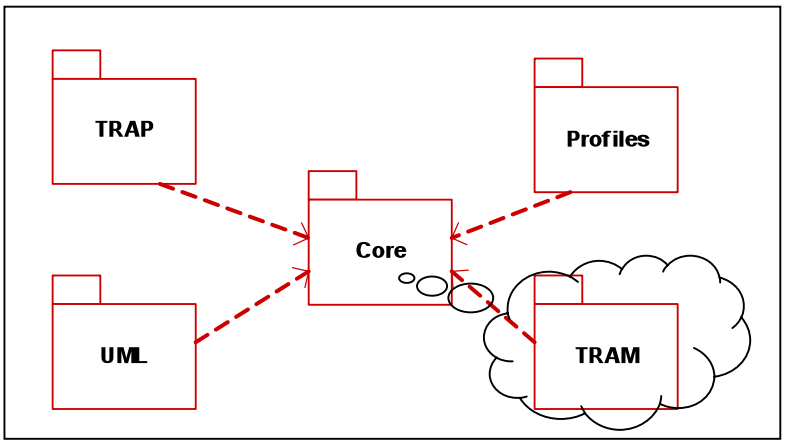



**Figure 7-8: TRAM extends the UML Core Package**



## 7.5.3 Layer M2 Profile

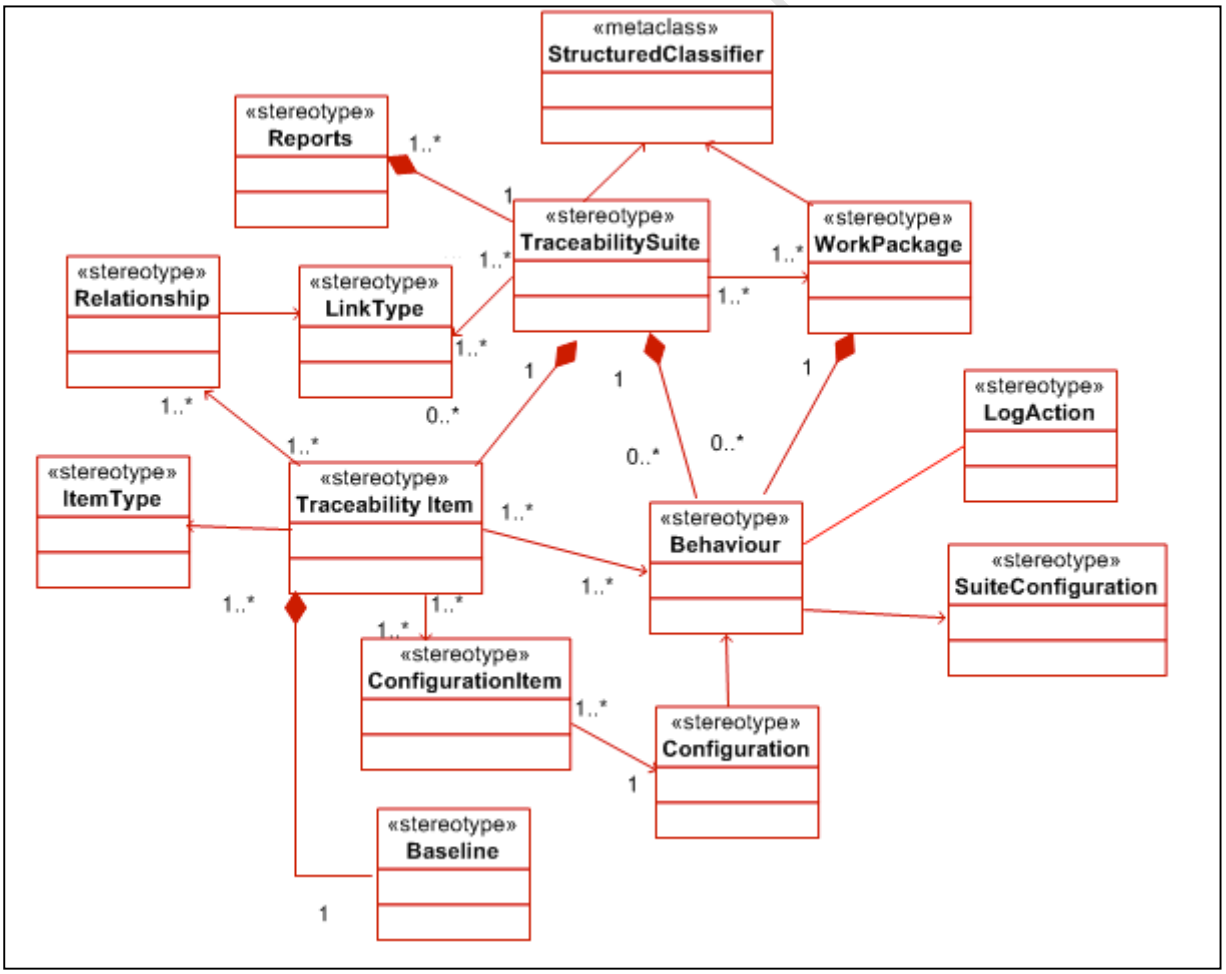



Figure 7-9 TRAM Level M2

The INCOSE (International Council on Systems Engineers) carried out a study based on the system environment, user interfaces, support and maintenance, and capabilities for managing a traceability system. (Website, 2001) They identified that the key capabilities of a traceability systems as predefined and customizable data types (traceability types, relationship types), predefined and customisable reports for the retrieval and grouping of the traceability items, configuration management, customisable change tracking functionality, trace analysis capabilities for identifying inconsistencies, impact analysis, status accounting, integration with other software engineering tools and an environment that supports good communication to all stakeholders in the software development lifecycle.

The set of concepts (in addition to the UML 2.0 structural concepts) to specify the semantic aspects of a traceability is shown in Figure 7-9, *TRAM Level M2* above. We now describe each model element and where they add further clarity to each element, the attributes and operations.

| **Traceability Suite** |
|---|
| **Description** |
| A Structured Classifier (extends Classifier) is an abstract metaclass that represents any classifier whose behaviour can be fully or partly described by the collaboration of owned or referenced instances. The traceability suite is a structured classifier acting as a grouping mechanism for a set of traceability items. The traceability suite has a predefined interface that is realized by all the components in the traceability system and the run-time system.<br><br>A traceability suite has the functionality to store items in the traceability systems so that algorithms can be used for traversing and interrogating theses items. The Traceability Suite contains a set of zero or more traceability items. A traceability item is a common name for any object that may be included in a traceability suite. The traceability items are created, modified, deleted within the suite while the types (functional, non-functional) are initialized by the itemType element. The traceability suite utilizes the behavior to control the handling of the traceability items. For example, the importing, exporting, copying, allocating and decomposing of the traceability items. |

| **Suite Configure** | |
|---|---|
| **Description** | **Operations** |
| The Suite Configure component is instantiated to set up the profiles of the users of the traceability suite, to create the traceability matrixes, to create the traceability projects, to create the versions, to define the tags, and the unique identifiers for each item. Each user can have different authorities in their usage of the traceability suite. For example some users may have only read access while other users may have read/write access.<br><br>The function of suite configure is to organize the traceability items into categories and subcategories. The | suiteConfigure(),<br>createProject(),<br>createUser(),<br>createVersion(),<br>createStucture(),<br>createTages(),<br>createTraceID(),<br>createAthority() |

| | |
|---|---|
| version is the state of the item, project, product or any configurable item that varies from its previous state or condition. The version of the traceability suite changes with each modification. | |

| Behaviour |
|---|
| **Description** |
| Behaviour includes control flow, data flow and state machines. The *Control flow* emphasizes the sequence of steps by requiring one step to finish before another starts. For example, a person must log in to the system before they can run reports. *Data flow* emphasizes calculation of inputs for steps by requiring that they explicitly provided by outputs from other steps. This is important when creating traceability rules on the flow of the data around the system. For example, rules for setting up data structures like a traceability matrix. *State machines* emphasize response to external stimuli by requiring that each step begins only when certain events happen in the environment of the system, with step inputs provided by the events. For example, the system must be in "state=log-on", and "change-request= accepted" by the Change Control Board before the Systems Engineer, can carry out Impact Analysis. The UML behaviour models start with state machines and extend them to cover control flow and data/object flow. |

| Work Package |
|---|
| **Description** |
| This is a generic term for a logical grouping of the traceability items that can be assigned to a specific project for development. Work packages are also called Features. Work packages have identifiable characteristics of a product that are distinguishable from other parts of the product. |

| Traceability Items | |
|---|---|
| **Description** | **Attributes & Operations** |
| A traceability item is any identifiable object that contains success critical information in the development of a software product. Traceability items include requirements, textual descriptions, model elements, design elements, source code and test artefacts. They also include items like assumptions, issues, change requests, glossary or term items. Each item must be placed under configuration control. | **Example Attributes:** "itemAttribute: string", "name: string", "owner: string" 'priority: string", "cost: long", "iteration: long"<br><br>**Example Operations:** It contains the operation assignAttribute(), which sets the attribute value of the traceability item. The changeAttribute() operation makes changes to the attribute value. The version() method sets the version of the attribute, for version control and change management to track changes made to each attribute. |

| Item Types | |
|---|---|
| **Description** | **Attributes & Operations** |
| Traceability Item Types are classifications of the items at different levels of detail or abstraction with the most common being "stakeholder requests," "features," "use cases," and "supplementary requirements" Whenever you document a traceability item it must belong to a specific type. | The setItemType() method sets the item type. |

| Traceability Relationship | |
|---|---|
| **Description** | **Attributes & Operations** |
| Traceability relationships help stakeholders understand the many associations and dependencies that exist between software artifacts created during a software development project. "A relationship is a semantic association between artifacts, portions of artifacts, or relationships" (Sherba et al., 2003)<br><br>There are implicit and explicit traceability relationships. Individual relationships between specific items can be useful in understanding portions of the system. However, the implicit relationships greatly outnumber the explicit relationships that exist between items. A traceability suite should be able to derive implicit relationships from the explicit relationships already represented in the system. The user should then be allowed to make these newly discovered relationships explicit. In this way, a synergy is formed. (Spence and Probasco, 1998) | ▪ Implicit().The operation executes a set of rules for identifying implicit traceability. For example it executes a check for relationships between the model elements or identifies dependencies between elements with the same naming convention.<br>▪ Explicit() This operation executes a set of rules that describes all the explicit (or manual) traceablity relationships. |

| Link Type | |
|---|---|
| **Description** | **Attributes & Operations** |
| The LinkType element captures the type for the traceability link between the items. | **Attributes**<br><br>Jarke defines traceability link types as process or product. (Jarke, 1998) We define two attributes and operations Product and Process for this purpose. Letelier further defines a number |

| | |
|---|---|
| | of link types; traceTo, modifies, responsibleOf, rationaleOf, validatedBy, verifiedBy and assignedTo for establishing links between the Stakeholder and the TraceableSpecification. (Letelier, 2002)<br><br>**Operations**<br><br>▪ product()Sets the trace type as product related.<br>▪ process() Sets the trace type as process related<br>▪ traceType().We propose that the TraceType has operations to overcome this problem. Limon et al stated that a link type will have a type and will be related either to a software/system development product or development process.<br>▪ setLinkType() Sets the link type. For example a link type on issues.<br>getLinkType() Returns the type of link defined. |

| **Log Action** |
|---|
| **Description** |
| The LogAction is used to log entities during any execution involving the traceability items and their associated dependencies. The logged entities can be simple strings, traceability matrixes, instance values or any other entity of interest. The target of a log action refers to a logging mechanism in the run-time system. The request refers to the information that is logged. The representation of the logging mechanism is not specified. The LogAction records a snapshot of the traceability items. |

| **Baseline** | |
|---|---|
| **Description** | **Operations** |
| A Baseline is a version of a configuration which is established at an agreed point in time after which only controlled changes are allowed. Baseline Management, is performed when deciding what a new configuration is to be used for a specific purpose and consists of the activities; Configuring, Change Management, Baseline Review, Baseline Establishment. Configuration Control concerns the activity of controlling changes *after* a Baseline has been established.<br><br>For example, for one product version, the Change Control Board is formed in time for selection of the first traceability items, and controls those items throughout the life cycle of that product version. Change Management is the activity to change the Baseline without formal | **Operations:**<br><br>baseline(): executes an algorithm that gathers all the traceability items that belong to the baseline. |

| | |
|---|---|
| control. When the Configuration Manager has created the baseline and connected traceability items to it, the baseline will be put under informal control, called Change Management. You may, for example, go back to a previous revision of a Baseline and check the configuration in that certain revision of the Baseline. | |

| Configuration |
|---|
| **Description** |
| Traceability items change and Configuration Management (CM) is the discipline of identifying and controlling these changes during the development of a product. Configuration Control is the activity where the Configuration Manager (CM) control changes made to a new or modified revision of a traceability item or a Configuration that has been added to baseline. After a configuration has been baselined, changes are not only managed, they are also controlled via a formal decision procedure. For example, the level of control has been raised to project level. Configuration management is about controlling changes after establishment of a Baseline. |



| Configuration Item | |
|---|---|
| **Description** | **Attributes & Operations** |
| CM controls the changes to a products definition. Any traceability item that is out under CM control is a Configuration Item.(CI) | Attributes: “cmName: string”, “baseline: long” (which baseline a CI is attached to). |

| Report | |
|---|---|
| **Description** | **Operations** |
| A query service allows altering of relationships so that different views of the information space can be created based on the needs of various users.<br><br>Many reports can be generated but we will discuss three namely; Traceability Item report, Traceability Report, and Suspect Requirement Trace. The Traceability Item | ▪ create() Creates a report. The createReport() contains a specific query and a table and will be executed each time the report is opened.<br>▪ view() We can view a report<br>▪ compare() Compares two reports<br>▪ history(): Gives us the history of the reports executed. For example we can go back to early versions of reports and compare our results. |

| | |
|---|---|
| report shows the marked-up requirements and their attributes. The Traceability Report shows the link relationships between requirements. The suspect requirement trace report shows the potential impact of a requirement change to other requirements that it is linked to.<br><br>A report is a function that enables the user to run reports, queries or traceability metrics. The Dynamic Report contains a specific query and a table and will be executed each time the report is opened. Discrepancies are noted and missing items are accounted for in plans to correct the oversights. | ▪ Impact Analysis() The impact analysis identifies the impact of change on an item to its related item. The process of understanding the complete effect of a particular change. |

## 7.6 LAYER M1: MODELS LAYER

While there is much discussion and literature on creating Layer M2 Models, one of the noticeable problems that we encountered was the lack of literature describing the creation of the M1 and M0 Layer models. In the following section we depict the M1 models that were created in accordance with the OSS-RC organisation in Ericsson. Each model was created with the guidance and support of a number of different roles, namely; the OSS-RC Requirement Manager, the Configuration Manager and the Project Manager. The diagrams are represented as classes and depict real-world scenarios within Ericsson. We do not describe the attributes and operations for each model in order to keep the models simple and easy to understand.

### 7.6.1 Layer M1 Model: Traceability Items

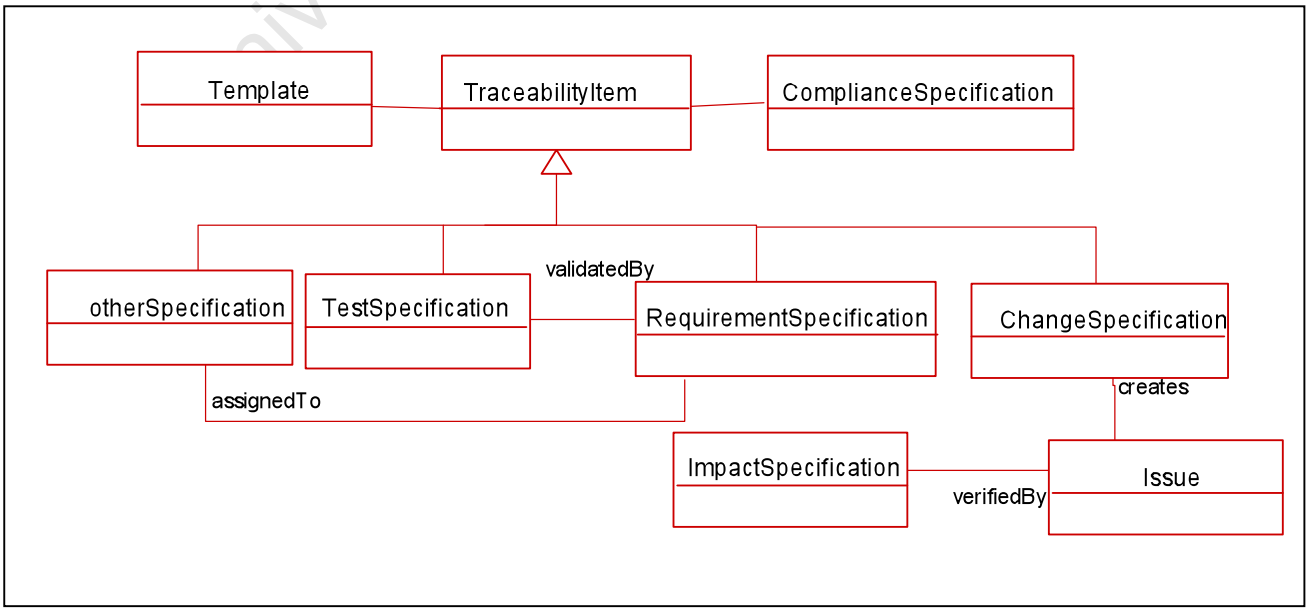



**Figure 7-10 Traceability Item**

In Figure 7-10 above, *Traceability Item*, we illustrate a Layer M1 model that was created with the OSS-RC R5 Requirement Manager which describes the creation of the traceability items. A traceability item is any specification that can impact the system to be developed, for example a model (otherSpecification in figure above), a diagram (otherSpecification), a use case (otherSpecification), a non-functional requirement (RequirementSpecification), a change request (ChangeSpecification), a test specification (TestSpecification), or any other specification in the development cycle that impacts the overall system. Traceability items are created in documents using *Templates*. Configuration management is the discipline of identifying the components of an evolving system for the purpose of controlling changes to these components. A *ChangeSpecification* is a traceability item specifying a proposed change to any traceability item which is also under configuration control. A Change Specification can be caused by an *Issue.* When a Change Request is received *Impact Analysis* investigation is undertaken. An *Impact Specification* contains the Impact Analysis data.

### 7.6.2 Layer M1: Change Control

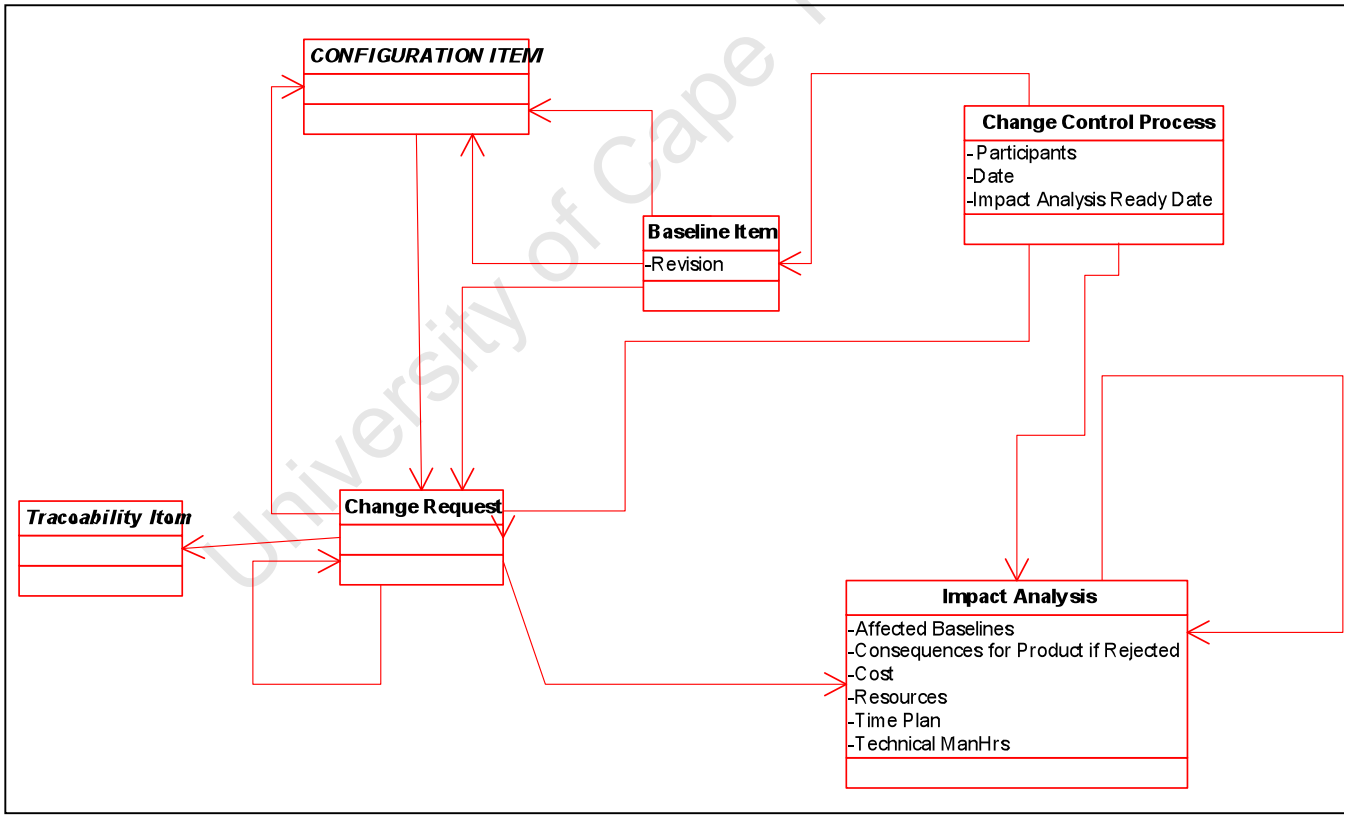





**Figure 7-11: Change Control**

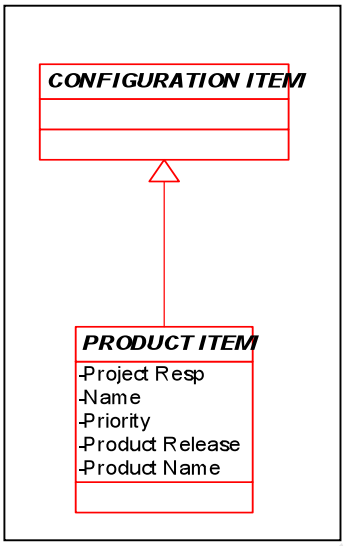
In Figure 7-11, above, *Change Control*, we see that a Change Requests (CR's)is created by any resource who has the authority to make a change the traceability items. CR handling is used to both control project scope and to move traceability items between project phases. Change requests are also called Configuration Items. Change Requests

(CRs) can be written on any traceability item in an approved baseline or directly on a baseline.

When the CM Manager has received the CR she/he has the possibility to send a Request for Impact Analysis of the CR. The request can be sent to one or several groups. This activity automatically changes the CR state to INVESTIGATION. A CR Author may also perform an Impact Analysis before the CR is sent to the Configuration Manager. An Impact Analysis is undertaken usually by the Systems Engineers. During the analysis they describe the Cost, the Technical Man Hours required, the number of Resources needed, a Time Plan and the Consequences if the change is not taken into account.

Configuration control concerns the activity of controlling changes after a *Baseline* has been established. A Baseline is a version of a configuration established at a point in time when only controlled changes are allowed.

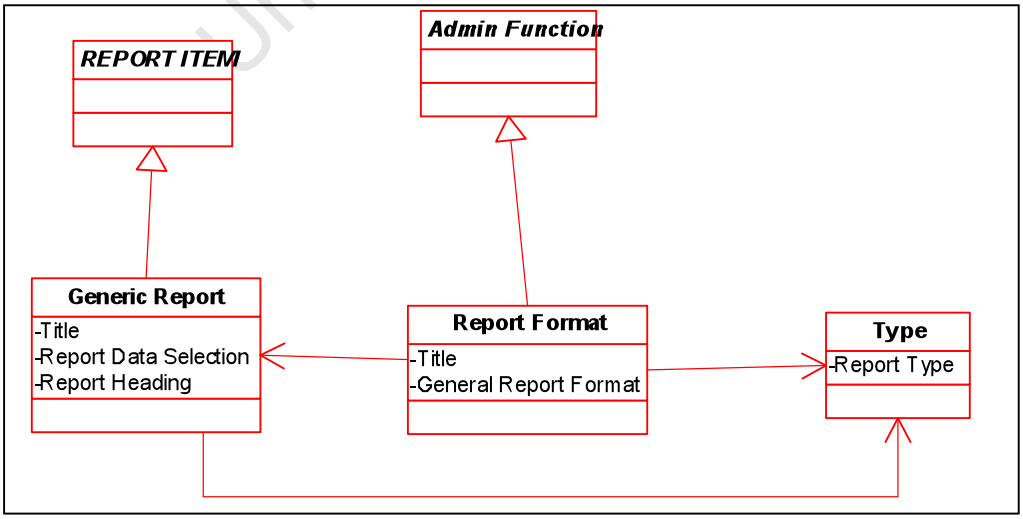
### 7.6.3 Level M1 Product Requirement




Each Product Requirement must have the attributes describing the Project Responsible for the Product Requirement, the name of the requirement, the priority (High, Medium, Low), which Product Release (OSS-RC R5.2) and what is the name of the Product (OSS-RC)



### 7.6.4 Layer M1: Report Generator




Working with the OSS-RC, Methods & Tools we created the above M1 model to describe the creation of reports. In the MAR's tool there is an Administration Function,

which generates the different reports. Each report has a pre-defined format and a type (for example Compliance Report) which is defined in the Generic Report class. Each report creates a Report Item. For example one of the reports that the methods and tools team were generating was a Traceability Item Compliance Report. (see Figure 7-12, *A Report from MAR's Traceability Tool*) When a Traceability Item has been designed and tested the tester than should add whether the item is compliant with the original product requirement. Traceability item compliance or regulatory compliance refers that a traceability item is in a state of being in accordance with regulations, established guidelines, specifications or legislation. For example a traceability item can be in the following states:

- **Rejected:** The item is relevant for the receiving organization but it is rejected from the receiving organization in decision forum, for example a product steering group or change control board. The item is excluded from the baseline in the receiving domain. A reference to a formal decision forum should be stated. This is a valid end state.
- **Not Compliant:** The item is relevant for the receiving organization but it will not be met in the current project. The item is excluded or shall be removed from the current project's baseline in the receiving domain. Reasons or motivation for none compliance shall be stated. This is a valid end state.
- **Partly Compliant:** The requirement is only partly fulfilled. Reasons or motivation for partly compliance shall be stated. This is a valid end state.
- **Compliant:** The requirement is relevant for the receiving organization and will be met in the current project. The requirement is included in the current baseline in the receiving domain. This is a valid end state

Search Results - found 231 objects (limit set to Unlimited)
(Report Name = "Valadilene CR report")

| View | CR Title | Status | Prepared By | Owner | Connected IA | Connected CI | New |
|---|---|---|---|---|---|---|---|
| 1/109 18-FCP 126 770 [PA1] | henke | INVESTIGATION | Henrik Blomberg | hb | Impact Analysis, 1/1597-FCP 126 770-1, PA1 Impact Analysis, 1/1597-FCP 126 770-2, PA1 | | |
| 2/109 18-FCP 126 770 [PA1] | Verification R3A | PREL | Lars Ohman | edtloeh | | | |
| 3/109 18-FCP 126 770 [PA1] | Test01 | PREL | Lennart Larsson | uablarl | | | |
| 4/109 18-FCP 126 770 [A] | Stens Test #1 | APPR | Sten Pedersen | uablarl | Impact Analysis, 4/1597-FCP 126 770-1, PA1 | | |

**Figure 7-12: A Report from MAR's**

## 7.7 LAYER M0 MODEL INSTANTIATION (OSS-RC R5)

While there are many examples of traceability that can be documented in the OSS-RC R5 project we provide two simple examples in this section.

### 7.7.1 Requirement Management

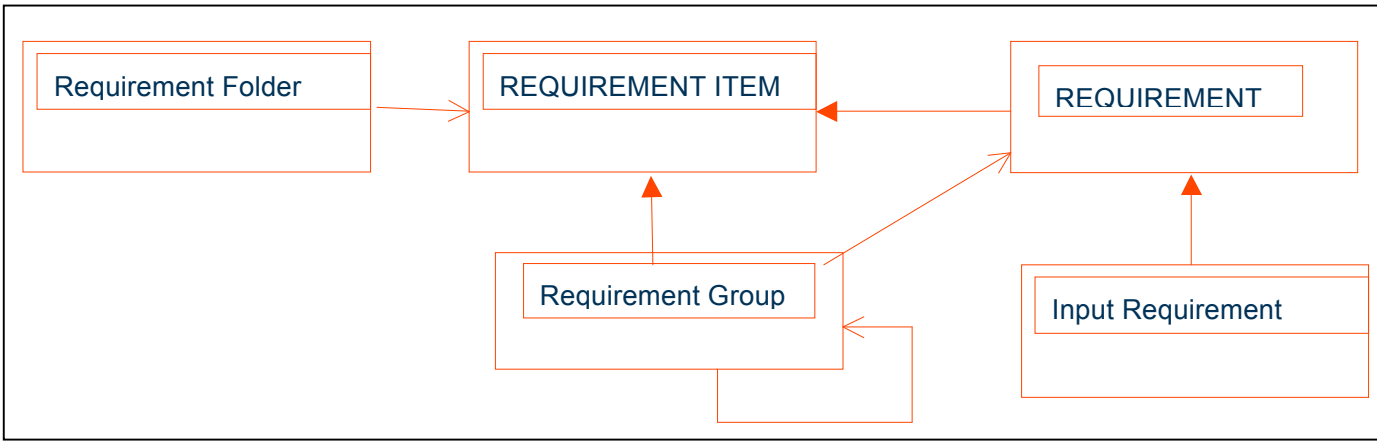



**Figure 7-13: Requirement Management Workflows OSS-RC R5**

In Figure 12, Requirement Management Workflow OSS-RC R5, we illustrate the main objects involved when creating a requirement. The Requirement Folder object contains all the Requirement Items in the OSS RC R5 project. A Requirement Item can be a part of a specific Requirement group. Input requirements are requirements that come from product management and there are detailed requirements which originate from the project team. The traceability flow is described as:

*=>Requirement Folder*

*=> Requirement Group (Main Requirement Specification/Requirement Specification)*

*=> Requirement Group*

*=> Requirement Item*

*=> Requirement (Input)*

In Figure 7-14, *OSS-RC R5 Requirement Flow*, we illustrate that the Input Requirements originate from the stakeholders UTRAN/BSS who produce a Main Requirement Specification document. This artifacts acts as an Input Requirement, which is divided into Detailed Requirements at the System Level and at the Node Level.

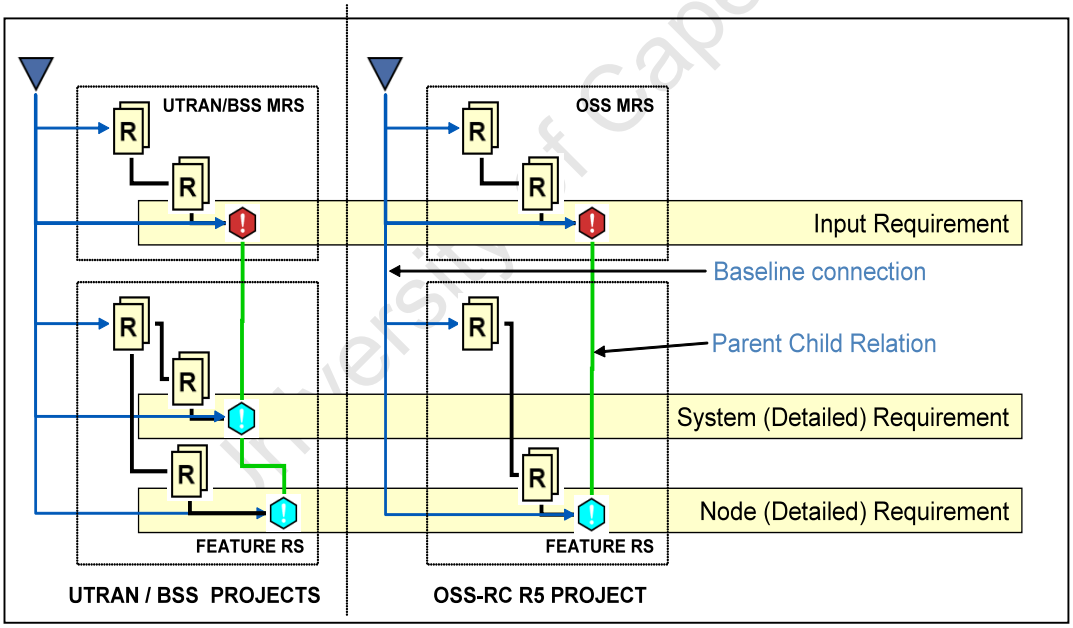



Figure 7-14: RM Plan OSS-RC R5 Requirement Flow

## 7.8 LESSONS LEARNED

Our initial attempts at modelling TRAM were disappointing. In our first attempt we integrated process and data models into the same model. The problem was that the models were too complex and not easily understood by many of the roles that provided input or we needed agreement from. While complete and fully functional, we found in practice that they contained too much information, especially if we included all the attributes and operations for each model element. We therefore recommend that in future research projects that model based approaches suppress the attributes and operations. This is

especially true if you are trying to gain agreement with engineers who do not have an extensive knowledge of UML. This problem highlighted our emerging opinion that, although traceability models are relatively easy to define, integrating data and process into one model view was not a suitable approach for achieving simplification and ease of use objectives.

Furthermore, we observed that different project team members gave input the TRAP than the TRAM. Moreover, for continued improvements and maintenance the ownership of the models is given to distinct project members. Within Ericsson it was the process team who took ownership of TRAP and the Requirement Manager who took ownership of TRAM. Therefore we believe that dividing the process and data models into two separate but related models is a good approach.

While carrying out this study we learned a number of lessons. As a result, a set of best practices for developing traceability MDA-style models can be distilled based on these experiences. While each organisation or research effort has its own particular concerns, we believe that the following steps should lead to consistently good traceability modelling approaches:

1. Begin with the findings from a case study or industrial survey to gain a better understanding of the problems that you are trying to solve. While metamodels are technology and domain independent one should still understand what problems the models are trying to solve. Using the empirical data define precise objectives for each model that you create. If a model does not map to a problem, then the model itself is in question.
2. Examine as many Requirement Management Plans or similar document that describe the traceability data to build up a good understanding of the traceability data, for example, traceability items, their attributes and the relationships.
3. Identify resources in the organisation who have modelling expertise or those who have an understanding of the traceability situation at different development phases. For example, a designer, a tester and a manager. Identify champions who will answer questions informally either on-site or electronically when you are back at the research institution. In this case a number of resources started to make sketches or gather data when the researcher was absent which provided us with extra information that was valuable to the overall modelling effort. One of the problems we faced here was the lack of testers who were willing to take part in the development of the models. A large number described a lack of UML experience as the major cause.
4. Begin discussions with a training organisation as soon as possible. Try where possible to have their participation in the group modelling sessions. If the modelling effort is to succeed, the training organisation must firstly understand the models and secondly believe in the solution that they promote. In our situation, we discovered a trainer who was developing course material on the MAR's traceability tool. While this resource did not partake in the modelling effort due to other commitments, he did provide us with course material and his pedagogical skills proved very useful in describing the main modelling elements that he would use in a training situation.
5. Carry out guided inspections of models using a variety of different resources from the organisations, fine tuning the profiles at each review. One of the problems with this approach is that you discover contradictions between the stories from different resources. However, by discussing these contradictions you gain a deeper understanding of the traceability domain.
6. It is impossible to create the M1 and M0 models without active participation of domain specific resources. All of our M1 models were created while working with the

Requirement Manager, Configuration Manager, M&T's and process team from the OSS-RC domain.

7. When all models are complete use presentation material to illustrate the models to the different audiences interested.

The lessons that we learned along the way can be best summarised as follows:

- *Lesson 1:* Before any attempt at modelling you must first acquire a full and thorough understanding of the problems that you are trying to solve. In this study, our initial efforts unfortunately produced useless models, which were thrown away on analysis of the empirical data. Once we understood the problems we were trying to overcome, for example, inconsistent terminology and creating a framework for gaining agreement from a variety of software engineers in the development lifecycle, which promoted reuse and better communication; only then could we start our modelling effort properly. Unless the models being developed provide a clear and useful solution to a problem only then should they be created. On the other hand, the experience we gained by creating these throw away prototypes was invaluable to our understanding of the OMG's MDA, UML and UML profiles.

- *Lesson 2*: One of the reasons for the success of our modelling approach was due to the flexibility of UML. It helped us create a set of models from a number of different traceability perspectives, from the abstract metamodels to the lower level application models. UML, encourages the partitioning of models into manageable pieces. In addition, relationships between model elements were easily maintained at all levels of abstraction. These models are easily understood by a variety of different roles and we therefore recommend that future research into traceability utilises UML as the standard modelling language.

- *Lesson 3:* To define the Layer M2 metamodel we used a UML Profile which was especially useful for accurately modelling and expressing specialized traceability semantics using this UML extension mechanisms. These profiles are an efficient and effective approach for modelling abstract traceability elements while still conforming to the OMG's standards. Most of the model elements that appear in our models do not appear in UML. Profiles provide an easy approach of using a standard language to develop sometimes obscure modelling elements like Traceability Suite or LogAction. Profiles are also a useful mechanism for managing model markings. Marking is a step or technique in MDA in which additional information, not within the semantic scope of the model itself, can be added to a model, solely used in this case, to provide additional information that can be referred to when there is a time gap between modelling efforts.

- *Lesson 4:* The M0 models are the most difficult to describe in modelling notation. At this level simple and easy to use notation should be used. We discovered in both the TRAM and the TRAP that local notation was the simplest approach. For example, using real-world diagrams like spreadsheets or Power Point style notation which is familiar to the organisation. The audience who use these models are not interested in complex UML object diagrams. Moreover during discussions and courses standard diagrams are easier to understand.

▪ *Lesson 5:* Models are just simplifications of complex real world scenarios; they do not automatically mean that an organisation will make strategic changes based on the models. Our models lacked cost analysis metrics and did not answer some of the upper managements cost related questions. While we gained the support from many project members in OSS-RC, especially the Requirement Manager and process team, we did not succeed in making any strategic changes based on our work. However, this was not a primary objective. Therefore, we believe that while the models proved successful at elevating the inconsistencies in terminology and promoted better communication and reusability they failed from a cost benefit analysis. The simple truth remains that models are an excellent approach to solving problems but they are not a complete solution onto themselves.

## 7.9 CONCLUSION & FINAL DISCUSSION

Our aim here was to demonstrate the suitability of a modelling approach to define the core semantics involved in traceability and to apply the models at an application layer. We see this investigation as a step towards the achievement of this goal. While we have focused here on the traceability data in the telecom's domain, further application of these models in other domains needs to be investigated. Furthermore, certain aspects of traceability were considered outside the scope of this study. For example, we did not focus on legacy requirements which in many projects are a major consideration.

Semantic modelling of traceability involves gathering a large variety of terms and rules from different perspectives in the software development lifecycle. Thus, it seems obvious to choose the most expressive formalism languages. We based our approach on the OMG's MDA specifications, to overcome the problems identified in the case study which accommodates easy to use modelling techniques promoting better communication, reusability and a general framework for gaining agreement across an entire development organisation on matters relating to traceability. The main purposes of the TRAM is to provide an adequate and adaptive way that is based on uniform principles for describing all the notions, relations, rules, the behaviour and anything else that proves necessary for discussions on all matters related to traceability. Implementing the TRAM and applying it to an application domain we have found that our modelling approach expedites the understanding by different roles involved in traceability. By applying the metamodel to a lower level we have simplified the traceability data concepts and stored them in a knowledge base in a consistent way.



So far the stated objectives for the TRAM have been met. Success of this project in terms of quality research results has been aided by the collaboration between the researcher and the OSS-RC project teams. This collaboration has been instrumental in providing a clear understanding of the challenges faced in traceability. The following research results have been obtained:

1. A framework for traceability across telecom engineering projects.
2. Meta-models for a representative set of traceability notations.
3. Layer M1 models using a wide variety of perspectives.
4. Layer M0 examples taken from the OSS-RC R5 project.

. We hope that this approach will help generate new insights and foster new research into the discovery of new techniques and the use of traceability.

# Chapter 8 TRACEABILITY PROCESS (TRAP)

## 8.1 BACKGROUND

"Software processes are software too!"

- Lee Osterweil (1987)

To respond quickly to the market, the organization must be able to create, manage and optimize dynamic business process. Building up dynamic processes using multi pre-defined process models can lead to higher efficiency in fulfilling complex product development. (Zhang, 2004) Organization's software development process ties together all activities and practices addressing all practices in the development of quality software. Software processes should include practices in project management, requirement engineering, change management, design, build, test and of course traceability.

Software development organisations are today concerned about the underlying processes with an increased focus on the assessment of these processes to improve the quality of the processes and hence improve the quality of the final products. Software process assessment and improvements are very complex activities due to the complexities of developing not only products but product families. In order to manage this complexity, it is useful to establish a conceptual process architecture, which includes all the aspects of this development process. This conceptual process architecture must involve the definition of the metamodels and models necessary in order to carry out an assessment and improvement process in an effective and integrated way. (Garcia et al., 2003)

Because the adoption of traceability practices goes to the very heart of systems development, there is still much debate about how far to go in adopting such a new and fundamental practice. In particular, there is intense debate about how much the processes involved in traceability need to be customized depending on the project needs. According to Domges and Pohl, "If requirements traceability is not customized it can lead to an unwieldy mass of unstructured and unusable data that will hardly ever be used." (Dömges and Pohl, 1998) In order to customize traceability a formalized approach must be applied in the development of traceability processes.

In 1998 Jarke recognized the importance of traceability in the management of software change and evolution, observing that "Tracing is a sub process of evolutionary system development that supplies and exploits these traces", and that "Requirements tracing is emerging as an effective bridge that aligns system evolution with changing stakeholder needs". (Jarke, 1998)

In this chapter we motivate and design a TRAceability Process (TRAP) that can be used by an entire organisation based on modeling standards, industry best practices and research efforts. This is no simple task because the usage of traceability may vary considerably across systems development efforts, ranging from very simplistic practices aimed at satisfying customer mandates to very comprehensive traceability schemes for managing the entire product development process. Therefore, an absolutely fundamental property of any standardized process model for implementing traceability must be well suited to a wide variety of perspectives, from the entire organisation to the specific software

engineers working on their day-to-day tasks. The objective of such a process model is to help with the development, and assessment of a quality traceability process.

The purpose of this chapter is to describe the author's methodical design of a generalised process model called the Traceability Process (TRAP). This Traceability Process has the fundamental objective of describing how to understand, manage, analyse and assess a process that defines traceability as a core practice.

### 8.1.1 Software Process & Traceability Process Essentials

Before launching into an in-depth analysis of traceability processes, in this section we briefly introduce the essential concepts of software process engineering and how they relate to traceability as follows:

- The Software Engineering Institute (SEI) states that "An essential aspect of software engineering is the discipline it requires for a group of people to work together cooperatively to solve a common problem. Defined processes set the bounds for each person's roles and responsibilities so that the collaboration is a successful and efficient one" (SEI, 2007b)

- A software process refers to the set of tools, methods, and practices used to produce a software product (Humphrey, 1989). It is a set of partially ordered process steps, with sets of related artifacts, human and computerized resources, organizational structures and constraints, intended to produce and maintain the requested software deliverables.

- A software process is composed of lifecycles, phases, activities, steps, iterations artifacts and roles. (Kruchten, 2003)

- Software process modelling is the task of developing software process models, which abstract the real world into sets of "entities" which defines all aspects of traceability and the information flow. A software process has process-components that provide help and guidance to produce the models that address the problem, solution, providing the role descriptions, activities, tasks, and deliverables that are the end result of the activities carried out within the process. The main idea of a process metamodel is the interplay of three basic elements: Process Roles that are responsible for and execute Activities that consume and produce Artefacts, which in our case are Traceability Items. Process Roles, Artefacts (containing Traceability Items) and Activities are all process definition elements. Relationships between these basic elements are illustrated in Figure 8-1, *Conceptual Process Model*, below. A Process Role is Responsible for Traceability Items and a Process Role performs Traceability Activities on the Traceability Items. While this is a simplified process model, it still encapsulates some very fundamental principles of traceability.

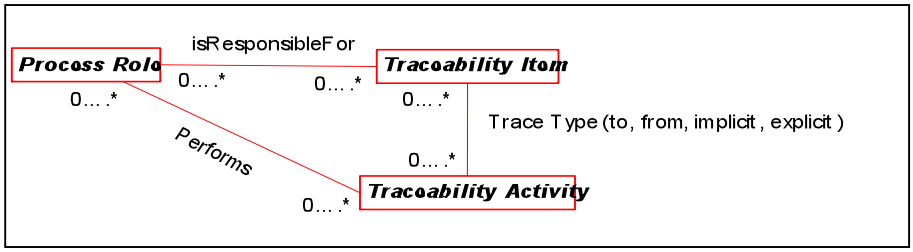



**Figure 8-1 Conceptual Process Model**

## 8.2 MOTIVATION FOR TRAP

As our survey illustrated of the 19 organisations who took part they all used informal process descriptions to describe their processes, usually in spreadsheet or document format. On further investigations during the interview sessions nearly all the organisations described their processes as not a true or an up-to-date representation of the current development practices. Some respondents believed that document or spreadsheet approaches were not user friendly and individuals often found it difficult to access the specific practice or activities to help them with the task underhand. Others believed that the processes required a body of knowledge about the process domain and in many cases the particular engineer did not have this body or knowledge, finding the language or terminology difficult to understand. Other problems identified included a shortage of training on processes, lack of a process team, lack of management commitment to process development and in the rare cases where a process team actually did exist their process output did not reflect the true nature of the problems that were faced, due to their distance from the real problems. In summary the problems with using informal process descriptions include the following:

1. Informal process descriptions, for example process description documents, are lengthy, perhaps hundreds of pages, and often not very well structured thus making information retrieval difficult.

2. Handbooks often lack a role-specific view which makes it difficult for project members to play particular roles to find relevant information with respect to their specific problems. In many cases the process handbooks are viewed as either too detailed leading to difficulties in information retrieval or not detailed enough causing a lack of confidence in the process information.

3. It is often difficult to modify informal process descriptions. This aggravates the adoption of the processes to the organizational contexts in which they are used. Tailoring of informal approaches is often a difficult undertaking and often not carried out by projects leading to a disparity between real situations and abstract process descriptions.

4. The dynamic behavior of informal processes is detrimental to their acceptance. Inconsistencies, ambiguity, and completeness of software process descriptions cannot be ensured on an informal basis. Reviews by numerous project team members, is often needed to ensure suitability exacerbating the cost both in time and money. An added cost that is often not feasible given the budgetary constraints of most projects.

These issues gave rise to the question: Can we replace these informal descriptions with process models?

While the problems identified in the case study in Ericsson concurred with the findings from the survey there were differences often because of scale or magnitude of the development effort. Many of the problems began at an organizational level and propagated down to the individual project space. The problems we encountered during the case study included:

1. Poor end-to-end definition of traceability causing gaps in the definition of certain roles and their corresponding traceability. This leads to a breakdown in the practice of traceability across the entire development process. For example, some testers argued that the process did not define their traceability tasks, therefore it was not an essential practice that they must carry out.

2. Lack of a unified process framework. In our case study we discovered that the OSS development was sub-divided into three separate product lines (GSM-OSS, CN-OSS, RANOS) each with their own processes. This lead to inconsistencies in the terminology being used by roles from different development organisations that interact regularly with each other.

3. Poor process training. In times of economic downturn cut backs on training budgets is often one of the costs saving measures introduced. During our case study we observed that process training was very rarely if ever carried out. This was partly due to the lack of a unified product development process, and partly because there were few resources who understood the entire process leading to lack of competence to deliver training. Of course, cutbacks in process development budgets lead to a shortage of staff to define and train on the process. This can lead to a loss of an understanding of processes or lack of a transfer of process knowledge within an organisation.

4. Poor communication between success critical roles often caused by a lack of a unified process.

The Centre of Excellence for Traceability describes the problems and challenges faced by the research and academic community in the Grand Challenges Technical Report. (Hayes et al., 2006) They provide the following motivating challenges that must be overcome to further the practice of traceability:

- *Process:* "In order to generate and maintain quality and sound traceability information, an organizational process is required; however, traceability is often not included as an integral part of the development lifecycle" We interpret this statement as a motivation for the development of a traceability process framework that can be easily integrated with the other development processes.

- *Traceability Knowledge:* Traceability is critical to the success of software and systems projects, but there is little consensus on best traceability techniques and methods, few recorded best practices, and a general lack of resources providing a body of knowledge in the field. TRAP contains a body of knowledge on the implementation of traceability at different levels.

- *Training & Certification:* Organizations need a way to identify individuals skilled in traceability methods and practices. They also need a tool for describing the traceability process.

- *Scalability:* Current traceability methods have been developed to trace well structured data, however, many industrial datasets are composed of large and unstructured documents that are hard to trace. The challenge of building a process framework that scales to large and small organisations may elevate this problem.

- *Human Factor:* Develop techniques to help humans overcome the semantic barrier in tracing to artifacts of other stakeholders**.**

- *Methods & Tools:* Develop effective methods for tracing multimedia artifacts. Build methods and tools with maximized levels of automation to support the entire trace life cycle including link construction or generation, link assessment, link maintenance, and link use.

- *Tracing across organisation barriers:* End-to-end tracing is critical to the success of a project, but organizational boundaries, for example, those between marketing and manufacturing or development, or those due to outsourcing, make it difficult to achieve due to differences in skill sets, process, terminology, and tools

This motivated us to develop a:

- A Traceability Framework that was platform and technology independent that addresses the challenges above.

- TRAP has also been designed to specifically address some of the human problems discussed above.

## 8.3 OBJECTIVES OF TRAP

There is no general agreement as to whether it is possible to describe real and practical traceability practices in a complete formal model. The primary objective of TRAP was to provide a modelling approach that unites all the factors that influence traceability into a consistent and coherent process model. A basic premise guiding our approach is that the effectiveness of describing traceability is determined by the quality of the process that is used to describe and maintain the practice. Thus, traceability improvements can be effectively achieved by improving the quality of the definition of traceability processes; consequently, another primary goal of this work is to facilitate and enhance the evolution of the traceability process toward greater effectiveness, efficiency and reliability. However, modelling a software traceability process supports the goal of improving traceability by providing a mechanism for:

- Building a framework that satisfies some of the problems identified during the case study and survey.

- Recording and understanding all the process elements involved in traceability.

- Communicating and promulgating all aspects involved in traceability both to the user community and the academicians.

At a more detailed level, we have identified five secondary objectives for the development of the TRAP:

- Facilitate our understanding of traceability by abstracting from the real world at one level and applying the abstractions at lower levels.

- Enable effective communications regarding traceability.

- Facilitate reuse of effective traceability practices.

- Support evolution of the process for future research and developments by the user community.

- Provide a framework for *capturing, analyzing*, *assessing and improving* the development of a traceability process that can be used by large and small organisations.

A lot of the current literature on traceability describes “what” steps need to be taken to implement traceability, or how they are performed. However to improve your knowledge you need to have a deeper understanding that reveals the “why’s behind the “what’s” and the “how’s”. Because traceability is carried out by many software engineers involved in the implementation of traceability by modeling a multi-layered approach we gain an understanding not only of the steps involved in practicing traceability but the factors that influence traceability. By considering different options for traceability we gain a deeper understanding of the problems faced which assist us to find a suitable solution that meets the needs of the community.

The second objective focuses on effectively communicating the description of practicing traceability to others, such as software engineers, managers, and customers. Process models are especially useful for sharing knowledge and expertise.

Reuse, the third objective, enables a specific software process to be instantiated and executed in a reliably repeatable fashion across multiple software projects or research efforts by what Osterweil has suggested that ". . . the most important benefit of process [modeling] is that it offers the hope that software processes themselves can be reused" (Osterweil, 1987)

To support the evolution of the traceability practices through (1) providing a framework that captures lessons learned, and tailoring; and (2) analyzing the effectiveness in a laboratory or simulated environment before actually implementing them in the field. After successfully tailoring the process models, the outcomes should be formalized and stored as part of the model, so that they can be consistently applied in the future.

Finally, by creating metamodels and models it allows us to better capture, analyse, understand and improve the development of a traceability process. By formalizing a traceability process we can assess the quality of the process and make recommendations for improvements.

The objective for TRAP is to organize all the process elements at different levels of granularity which is adaptive to different traceability situations. This is an experimental process and while the lower levels illustrate that we have applied this against a real world situation we still understand that further project enactments of the models is required in future research. We merely state that our experimental approach provides a novel approach for modelling a traceability process.

## 8.4 INPUTS TO TRACEABILITY PROCESS

The inputs to our process framework are shown in Figure 8-3 below, *Inputs to TRAP*, and include:

1) Software process standards, best practices, state of the art.

2) Research efforts (literature, research projects, empirical studies) by academia and software research groups;

3) Industry knowledge and a variety of modelling expertise

4) Empirical data gathered from the case study and survey.

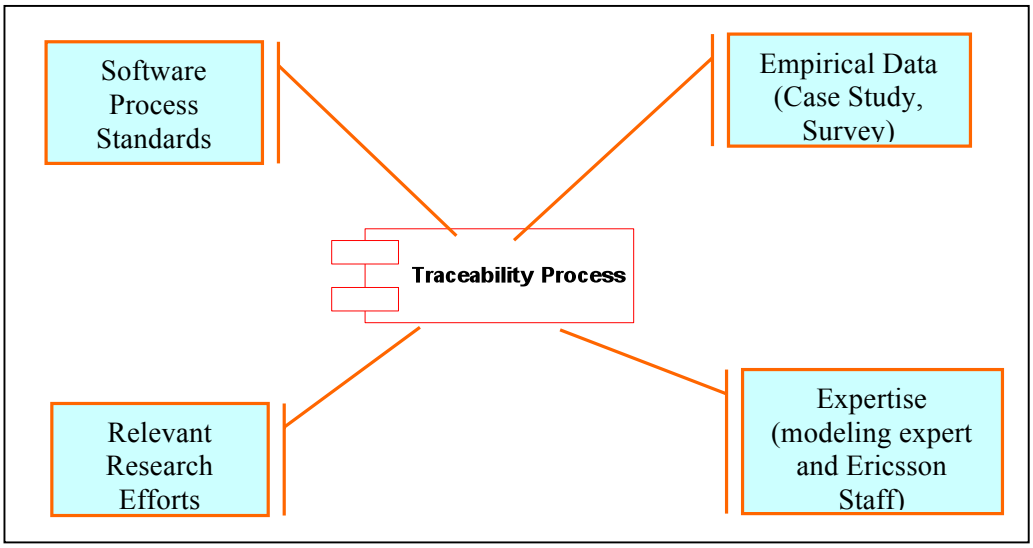



**Figure 8-2: Inputs to TRAP**

Table 8-1, *Process Modelling Method*, we summarise the various steps we followed in the creation of the Traceability Process Framework.

| Stage | Activity | Data Source | Outputs |
|---|---|---|---|
| Process Prototype (2005) | Develop M2 Process Metamodel | Current Literature (SPEM, IEEE, MOF etc. | Process Metamodel Layer M2 |
| Desk Check & Model Check | Complete model checking activity with process modelling expert. Model walkthroughs, scenario tests etc. | Expert Modeller | M2 Process Metamodel assessed by process modelling expert |
| Prototype introduction (early 2006) | A series of workshops carried out where process metamodels were demonstrated to the OSS-RC process team and the requirement manager. New process elements identified and added to metamodel | Input from participants. | M2 metamodel refined for OSS-RC domain. |
| Development Unit Layer (M1) (2006) | During participation workshops, we designed and analysed the models. We assessed the models against the objectives particularly against the criteria for making improvements to traceability | Participation input from OSS-RC(RM, CM, Process Team, Systems) Assessment by OSS-RC modeling expert. | M1 Layer Models |

| | | | |
|---|---|---|---|
| Project Layer or Enactment Layer (M0) 2006-2007 | During project process definition meetings the process M1 models were instantiated and new M0 web based pages were developed | Project representatives Project Manager, Process Team, RM, CM | M0 Layer Models |
| Assessment 2007 | Analyse and assess modeling effort | Modeling expert, Requirement Manager, Project Manager OSS-RC | Assessment |

**Table 8-1Process Modelling Method**

## 8.5 TRAP MODELLING APPROACH

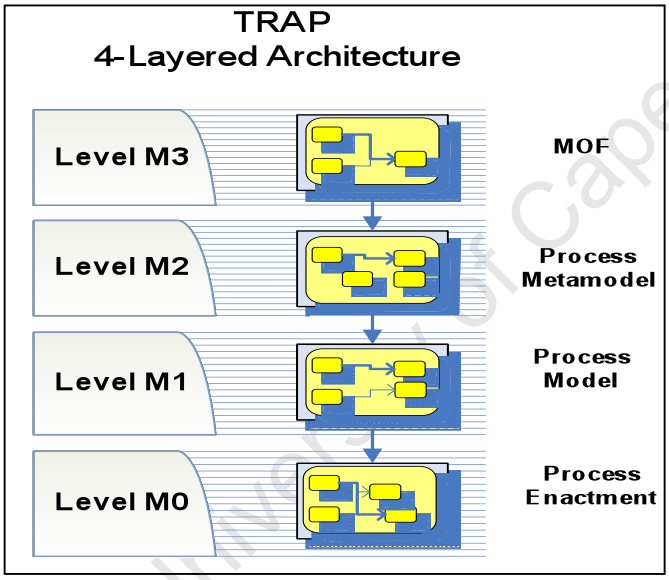



Figure 8-3 TRAP Four Layered Architecture

As shown in Figure 8-2 above, *TRAP Four Layered Architecture*, and already described in Chapter 6, we adapt the recommended OMG's four layered modelling approach.

Our traceability process framework encompasses a series of models which describe the implementation of traceability. Persons are the least formalized feature in present traceability practices. Yet, their importance is obvious: they present a non-deterministic and subjective behaviour, which plays a decisive role in the results of the implementation of traceability. Hence, the definition of a traceability process model must state all the elements needed to implement traceability, but also the way in which this model was executed or enacted. This idea leads to the notion of static and dynamic parts of a model. The static part is given by the description of the tasks, documents, tools and resources that take part in traceability. On the other hand, the dynamic part consists of a description of the way in which traceability is implemented, so, it mainly focuses on questions like what and how must be done to develop a piece of the model. The systematic description of both parts not

only helped us in understanding traceability, but also makes feasible the construction of a model that can be reused by researchers and practitioners in any domain.

It is important to note that one of the key benefits of creating any graphical representation of a process is that it will helpful to development organisations or research efforts to find inconsistencies in the processes and to clarify the interactions between the roles, activities, artifacts and so on.

### 8.5.1 The Who, What, When of TRAP

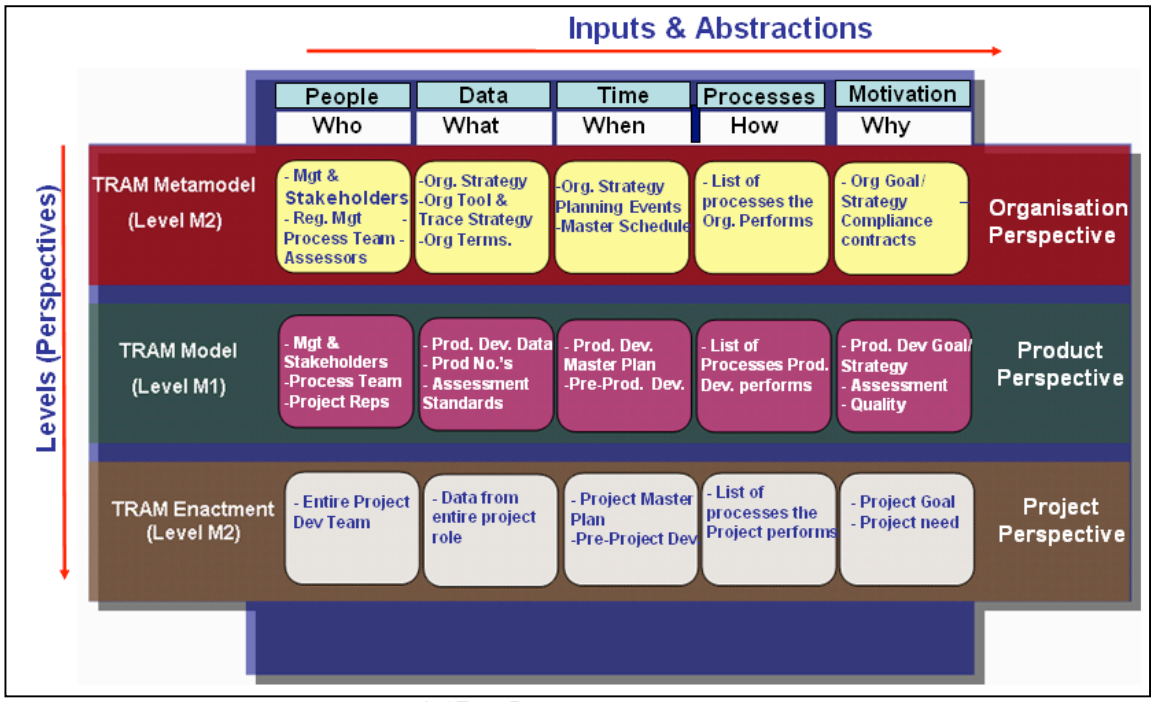



Figure 8-4 Traceability Process Framework

In Figure 8-5 above, *Traceability Process Framework*, we illustrate that a traceability process traverses the boundaries of organisation, product and project perspectives. It is important to describe the "who, what, when, how and why" in the development of a process. As can be seen from the diagram above we illustrate the layers again except this time we illustrate that each layer is from a different perspective. M2, is a template for creating a traceability process. So who would use this? An organisation, for example Ericsson, would use the M2 metamodel to create an organisation wide template of traceability. The key players involved in such an endeavour would include the strategic management team, for example, representatives from corporate methods and tools, strategic product management, requirement management, configuration management and any key resource that understands the organisations strategic traceability goals. Representatives from the customer perspective would also complete the traceability picture at this level. The main inputs into the creation of the metamodel would come from the organisations overall traceability goals. At the M1 level representatives from the development management team, for example, the Ericsson's OSS-RC product, would build a process model that could be used by the entire product development organisation. While at the enactment layer, which is the project perspective any key players who will involved in the process definition or roll out of the process to the project team. The inputs to the different layers varies from generic standards or organisations strategies to product development inputs to specific project

variables, like the specific tasks a software engineer needs to carry out to implement traceability. In summary each layer has a specific perspective:

- Layer M2: Overall organisation
- Layer M1: Development Unit
- Layer M0: Individual Projects

### 8.5.2 Underlying Technologies

The OMG use the four layered architecture to demonstrate how the different standards relate to each other. Similarly we base our modeling effort on OMG standards. The process metamodel (or M2) is based on the OMG's Software Process Engineering Metamodel (SPEM). The Software Process Engineering Metamodel (SPEM), is an object-oriented specification that tells us how to model a software process. UML is used as the notation. The SPEM is a metamodel for defining processes and their components. The purpose of SPEM 1.1 was limited at providing process descriptions to be read by humans and to be supported by tools, but not to be executed. (Bendraou et al., 2005) SPEM defines Life Cycle, Phase and Iteration that are used for dynamic structuring of the process. A Life Cycle defines the ordering of Phases, which in turn can contain Iterations. A Process must have one Life Cycle. SPEM also defines elements that are meant for organizing other process elements from the viewpoint of process authoring, assembly and reuse. In the chapter conclusion we evaluate SPEM and its usefulness in the development of the TRAP.

### 8.5.3 Design Considerations

The M2 and M1 process metamodel and model has been architected with the following design considerations:

- Modularity. We organize the models with loose coupling by applying a group of constructs into packages and organize process elements using metaclasses, classes and objects.
- Layering. Layering is applied by using package structures and applying the four layered architecture for describing process elements at different levels of abstractions.
- Extensibility. Because we use the UML this can be extended by using Profiles.
- Reuse: To promote reuse of processes the process metamodel identifies "the common, generic features of process models and represents them in a system of concepts" (Rolland, 1998)

At the M0 level, which is at the development unit and project level, we define simple design principles as follows:

- ease problems in terminology and usage;
- make it easier to adopt new processes with the changing environment;
- provide concise, unambiguous and complete software process descriptions;
- avoid the need for scripted process descriptions of great length, perhaps hundreds of pages, which are often not very well structured thus making information retrieval difficult;

- make it easier for project members to play their particular roles and help them to find relevant information with respect to their specific problems on requirements, traceability, and change control.

- make it easier for all in the project team to understand the dynamic behavior of a process and the repercussions of changes at any level.

## 8.6 TRAP METAMODEL (LAYER M2)

As shown in Figure 8-6 below, *M2 Process Metamodel Example*, we define the metamodel under the following headings: (we describe each one of these headings in the following sections)

1. Foundation Packages
2. Process Structure
3. Process Components
4. Process Lifecycle.

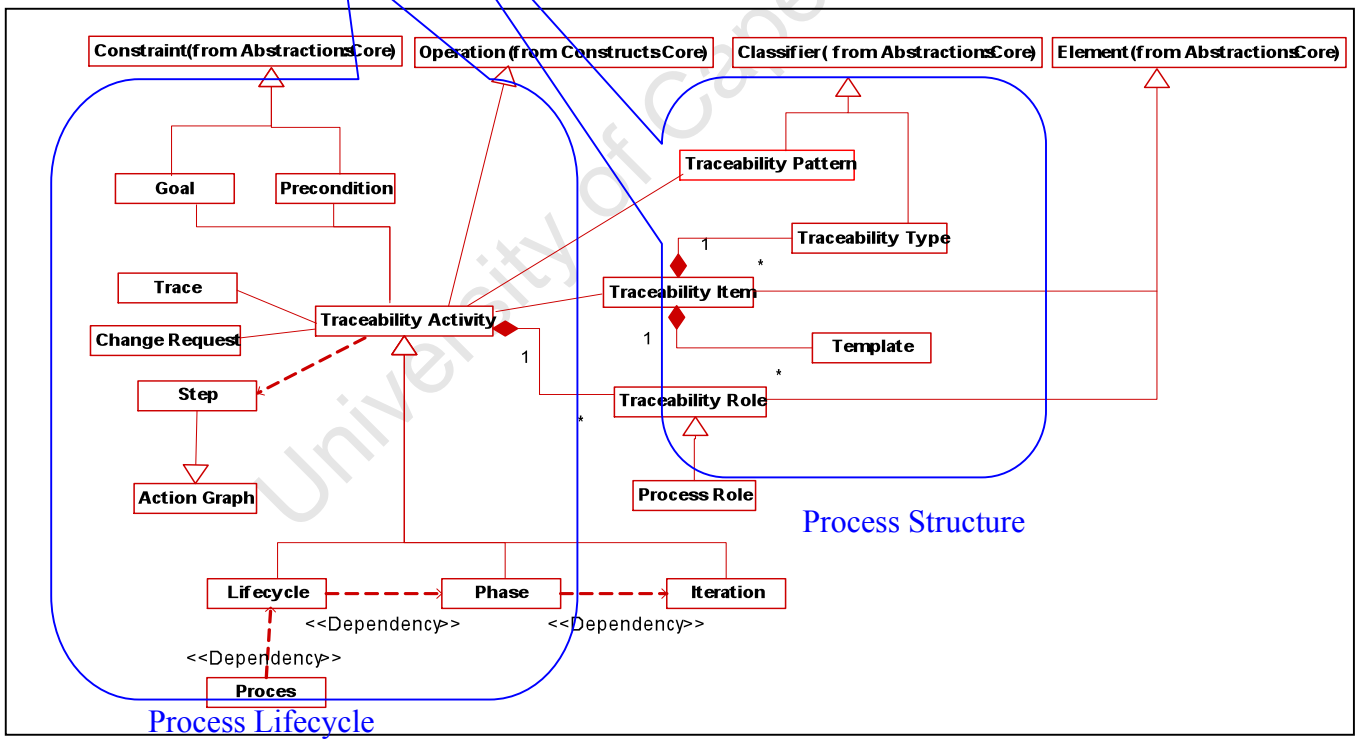



**Figure 8-5: M2 Process Metamodel Example**

### 8.6.1 Package Structure

UML is a metamodel and the TRAP metamodel is built by extending the UML metamodel reusing elements where possible. A package is a general purpose mechanism for organizing modeling elements into groups or simply process descriptions into self contained parts. These parts can then be placed under configuration and version management and used in assembling and tailoring software development processes. The UML has a pre-defined package structure. All metamodels that reuse the UML should clearly specify which packages they reuse, and further clarify which packages are imported without change, and which packages are imported and extended via specialization.

Similarly to the TRAM, in the UML the Core package is the highest level package not depending on any other package. As shown in Figure 8-7 below, *TRAP extends the UML Core package*, we illustrate that before starting modeling our process we must extend the *UML Core Package*. The Core package provides basic constructs for creating and describing meta-model classes.

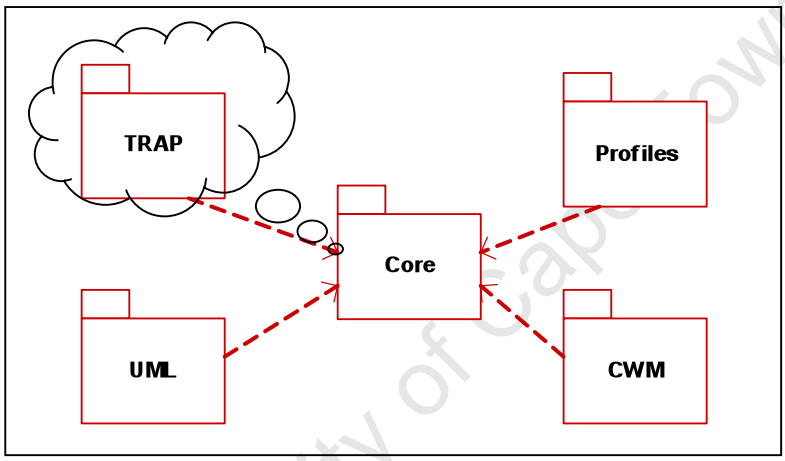



**Figure 8-6: TRAP extends the UML Core package**



As illustrated in Figure 8-8, *The two main packages to extend the UML metamodel*, the two highest packages are the TRAP Extension and the TRAP Foundation package, which is a subset of UML 2.0 and the TRAP Extension package, which adds the constructs and semantics required for defining a traceability process.

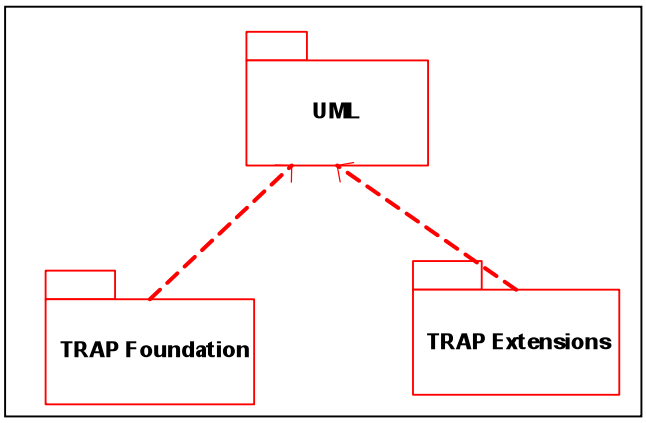



**Figure 8-7: The two main packages used to extend UML metamodel (Conceptual)**

### 8.6.2 TRAP Foundation

The TRAP Foundation is subdivided into the following sub-packages:

| Sub-Package Name | Overview |
|---|---|
| TRAP_Foundation:: Core | The *Core* package is a complete standalone metamodel. It contains the structural backbone of any metamodel and is divided into a number of finer-grained packages to facilitate flexible reuse when creating metamodels. These packages define the model elements to define relationships and the model elements to define dependencies. |
| TRAP_Foundation:: Datatypes | Contains definitions of the data types. |
| TRAP_Foundation:: Actions | The TRAP_Foundation::Actions package is a subset of the UML 1.4 Common_Behavior package. The Common Behaviors packages specify the core concepts required for dynamic elements and provides the infrastructure to support more detailed definitions. |
| TRAP_Foundation:: Statemachine | The StateMachine package defines a set of concepts that can be used for modeling discrete behavior through finite state transition systems. |
| TRAP_Foundation:: Activity_Graphs | An activity specifies the coordination of executions of subordinate behaviors, using a control and data flow model. Activities may describe procedural computation. In this context, they are the methods corresponding to operations on classes. Activities may be applied to organizational modeling for business process engineering and workflow. |

**Table 8-2: TRAP Foundation Package**



### 8.6.3 TRAP Extension Package

The TRAP Extensions add adds the constructs and semantics required for the traceability process. The TRAP Extension packages are illustrated in Figure 8-9, *Trap Extensions*, below. The five sub-packages are TRAP Basic Elements, TRAP Components, TRAP Dependencies, TRAP Process Lifecycle and TRAP Process Structure.

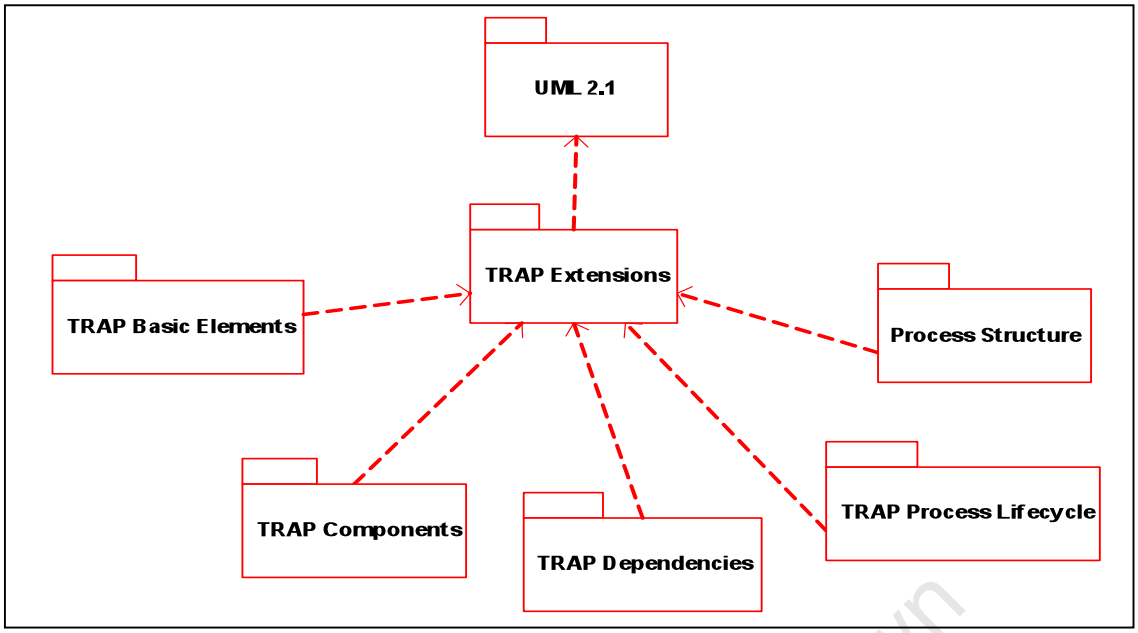



**Figure 8-8: TRAP Extension**

| Package Name | Brief Description & Components |
|---|---|
| Basic Elements | The basic elements of the process; External Descriptions, Guidance, Patterns |
| Structure | Consists of Artefacts (which contain traceability items) and their types, Work Definition which is the operations that describe the work performed in the process, and Activity that describes a piece of work performed by one Process Role: the tasks, operations, and actions that are performed by a role or with which the role may assist. An Activity may consist of atomic elements called Steps. Finally a Process Role defines responsibilities over specific Traceability Items, and defines the roles that perform and assist in specific activities. |
| Components | A Process Component is an internally consistent and self-contained chunk of process description. It consists of Process, Package and Discipline. The Discipline is used for representing activities for example, traceability workflows, like implementing traceability practices. While Process is also is intended to stand alone as a complete end-to-end process. |
| Dependencies | This package defines the different types of dependencies between the modeling elements. *This is a very important aspect of any modelling effort into traceability. However, this was covered in Chapter 7, the TRAceability Model (TRAM), which has the objective of defining the dependencies between the traceability items.* |
| Lifecycle | Package Process Lifecycle introduces concepts to describe the |

| | |
|---|---|
| | lifecycle in terms of goals and precondition, and to allow the decomposition of the process lifecycle into phases and iterations. A Life Cycle defines the ordering of Phases, which in turn can contain Iterations. A Process must have one Life Cycle. |

**Table 8-3: Sub-packages from TRAP Extension**

In the rest of this section we will describe each one of the packages, their model elements and the relationships.

#### 8.6.3.1 Basic Elements

There are three basic elements in this package. External Description, Guidance and Patterns.

**1. External Description:** With every Model Element is associated one or more External Descriptions, which contain a description of the Model Element suitable for a reader of the process description. An External Description has four attributes of type String,

- content: A natural language description of the Model Element.
- name: The name of the Model Element in a natural language.
- language: The name of the natural language used for the value of content and name.
- medium: A description of the medium and format of the External Description.

**2. Guidance:** Is something that provides direction or advice in the implementation of traceability. Best practices can be described inside as a Guidance that organisation should follow to improve the performance of certain practices. Checklist is a kind of Guidance. A checklist is a document representing a list of elements that need to be completed. Guideline is a kind of Guidance. A Guideline is a set of rules and recommendations on how a given task is completed. For example, a traceability guideline could describe how to implement traceability tasks, perhaps like, “trace_to”, “trace_from”. Any guideline that describes a traceability task. Template is a kind of Guidance. A Template is a predefined document that provides a standardized format for a particular kind of artifact.

In previous chapters we described Traceability Patterns as a structured approach for describing a solution to a problem. This is the first introduction to Traceability Patterns the context of software process models. Patterns are described using templates and the templates have a name, a problem description, a context, a consequence, the forces involved and they describe a solution. Therefore Traceability Patterns are an extension of Guidance which is an extension of ModelElement (From the Core Package in UML)

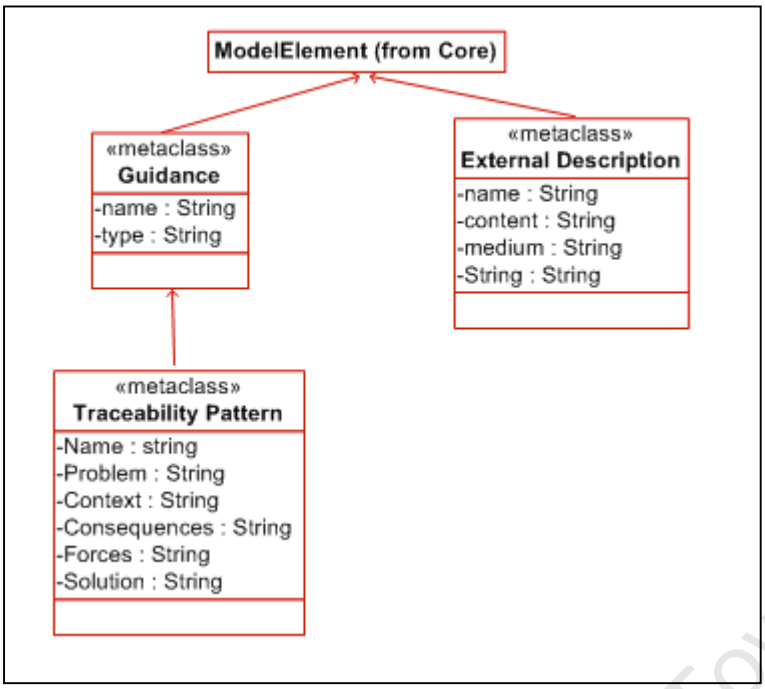



**Figure 8-9: Basic Elements: External Descriptions, Guidance, Pattern**



### 8.6.4 TRAP Extension: Process Structure

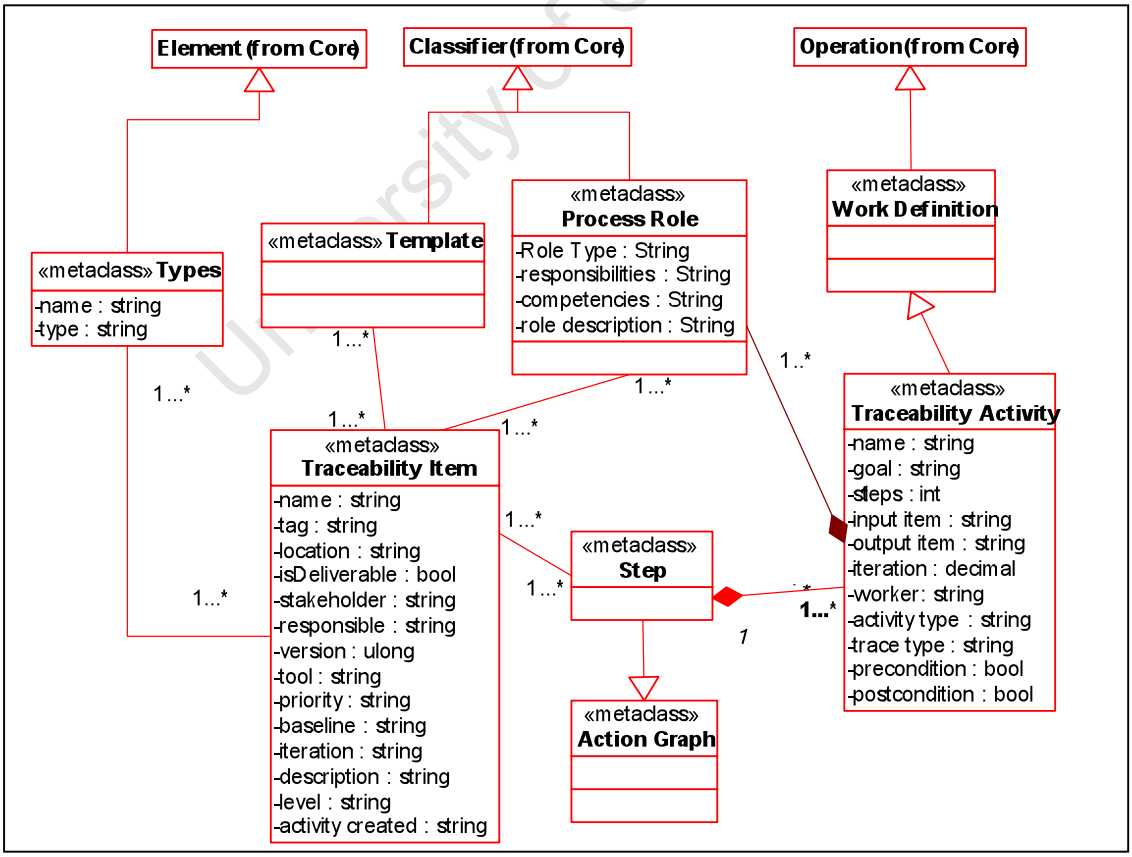



**Figure 8-10: TRAP Process Structure**

Using the data taken from Figure 8-11, *TRAP Process Structure*, we define in the tables below each model element, their attributes and their associations.

| **Traceability Item** | |
|---|---|
| **Description** | **Attributes** |
| A traceability item is defined as "Any textual, or model item, which needs to be explicitly traced from another textual, or model item, in order to keep track of the dependencies between them". A general definition of a traceability item would be a "project artifact" or work product. We combine both definitions stating that traceability is the technique of identifying, documenting and maintaining trace information between any traceability items in the product lifecycle.<br><br>The Operations are described as:<br><br>1. **getVersion()** : VersionReturns the current version.<br>2. **setVersion(v : Verdict)** Sets a new version value.<br>3. **getResponsible():** RespnsibleReturns the person responsible for the item.<br>4. **setResponible(r: Responsible)** Sets the name of the person responsible for the item<br>5. **getBaseline():** BaselineReturns the baseline that the item belongs to.<br>6. **setBaseline (b:Baseline):** Sets a new baseline value.<br>7. **getIteration():**Returns the iteration that the item is to be developed. | ***1. name: string*** Stores a unique identifier for the traceability item. For example, 'name=rs1'<br><br>***2. tag: string*** Stores the tag that is used to identify the traceability item. For example a Requirement Specification could have the tag <RS>.<br><br>***3. location: string*** Stores the location of the traceability item. For example, the URL for where the traceability item is stored.<br><br>***4. stakeholder: string*** Stores the name of the stakeholder that requested the item. For example, "stakeholder=AT&T'<br><br>***5. responsible: string*** Stores the name of the role responsible for the item. For example, 'responsible=sub-projectA"<br><br>***6. version: ulong*** Stores the version of the item. For example, 'version=PA1" this is version PA1, preliminary version 1. Once the item has implemented the version will change to A1.<br><br>***7. Tool: string*** Stores the name of the tool where the item is stored. For example, 'tool=DOORS'<br><br>***8. priority: string*** Stores the priority of the item. For example, the item may have 'priority=high'.<br><br>***9. Baseline: int*** Stores the point at which the traceability item is put under formal change control. For example, the 'baseline=060606', states that the item is placed under source control on the 06 June 2006.<br><br>***10. iteration: int:*** Stores which iteration the traceability item is to be developed. For example, "iteration=3", describes that the item is to be developed in iteration 3.<br><br>***11. description: string*:** Stores a description of the item in text. For example, 'description=I want a web-page'.<br><br>***12. isUnderCM: Boolean:*** States whether the item is under Configuration Management.<br><br>***13. Activity created: string:*** says what activity in the traceability process created the item. |
| ***Associations*** | Traceability Items have Types, for example functional and |

| | |
|---|---|
| | non-functional. They are created and stored in Templates (or what most organisations call documents or artifacts, for example, requirement specifications, change requests or impact analysis documents). Traceability is carried out in a number of atomic Steps. |

| Traceability Activity | |
|---|---|
| ***Description*** | ***Attributes*** |
| A traceability activity extends Operation (from Core) and is a function, or task that can be identified and combined in the traceability process, which use up resources to carry out traceability practices. It describes a piece of work performed by one Traceability Worker: the tasks, operations, and actions that are performed by a role or with which the role may assist. | **name:string :** The name of the activity. For example, 'name=trace'.<br><br>**goal:string :** The goal of the activity. For example, 'goal=create traceability item'<br><br>**steps: int :** The number of steps with the activity. For example, 'step=5'.<br><br>**input item: string:** The input items to the activity. For example, 'input item= requirement specification, test specification'<br><br>**output item: string:** The output from an activity. E.g. 'output item= traceability matrix'<br><br>**iteration: int.** The iteration that the activity takes place. For example, 'iteration =3', means the activity takes place during the third iteration.<br><br>**worker: string** : The name of the worker that carries out the activity. E.g. 'worker=john'.<br><br>**Activity:** type : The type of activity.<br><br>**trace:** type : the type of trace<br><br>**precondition: Boolean**: The preconditions for an activity define conditions that must be true when the activity is invoked. May be assumed by an implementation of this activity.<br><br>**postcondition: Boolean:** The postconditions for an activity define conditions that will be true when the invocation of the activity is completed successfully, assuming the preconditions were satisfied. Must be satisfied by any implementation of the activity. |
| ***Associations*** | ▪ Activity inherits from WorkDefinition (from Operation (Core))<br>▪ An Activity *is a part* of a Process Role<br>▪ Activity uses Steps, which are atomic steps to carry out an activity. |

| Process Role | |
|---|---|
| ***Description*** | ***Attributes*** |
| A Process Role is a set of activities performed; usually realized by software engineers. The role describes how individuals behave when carrying out traceability activities and what responsibilities these individuals have. For example, the requirement engineer must trace the traceability items in the requirement specification work product with the network use cases. | **name:string :** The name of the role. For example, 'name=requirement engineer'.<br><br>**responsibilities: string :** The responsibilities of the role. For example, 'responsibility= trace to design'.<br><br>**competencies: string** : The competency of the role. For example, 'role=senior'.<br><br>**role description : string** : The role description described in text. For example 'role description = The requirement engineer is responsible for…." |
| ***Associations*** | ▪ Process Role is a specialization of Classifier, and thus may participate in inheritance relationships and associations within the process definition.<br>▪ A Process Role is responsible for a set of Traceability Items.<br>▪ A Process Role is the performer of Activities. |

| Template | |
|---|---|
| ***Description*** | ***Attributes*** |
| A pattern used to create documents. Templates are pre-existing forms that include standard text and spaces to fill-in-the-blanks with standard information. Templates saves time since each person does not have to create the document format on their own. Templates also allow information to be presented in standardized and recognizable formats for the reader. | **name: string** : The name of the template. For example, 'name=requirement specification'.<br>**location : string** : Where the template is stored. For example the template could be stored in a document management tool, or in the requirement management tool, 'location= RequisitePro'.<br><br>**underCM: bool** : Is the template being kept under configuration management. For example, 'underCM=Yes' |
| ***Associations*** | ▪ Each Traceability Item is created in a document which has a Template<br>▪ Template is associated with Classified (from Core) |

**Table 8-4: Process Structure Elements**

### 8.6.5 TRAP Extension: Process Lifecycle (Layer M2)

In this package, shown in Figure 8-12, *TRAP Process Lifecycle*, we introduce process elements that help define how the process will execute. Process Lifecycle introduces concepts to describe the lifecycle in terms of goals and precondition and allowing the decomposition of the process lifecycle into phases and iterations.

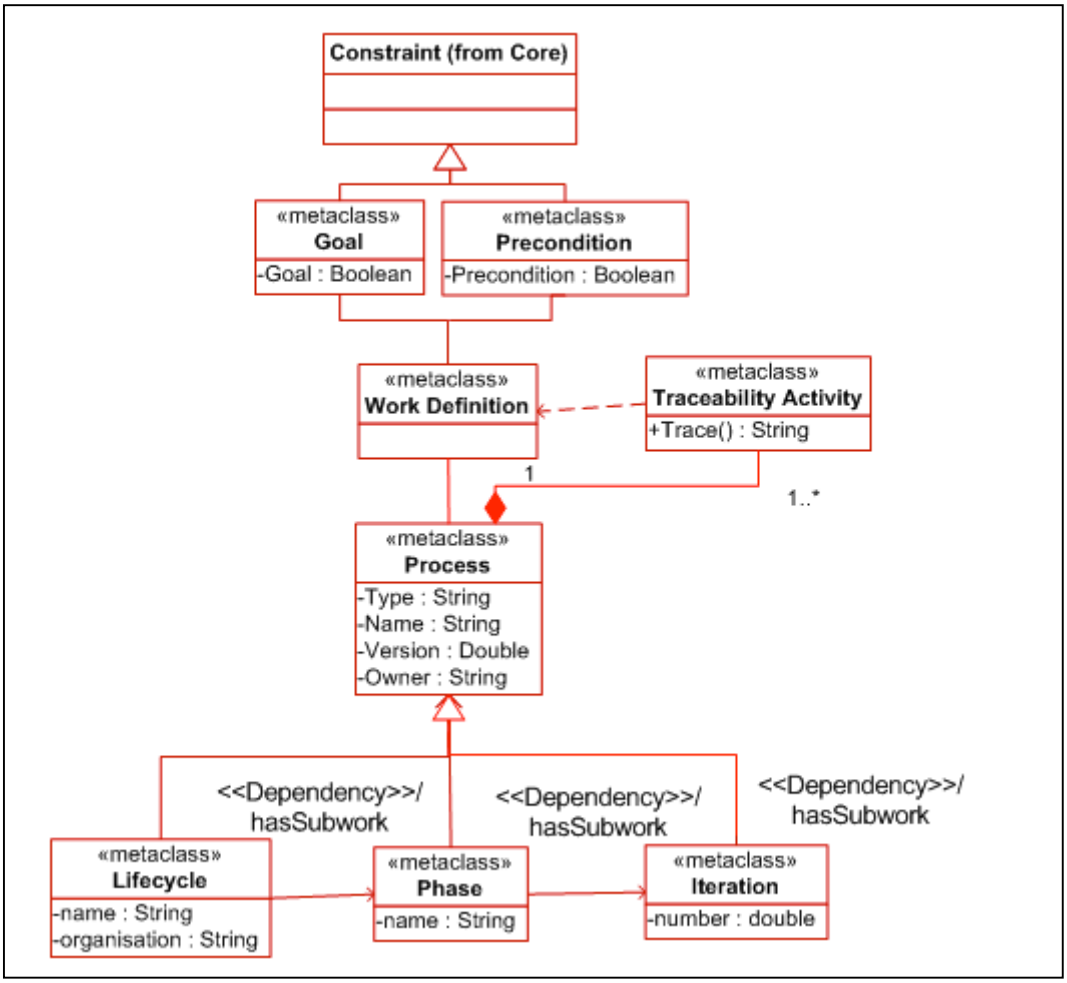



**Figure 8-11: TRAP Process Lifecycle**

The process elements describe the overall behaviour of the performing process, and are used to assist with planning, executing, and monitoring the process. As we stated earlier, a process can be seen as collaboration between roles to achieve a certain goal or an objective. To guide its enactment, we can constrain the order in which activities are executed. Also there is a need to define the "shape" of the process over time, and its lifecycle structure in terms of phases and iterations. Note that these elements do not describe the enactment itself: they are elements of the process description that are used to help plan and execute enactments of that description. (OMG, 2005)

| Precondition & Goal | |
|---|---|
| ***Description*** | ***Attributes*** |
| For each Traceability Activity there are an associated *Precondition* and a *Goal*. Preconditions and Goals are Constraints (from Core), where the constraint is expressed in the form of a Boolean Expression. The condition is expressed in terms of the state of the traceability items before and after the traceability | 1. **precondition: boolean** The precondition that must be true or false before the traceability activity can take place. For example, 'precondition=true'.<br><br>2. **goal: boolean** The goal is to execute a trace before the goal is achieved, for example, |

| | |
|---|---|
| activity.<br><br>If a *Traceability Activity* called *Trace* has input parameters Traceability Items and Artefacts and output parameter *Trace_to*, then a Precondition can have the form Traceability Item in state Approved and Artefacts in state Approved and a Goal Trace_to in state Executed. | 'goal=executed'. |
| ***Associations***<br><br>▪Goal and Precondition inherit from Constraint (from Core)<br>Traceability activities can only take place when a certain precondition is defined in order to achieve a certain goal | |

| ***Process*** | |
|---|---|
| ***Description*** | ***Attributes*** |
| A Process is a Process Component intended to stand alone as a complete, end-to-end process. It is distinguished from normal process components by the fact that it is not intended to be composed with other components. In a tooling context, the instance of Process is the "root" of the process model, from which a tool can start to compute the transitive closure of an entire process. (OMG, 2005) | ▪ **type: string***:* Is the process agile, iterative or incremental. For example, 'type=agile'<br>▪ **name: string:** The name of the process, for example, 'process= OSS-R2 process'<br>▪ **version: double.** The version of the process, for example, 'version=2'.<br>▪ **owner: string.** The name of the process owner, for example, 'owner=OSS-RC PDU' |
| ***Associations***<br><br>Associated to Phase, Lifecycle and Iteration. | |

| ***Phase*** | |
|---|---|
| ***Description*** | ***Attributes*** |
| A *Phase* is a specialization of Work Definition such that its precondition defines the phase entry criteria and its goal (often called a "milestone") defines the phase exit criteria. Phases are defined with the additional constraint of sequentiality; that is, their enactments are executed with a series of milestone dates spread over time and often assume minimal (or no) overlap of their activities in time.(OMG, 2005) | ▪ ***name: string.*** The name of the Phase, for example, 'phase= prestudy' |

| | |
|---|---|
| ***Associations***<br><br>Associated to Process, Lifecycle and Iteration. | |

| ***Lifecycle*** | |
|---|---|
| ***Description*** | ***Attributes*** |
| A process Lifecycle is defined as a sequence of Phases that achieve a specific goal. It defines the behavior of a complete process to be enacted in a given project or program.(OMG, 2005) | ▪ ***name: string.*** The name of the Lifecycle, for example, 'phase= Product Development Lifecycle.'<br>▪ ***organisation: string*** The name of the organisation that described this lifecycle, for example, 'lifecycle=OSS-RC' |
| ***Associations***<br><br>A Lifecycle is associated with a sequence of Phases. A Lifecycle is associated with one or more Processes via the association that associates a Lifecycle (describing the behavior of the process) with a Process (that packages up all of the descriptive material contained in the process). | |

| ***Iteration*** | |
|---|---|
| ***Description*** | ***Attributes*** |
| An Iteration is a number of Activities with a minor milestone or a single execution of a number of Activities. | ▪ **iteration: double** The iteration, for example, 'iteration=1' |
| ***Associations***<br><br>There are a number of iterations in a Phase while a Process consists of Phases. | |



### 8.7 LAYER M1 & M0

As previously discussed M1 is an instantiation of the M2 layer, while the M0 is an instantiation of the M1 layer. In software process terms, the M1 layer is an overall process from an end to end perspective while the M0 is the actual project enactment. As shown in the last section a traceability process may be described in the context of an underpinning metamodel, but describing all the process components in terms of metamodels are usually difficult to all roles involved in the process is difficult. For example, it is difficult to devise a way to capture all aspects of traceability in a number of models which are understandable to all parties involved. Table 8-5 below, Icons from M1 Models, illustrates a list of icons that we created to simplify M1 and M0 modelling, in order to make the models easier to

communicate to all the roles involved in the development lifecycle. These icons simply provide a notation that all parties involved in the process can understand.

| Icon | Brief Description |
|---|---|
|  | Activity |
|  | Process Role |
|  | Work Definition |
|  | Traceability Item |
|  | Phase |
|  | Process Package |
|  | Guidance |

**Table 8-5 Icons for M1 Model**

In what follows we illustrate the M1 Layer, as a structured sequence of high level Workflows and Activity diagrams. Each workflow is broken into discrete steps supported by descriptions of traceability activities, roles and artifacts associated with each step. These models were created during participation workshops with some key personnel from the OSS-RC product development team. It is important to note that we only illustrate a subset of the Layer M1 models.



### 8.7.1 M1 Business Unit Workflow

A business unit is a subset of an organisation that is independent with regard to one or more operational or accounting functions. A Business Unit has a standard set of business processes defined by Corporate Methods & Tools. In Figure 8-13 below, *Business Unit Workflow Diagram,* we represent the business unit with the ‘  ’ symbol. Below we illustrate an M1 Workflow diagram which can be adapted by each product development unit. The Product Managers and Stakeholders specify the functionality usually in Use Cases or in Requirement Specifications. Once the functionality has been specified requirement prioritization activities take place. These include prioitisation from the Stakeholders, Product and Project (Systems Architects). These workshops are usually coordinated by the Requirement Engineer. The Systems Architects distribute the functionality across the different nodes and projects. The System Testers create the System test Cases from the Specification of Functionality and Distribution of Functionalities, which is carried out in the Traceability Tool. The Configuration Management control all changes to the requirements and trace links.

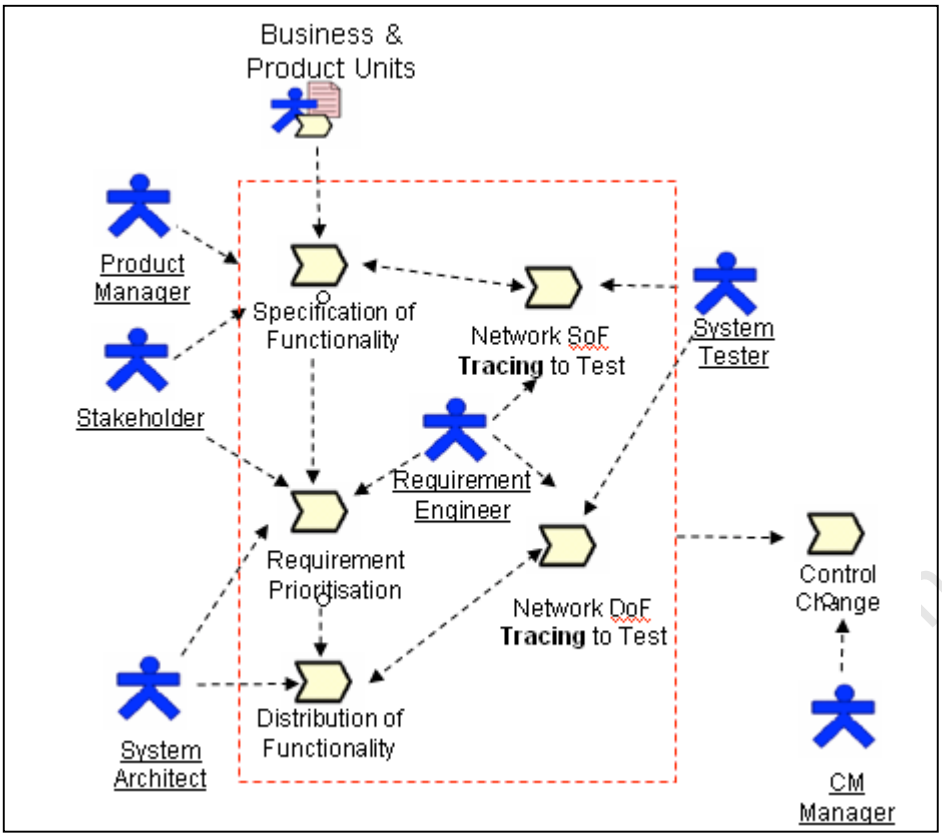



**Figure 8-12: Business Unit Workflow Diagram**

### 8.7.2 Business Unit Level M0 (Enactment)

In Figure 8-14, *M0 Enactment of Business Unit*, is the M0 model which in Ericsson is represented using web based workflow diagrams. The below diagram was abstracted from OSS-RC R2 project web-page, which could also be represented using object diagrams. We illustrated this workflow already in Chapter 4. This M0 model is taken from a real-life project. The reader should note that the objective of showing the M0 models is to illustrate that it is possible to use different layers to respresent different perspectives.

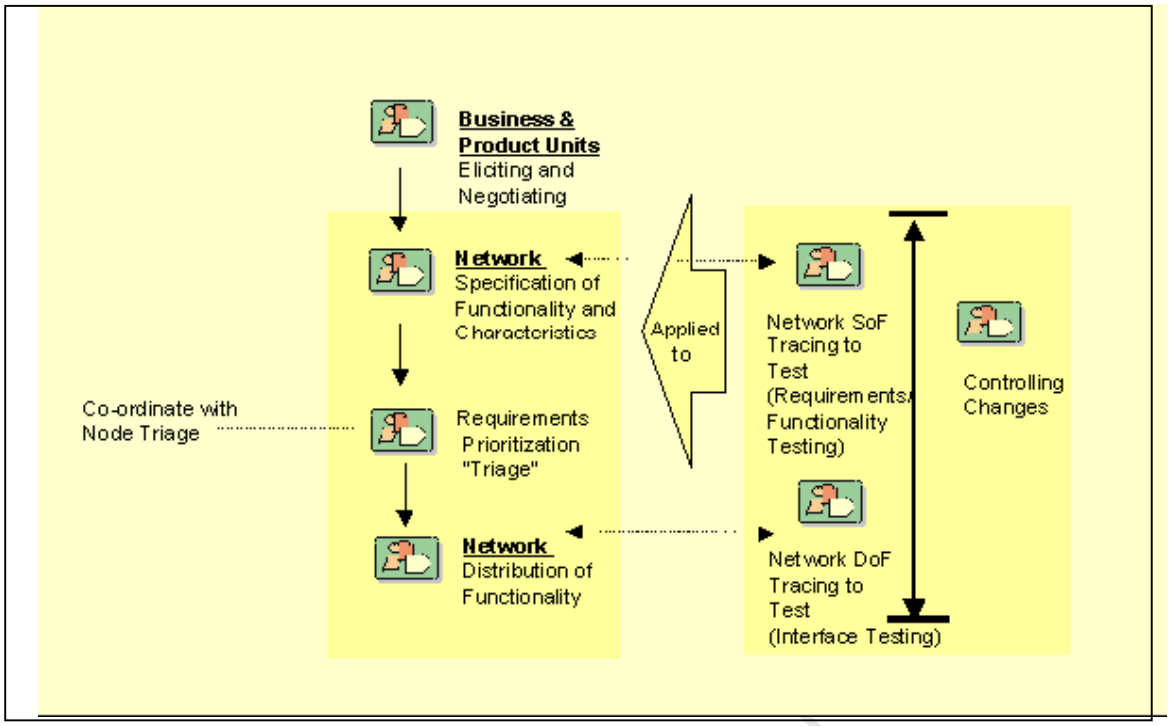



**Figure 8-13: M0 Enactment of Business Unit**

### 8.7.3 Activity Diagram (layer M1)

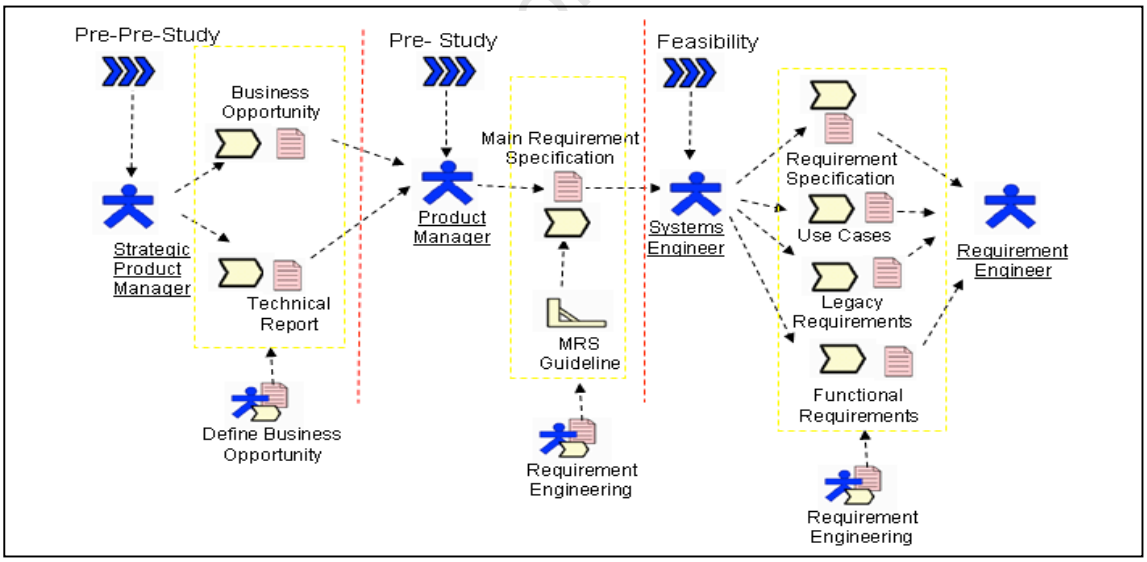



**Figure 8-14: Activity Diagram**

The Ericsson organisation has three Systems Engineering Phases; Pre-Pre-Study, Pre-Study and Feasibility Study. The Strategic Product Manager defines a business opportunity and creates a technical report of new features to the product. The Product Manager defines the Main Requirement Specification Document, using an MRS Guideline. The systems engineers creates the requirement specification documents, use cases, updates

the legacy requirements and describes extra functionality in the functional requirements documents. The requirement engineer ensures that the all traceability items are placed in a tool and that the mandatory traceability links are applied.

### 8.7.4 M1 A Project Activity Diagram

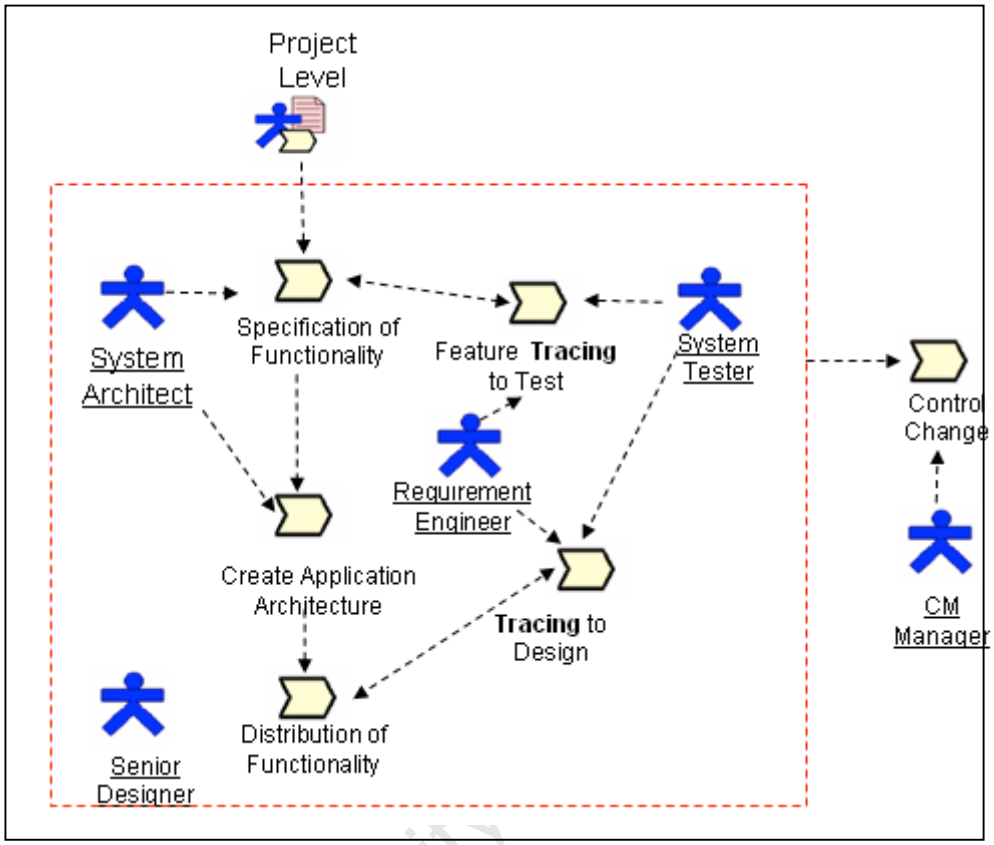



**Figure 8-15: Activity Diagram**

In Figure 8-14 above, *Activity Diagram*, the Systems Architect creates a Specification of Functionality. (SoF) The SOF is the collection of Use-Cases, Supplementary Specifications and Use-Case Models which describe the functional and non-functional requirements of the system, at the network or node level. The next stage is to create an application architecture in the form of class structure diagrams. The Senior Designer distributes the functionality across the different classes using sequence diagrams. With the assistance of the requirement engineer the system tester traces the systems test cases to the different artifacts produced by the senior designer and systems architect.

### 8.7.5 M0 Project Enactment

In Figure 8-17 below, *Project Enactment*, we illustrate the enactment of the M1 layer model that was carried out by the OSS-RC R2 project. This is simply a copy taken from the web based process description.

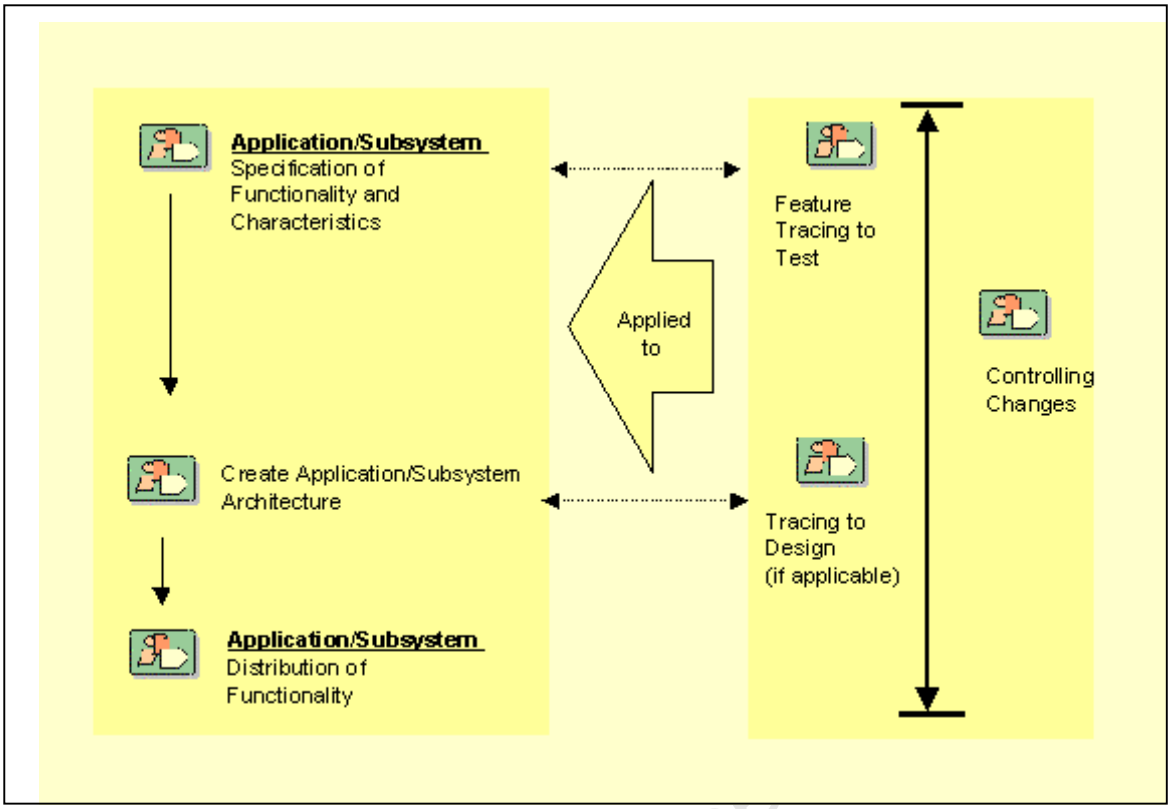



**Figure 8-16: Project Enactment**

## 8.8 OBSERVATIONS AND EXPERIENCES FROM MODELING EFFORTS

Implementing the TRAP and modeling the layers at different levels of abstraction we found that our approach expedited our understanding of the complexity involved in traceability. Moreover, we discovered that the models assisted greatly in communicating traceability concepts to others involved in the implementation of the practice. The complexities of traceability concepts are simplified as all the relevant information is stored in a knowledge base in a consistent way. The models prove versatile enough for expressing complex concepts, but we found that over time the models also became complex losing their overall objective of simplification and understandability. We did however find that each layer provided advantages from different perspectives.

1. *Layer M2:* From a strategic organizational perspective these models provided a solid foundation to get the key roles involved in any strategic definition of traceability. However, we did not gain access to any corporate personnel therefore the models were not used for their original objectives.

2. *Layer M1*: The process models and the notation we used proved very successful in describing the process providing a better overall understanding of the processes that impacted traceability. The models that are presented here were developed during modelling workshops with members of the OSS-RC process team and requirement manager.

3. *Layer M0:* The enactment diagrams presented reflect the practices of the OSS-RC R2 projects.

We did encounter a number of problems by modelling the process. The software industry has not mandated, nor enforced, a standard for the development of software

processes. There are some commercially available software processes that have MOF based metamodels as their backbone. For example, IBM's, UP is an extension of the MOF based metamodel SPEM.

The SPEM specification version 1.1 was published in January 2005 and provided a complete Meta-Object Facility (MOF) based metamodel, facilitating exchange with both UML tools and MOF-based tools and repositories. However, the specification is compliant to MOF 1.4 (not the newer MOF 2.0) and hence the metamodel lacks precise semantics. We therefore had to reference an earlier version of MOF in the development of our models, which makes our TRAP framework non-compliant with UML 2.0 and MOF 2.0. Therefore a new SPEM metamodel based on MOF 2.0 would give us more precise semantics and would help us to develop models compliant with the underlying OMG specifications.

The reality is that SPEM is the best of breed, process metamodels which overall is well designed to model basic process definition elements. However, some of the definitions and descriptions are ambiguous which forced us to interpret the metamodel more than we should for any scientific project. In some cases we found reasoning on the SPEM difficult. One of the other problems we encountered was the poor definition of an OCL (Object Constraint Language) rules to accompany the SPEM. An OCL allows certain constraints to be defined that cannot be expressed by diagrammatic notation alone. The SPEM lacked enough supporting OCL language so it was difficult to add constraints to our model.

Applying process analysis techniques to such models was another problem. While the appearance of UML as a modelling language has been a quantum leap for building higher quality process models, the truth of the matter is that a modelling language can only give us syntax and semantics to work with, but cannot tell us whether or not a 'good' process model has been produced. Naturally, even when a language is mastered there is no guarantee that the models produced are correct, complete and useful they are rigidly tested in the domain that they were applied. It is like writing a story in a natural language: the language is merely a tool that the author has to master. It is still up to the author to write a good story. There are no generally accepted guidelines for evaluating the quality of process models, and little agreement, even among experts, as to what makes a process model 'good'. As a result, the quality of process models produced in practice is almost entirely dependent on the competence of the modeller.

During the early development of the prototype stage of the development of TRAP we used an expert modeller to assess the models at the different layers. This modelling expertise ensured that the models were in fact compliant with best practice modelling approaches. We also discuss in Chapter 11, that we used the ISO 15504 assessment framework to assess the TRAP. However, for process modelling to progress from a 'craft' to an engineering discipline, the desirable qualities of process models need to be made explicit in a standard that is easy to use and understand by developers of software processes. In this case we strongly believe that the SPEM has failed. Even with a strong process engineering background it took approximately three weeks to understand the core principles in SPEM, for example the package structure, the relationship to MOF, the process elements were not clear and the process structure diagrams were difficult to understand. The lack of examples especially demonstrating how to describe the M1 and M0 was very frustrating and we were forced to make interpretations which proved difficult. We generated the different models over a 3 year period spending at least four months building the TRAP models.

▪ Many organisations simply do not have the time or the resources to build a process framework. During the case study between 2004 and 2007 the OSS-RC employed a process modelling expert whose primary objective was to model an overall framework for the OSS-RC product development unit. He used Rational/IBM's RMC process modelling framework in his endeavour. During a number of meetings we gained an understanding of the modellers approach and gained an insight into the reasons for modeling the processes with Ericsson. One of the primary reasons was to gain a better CMMI rating, because CMMI mandates the production of process models. We were impressed with the capabilities of RMC to show the relationships between artifacts.

In Table 8-6, *Review of Objectives*, we take a brief look at the objectives:

| Objective | Comment | Achieved |
|---|---|---|
| Enable effective communications regarding traceability. | During the modelling stages many sessions commenced with a demonstration by the author of the different models. As our understanding of the different models and their abstractions improved our ability to communicate the main concepts and process elements improved. We did have difficulty in gaining a common acceptance of the M2 models, receiving some negative comments from personnel not familiar with the basics of UML diagrams. | Achieved. |
| Facilitate reuse of effective traceability practices | The models were developed between 2004 and 2005 and were used until the end of the case study in 2007. We did effectively reuse the same process components during the transition between projects, however further testing on reuse needs to be undertaken in different industrial situations than the OSS-RC domain. | Partially Achieved |
| Support evolution of the process for future research and developments by the user community. | While the process models were used not used by any further research project we firmly believe that our approach and models form the foundation for future research. Therefore before this objective is achieved further experiments and tests need to be undertaken. | Partially achieved. |

Table 8-6: Review of Objectives



## 8.9 CONCLUSION

In companies for which traceability has strategic significance, a software process is successful if it contributes to the achievement of providing a framework that helps to achieve the organization's strategic goal. Engineering an effective process involves balancing the forces among the overall development strategy, the technologies being used, and the process.

In this study we aimed at understanding the many faceted problem of decomposing a software process models into reusable and easily pluggable components that can be used by any organisation to define a traceability process. Our original modelling approach, motivated by the results from our Case Study and Survey illustrated the lack of a process standard as one of the problems that needs to be overcome. In addition the guidance of the SPEM standard, specifying the creation of large components using the structure of existing process frameworks seemed to yield a manageable approach in our endeavour. Thus, this is a first step at looking at using a process metamodel approach for defining such a process. Some promising ideas have emerged, in that we have proved that using a multi-layered framework can be used by an organisation in the development of a traceability process.

However, there are many forces involved that must be taken into account. Traceability is a complex discipline which traverses the entire development lifecycle with many process attributes that are difficult to capture in a number of metamodels or models. The issue must be addressed from many viewpoints: process complexity, organizational complexity, product complexity, poor communication between success-critical roles and so on. It is very difficult to sort these problems out by simply creating a process modelling framework. The SPEM standard version 1.1., had promising attributes for process modelling but has not gained widespread acceptance, possibly due to insufficient guidance on how it should be put to use. Furthermore, there are few worthwhile examples of modelling from Layer M2 to Layer M0 that we could reference in this study. We therefore believe that while the standard did offer some excellent guidance it needs further examples and case studies before it is useable by a large group of practitioners. Continued work on SPEM version 2.0 is ongoing and hopefully succeeds better on these issues.

This work represents an initial step in modelling traceability process concepts to help understand, communicate, and promote reuse in the evolution of a traceability process.

# Chapter 9 TRACEABILITY PATTERNS: A PATTERN APPROACH TO THE FORMAL SPECIFICATION OF TRACEABILITY

## 9.1 INTRODUCTION

During the literature review stage we observed trends or common recurring themes appearing in the literature. For example, support of recording rationale for traceability links, traceability between artefacts and processes, schemes for representation the semantics of traceability links, the visualization of traceability links and empirical studies of benefits of traceability.

On analysis of our empirical data problems recurred in many small, medium and large organisations. For example, the lack of a common terminology, communication problems, lack of reuse of successful practices, lack of training on all aspects of traceability and poor definition of traceability in the underlying development processes. While some progress has been made on automating traceability, the fact still remains today that a large aspect of implementing traceability is dependent on the knowledge of the team members. Egyed states that: "Traceability is an important means to facilitate communication among the success-critical stakeholders, to preserve knowledge and dependencies created during the design process, to assure quality, and to prevent misunderstandings." (Egyed and Grunbacher, 2002)

Furthermore, fundamental to any software engineering technique is the use of a common vocabulary or language to describe the core concepts. Patterns create a "shared language for communicating experience and insight" (Alexander et al., 1977) Patterns represent abstractions of empirical experience and everyday knowledge. (Lea, 1998) Patterns are a compact way to reference a set of decisions and designs while suppressing the details not relevant at a given level of abstraction. (Edwards and Howell, 1991)

During the early modelling sessions of TRAM and TRAP we observed recurring structures emerging from the model elements. These patterns consisted of metaclasses, classes or object depending on the layer of abstraction that they emerged from. We utilise a pre-defined template to capture these recurring structures. This pattern process helped us to describe traceability experiences from a number of different modelling perspectives.

Before launching into the core principles of traceability patterns we set the context by describing the history, the different types of patterns and the method we used for capturing patterns. We introduce the TRAceability Pattern Tool (TRAPT), a prototype environment that was developed for storing, discovering and creating patterns. We do not produce an exhaustive list of patterns, however we provide clear examples of each type of pattern that emerged. We conclude by describing the lessons learned and future research possibilities.

## 9.2 PATTERN BACKGROUND

### 9.2.1 Patterns

According to the Oxford Dictionary patterns are a model or design or instructions from which a thing is to be made. (Dictionary, 2006) Patterns have been used in the many areas, from sewing to car manufacture, and in the last decade, they have become a researched and well documented topic in software engineering circles.

Patterns are used to describe best practices, good designs, and capture experience in such a way that it is possible for others to reuse this experience. (Folmer et al., 2006) Patterns are a literary form of problem solving that encapsulates the lessons learned and best practices in various aspects of software engineering. Because the patterns are named; individuals can use those names to easily refer to that experience. Furthermore, a pattern is a generic solution to a recurring problem in a given context that balances the various forces within the context. (Appleton, 2000) The structure of patterns are not themselves solutions, but they generate solutions. (Alexander, 1979)

### 9.2.2 History of Patterns

The term pattern was first used by an architect named Christopher Alexander. He observed that many of the architectural problems recurred and he reused the recurring principles to design buildings and towns. He describes patterns as a three-part rule, which expresses a relation between a certain context, problem and a solution. While Alexander's books were about actual building architectures the principles of the use of patterns are applicable to other fields. (Alexander, 1979)

In the late 1980s, the idea of patterns was being applied to software engineering and in 1987 Ward Cunningham and Kent Beck wrote a paper about the use of patterns in object-orientated programs. The paper detailed five patterns to design windows in Smalltalk. Beck and Cunningham's results were presented at conferences and workshops. Many people agreed that patterns had value but because there were so few available no one took much notice. (Appleton, 1997)Erich Gamma, who attended these conferences and workshops, realized the potential of patterns and started to think about how they could be captured and Gamma and Richard Helm came up with a number of patterns which formed the beginning of a design pattern catalogue. At the 1991 OOPSLA conference, Gamma, Helm, Ralph Johnson, and John Vlissides first got together. They became known as the Gang of Four (GoF) and in 1995, went on to publish their book *Design Patterns: Elements of Reusable Object-Oriented Software*. This book popularized patterns in the software engineering discipline. It details 23 design patterns describing the problem, context and solution as well as visually representing the relationships between different participants in the pattern. In 1993, Beck and Grady Booch sponsored a meeting in Colorado. All the researchers interested in pattern came to discuss their ideas and try to find a way to combine their ideas of objects and patterns. The people who met here became known as the Hillside Group. (Coplien, 1996)In 1994, the Hillside Group got together again to plan the first PLoP (Pattern Language of Programming) conference.

Prior to starting this research the author had used architectural patterns, during the researcher's deployment as part of the PACT (Project Acceleration Team) in the Cello-Media Gateway (C-MGW) project. These architectural patterns were used to express the organisation of the telecom system being developed. One definition of architectural patterns is to "express a fundamental structural organization schema for software systems. It

provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them". (Coplien, 1996)

### 9.2.3 Pattern Language

Coplien defines a pattern language as a structured collection of patterns that build on each other to transform needs and constraints into an architecture. (Coplien and Schmidt, 1995)All patterns explain the details of a given problem, describe the context of the problem and offer a solution. In order to do this successfully, there should be a common way of representing patterns. It is important to express a problem, context and solution so that the pattern can be understood and applied successfully in the correct situations. Patterns have become a common way to communicate experience and as a result existing patterns have been organised into catalogues. Currently, most patterns can be viewed as independent solutions to recurring problems. As developers gain more experience using patterns, they are being integrated into groups of related patterns. These integrated groups are called pattern languages. Therefore if we group all the traceability patterns together we are providing a common language for describing traceability.

### 9.2.4 Patterns and Frameworks

As discussed in Chapter 6, patterns are particularly useful in imposing structure on complex subjects, and for getting over structural concepts, command and organisational relationships and ideas about other hierarchical systems. In Figure 9-1 below, *Grouping Traceability Patterns into a Framework*, we illustrate that the patterns that emerged from the TRAP and the TRAM and from the empirical studies and literature review, collectively describe our proposed traceability solution.

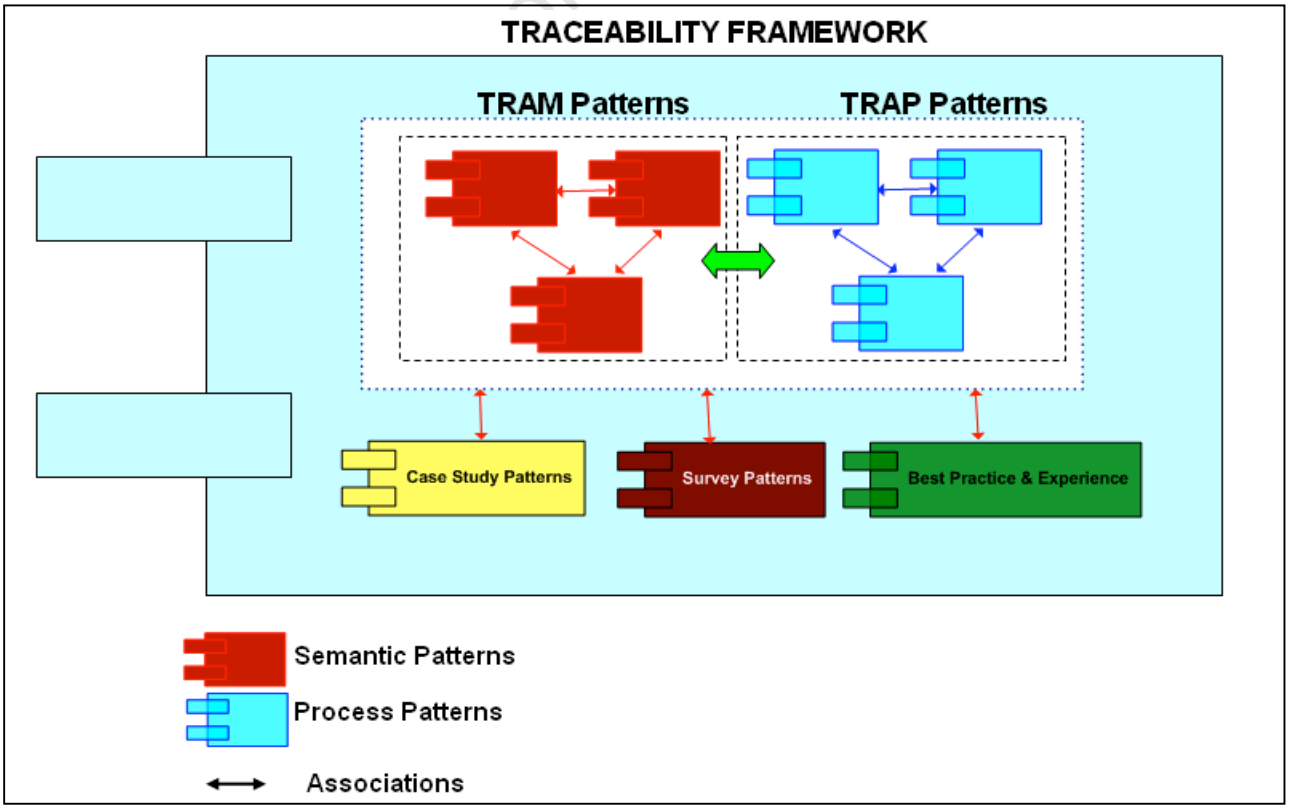
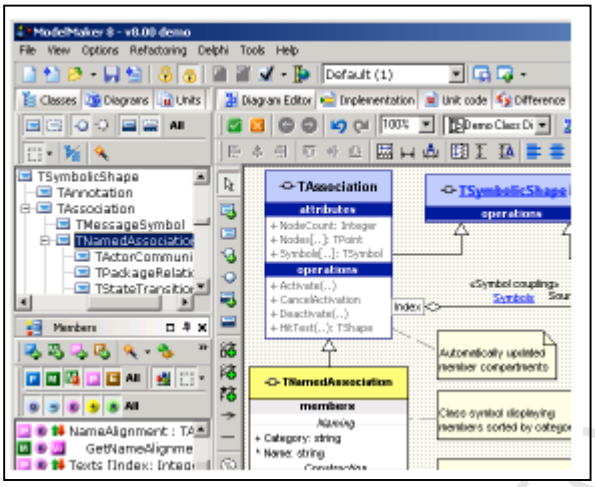



**Figure 9-1: Grouping Traceability Patterns into a Framework**

### 9.2.5 Pattern Tools

Tool support is essential for the widespread use of patterns. There are some pattern tools available. According to Mak et al, CASE tools may provide assistance to apply software patterns; validate pattern implementations; and discover pattern instances for system comprehension and documentation. (Mak et al., 2004) Many pattern developers generate programming code to illustrate their patterns. There are a few tools available to generate code from design patterns. These code generation tools often require a certain template to be followed and incorporate UML diagrams. IBM s automatic code generation tool for design patterns is described in a number of papers. (Budinsky et al., 1996)




As shown in Figure 9-2 across, *Model Maker Example*, Borland's ModelMaker Delphi CASE tool supports UML and Design Patterns. In ModelMaker patterns are part of the modelling engine, just like classes and members. You use the ModelMaker Tools API to create your own. In this tool, the design patterns are illustrated as active UML diagrams and implemented as Delphi code. When the user changes something in the code, the change is reflected in the UML. (Walker, 1997)

**Figure 9-2: Model Maker Example**

There are tools that check the validity of design pattern by verifying that the pattern conforms to certain rules. Pattern Lint is a compliance checker tool which debugs pattern implementation and design aspects like cohesion, coupling, profiling and instrumentation. (Sefika et al., 1996) Pattern-based development environments allow the construction of whole systems using patterns. They provide three abstraction levels for creating software with patterns. The first level is the pattern level. This is where you decide on the patterns that you are going to use and decide on the relationships between them. The second level, the design level, allows you to define classes, methods, associations between class and inheritance. The last level, the code level, is where the actual code is written in a particular programming language. (Buschmann et al., 1996)

As traceability patterns are a new concept we developed a tool called the TRAceability Pattern Tool (TRAPT) which is an experimental pattern creation and discovery tool. We describe TRAPT in Section 9.11

## 9.3 TRACEABILITY PATTERN OVERVIEW

"Traceability patterns generalize comparable observations of successful recurring traceability practices from different projects into a structured format that guide a solution to a traceability problem or need in a specific context"

- Justin Kelleher, ECMDA 2006

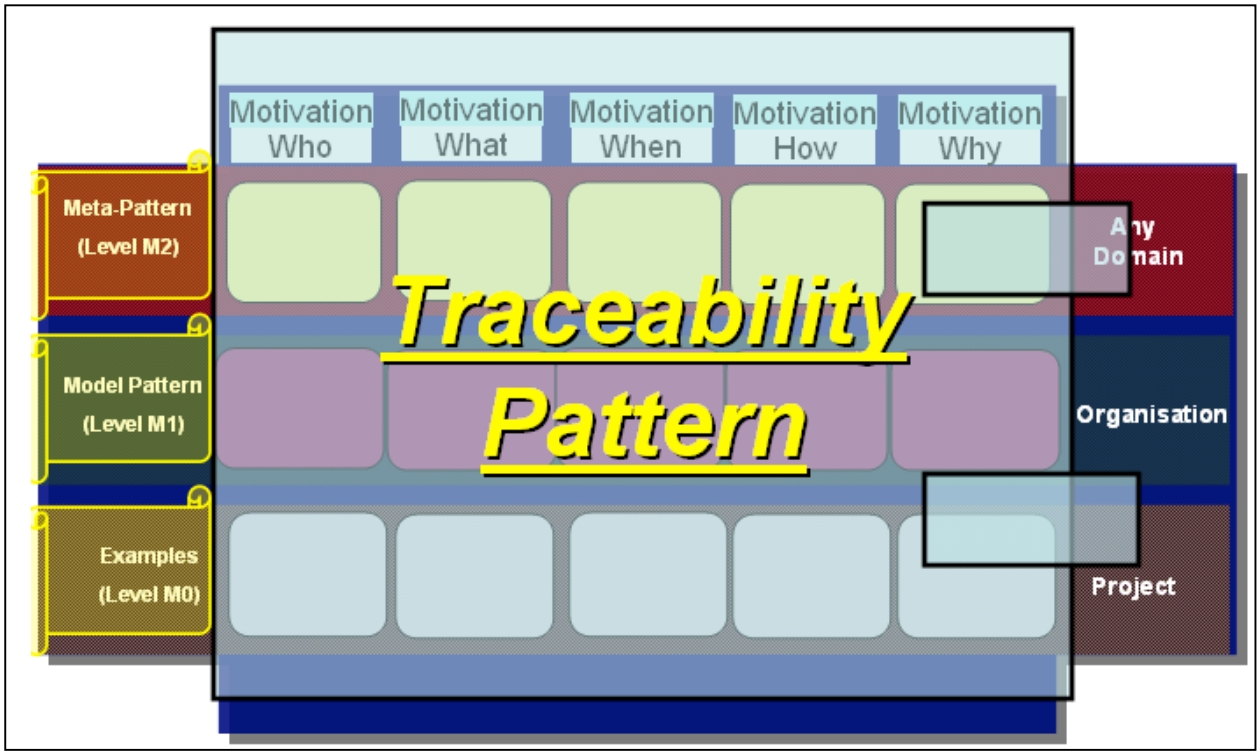



**Figure 9-3: Layers of Abstraction for Patterns**

In previous chapters we described a layered approach for abstracting models of traceability. The TRAM model describes manipulation of the traceability data while the TRAP describes the process elements that encompass a traceability process. The patterns emerge from the TRAM and the TRAP and also provide an approach for capturing traceability best practices and for representing the empirical findings. In simple terms patterns is the glue that brings together the "who, what, when, how and why" criteria into one solution space.

As can be shown in Figure 9-3, *Layers of Abstraction for Patterns*, we illustrate that patterns can be described at different levels of abstraction. For example, metapatterns describe the emerging structure from the metamodels while the instance patterns are similar to examples of traceability in real life situations. A traceability metapattern refers to a set of rules or a group of rules that describes the way of creating, using and composing the elements to establish the TRAM and TRAP. The metapatterns describe the meta-classes and meta-associations that emerge at the meta-level. In our case, a collection of traceability metapatterns were the starting point for creating the TRAM and TRAP.

It is not uncommon for practitioners to have little or no experience with traceability. Traceability patterns aim to make traceability concepts and practices understandable to

project team members in the product development cycle. The following properties give a brief introduction to the traceability patterns:

- Each pattern has a form or template that can be used in the creation of a traceability pattern.

- They provide an abstract description of a *problem* related to the implementation of traceability describing a *solution* within a *context*. In simple terms we describe the problem, the context and propose a solution to the patterns.

- Traceability patterns have a significant human component appealing to the aesthetic communication of core concepts in a comprehensive manner. Patterns help to standardize the terminology used in the traceability domain.

- They capture experience and insight which should be reused in subsequent projects. Because the patterns are named, individuals can use those names to easily refer to that experience.

- Traceability patterns don't just describe experiences and insights, but they describe deeper structures that emerge from the TRAM metamodel and model. For example traceability patterns can be used to describe hierarchical structures.

- A traceability pattern is a set of traceability types that can be instantiated to create object models. A pattern for a set of object models is created by identifying and defining the common types among these objects models.

- The overall traceability strategy can be described as a pattern language. If we combine all the patterns together we are actually describing the strategy for implementing traceability. For example, all the patterns that emerged from our work in the OSS-RC product line, is in essence a description of how to implement traceability. This pattern language is easily understood, communicated between the critical resources and can be reused for later projects. The traditional requirement management plan could be replaced by this catalogue of patterns.

- Rules on implementing traceability can be represented with a pattern with constraints encapsulated in it. For example a pattern can describe a set of constraints on how traceability items have relationships between each other. Therefore we must provide the mechanism for constraint inheritance. A traceability pattern at the meta-level refers to a set of rules on how to create patterns.

## 9.4 MOTIVATION& OBJECTIVES

### 9.4.1 Problems & Motivation

The following problems were identified during the empirical study:

- *Problem Poor Communication:* Between success-critical roles on matters related to traceability. The Centre of Excellence for Traceability stated that: Tracing requires communication between stakeholders, but semantic mismatches and disparate use of terminology across various stakeholder groups create communication barriers.

- *Problem Lack of knowledge:* During the case study and the survey it became clear that many software engineers do not have sufficient knowledge to practice traceability.

- *Problem Poor training:* As illustrated by the case study and the survey training, especially in times of difficult economic uncertainty, is often cut as a first step in cost cutting. Organizations need a way to create certification programs that teach skills needed by traceability practitioners.

- *Problem Process:* In order to generate and maintain quality traceability information, a good process is required; however as the empirical study shows and the Centre for Excellence for Traceability describes, "traceability is often not included as an integral part of the development lifecycle"

- Problem *Traceability is not practiced across the entire product lifecycle*. Our results show that while some roles place a high level of importance on traceability (for example project managers, requirement engineers and product managers) many in the development team do not share the same attitude on the importance of traceability.

- *Problem Reuse:* Poor reuse of traceability techniques. Many organisations do not have a formal approach for capturing successful practices and often good practices are not reused.

- *Problem Metrics:* Measuring the success or failure of traceability is important, but good traceability metrics do not exist.

### 9.4.2 Traceability Pattern Types

In Table 9-2 below, *Types of Traceability Patterns*, we introduce the different types of traceability patterns that we discuss in the remainder of this chapter.

| Type | Brief Description |
|---|---|
| *Semantic Traceability Patterns* | A semantic pattern describes how to describe and manipulate traceability data. Semantic patterns define traceability semantics, define concepts related to creating traceability data, the relationships between the data and the how to manipulate this data. |
| *Traceability Process Patterns* | Traceability process patterns are defined as a pattern which describe a proven, successful approach and/or series of action for developing traceability. |
| *Empirical Patterns* | The aim of empirical patterns is two-fold. First, to provide a synopsis of current knowledge of empirical data gathered during case studies or surveys. Secondly, to provide an overview of the development of the ideas and understanding of traceability in different contexts. |
| *Traceability Best Practice Patterns* | These patterns capture traceability best practices for implementing traceability practices. |
| *Traceability Anti-Patterns* | These patterns describe solutions that didn't work. Sometimes knowing a wrong approach up-front is useful when implementing traceability. In general they provide traceability users information of what not to do. |

**Table 9-1: Types of Traceability Patterns**

## 9.5 OUR APPROACH

### 9.5.1 Methodology

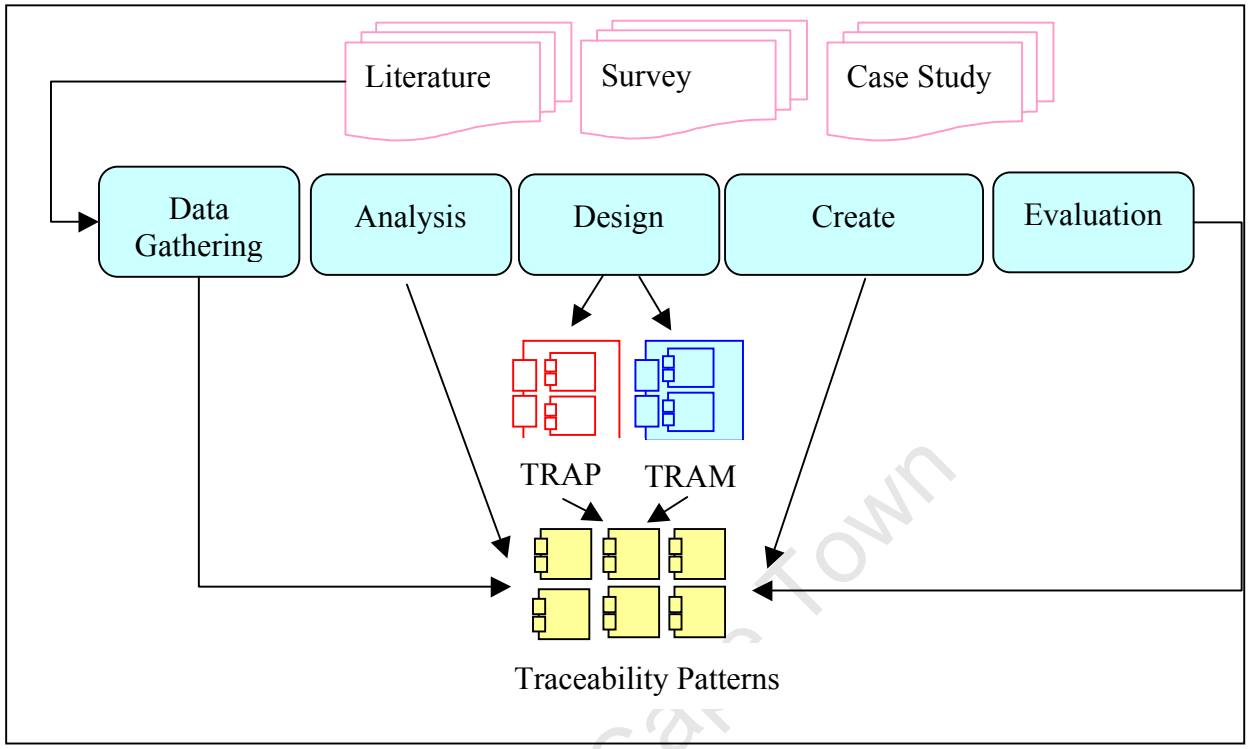



**Figure 9-4: Pattern Methodology**

As this is a new technique for describing traceability, there are no methodologies to reference, therefore we define our own. This section describes the proposed methodology and how it represents a synthesis and extension of the research described in the chapters on TRAP and TRAM. The first part of this section lists the stages that constitute the methodology.

The proposed methodology consists of five main stages, these being *information gathering*, *analysis*, *design*, *create*, and *evaluation*. Initially, information was collected from current literature before the data gathering from the case study and survey. We structured this data into a table adding codes where possible. As shown in Figure 9-5 below, *Coding Assists with the Identification of Patterns*, we code the source of the data and map this to the pattern template looking for commonalities where possible. We use a pattern template to map the data against the necessary fields. The fields of the template that are not complete were further investigated during the case study and survey.

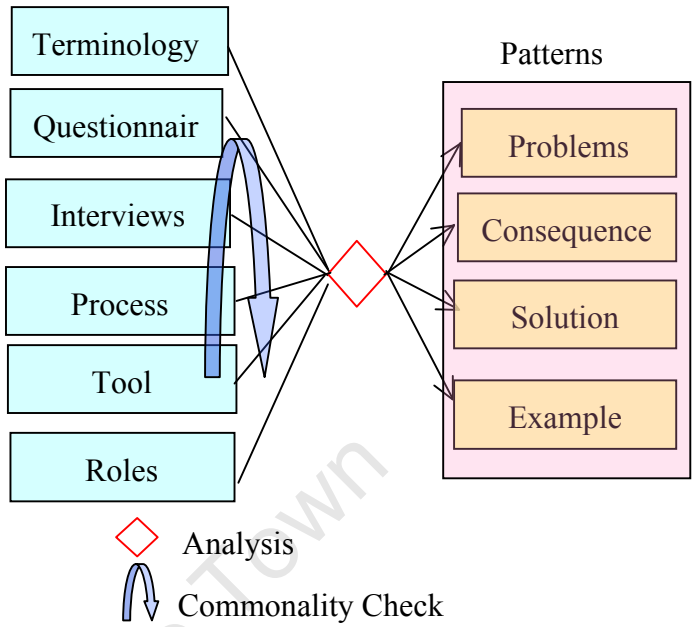
| Data Source | Coding | Approach |
|---|---|---|
| Literature | Terminology, techniques, problems, solutions |  |
| Case Study | Process, Tools, Interviews, identified problems, consequences, solutions. | |
| Survey | Process, tools, questionnaire, interviews, problems, consequences, solutions | |



**Figure 9-5: Coding Assists with the Pattern Identification**



### 9.5.2 Analysis & Design

As shown in Figure 9-6, *Data Analysis*, we analysed the terminology, the data, the tasks, the users and the usability factors that are important to the implementation of traceability. This information defines the structure and content of the traceability pattern, the tasks that the pattern needs to support, the target users, and the usability goals. The analysis phase simply highlights the relevant factors that need to be considered for each component so that the pattern creator can then try to balance the forces from each one. Terminology analysis identifies common terms and any conflicts that might exist. It was at this stage that certain terms were decided, for example, the term *traceability items* became our standard for describing any model element or item that impacted the success of the system under development.

Analysing the data helped us determine the content and form of the patterns. The data analysis was used to decide when data needs to be manipulated, what graphical structures and objects are be used, and what data attributes are to be mapped to the model elements. The source of the data can provide a useful starting point for the primary visual structures to be used in the pattern. Data items represent data objects and collections of data items form datasets. The data range refers to the set of values that a data attribute has assigned to it within a dataset. For example, the priority of certain traceability items, varied from “high, medium, low” to number values ranging from “1-highest number”. Some data structures lend themselves more readily to visual structures than others.

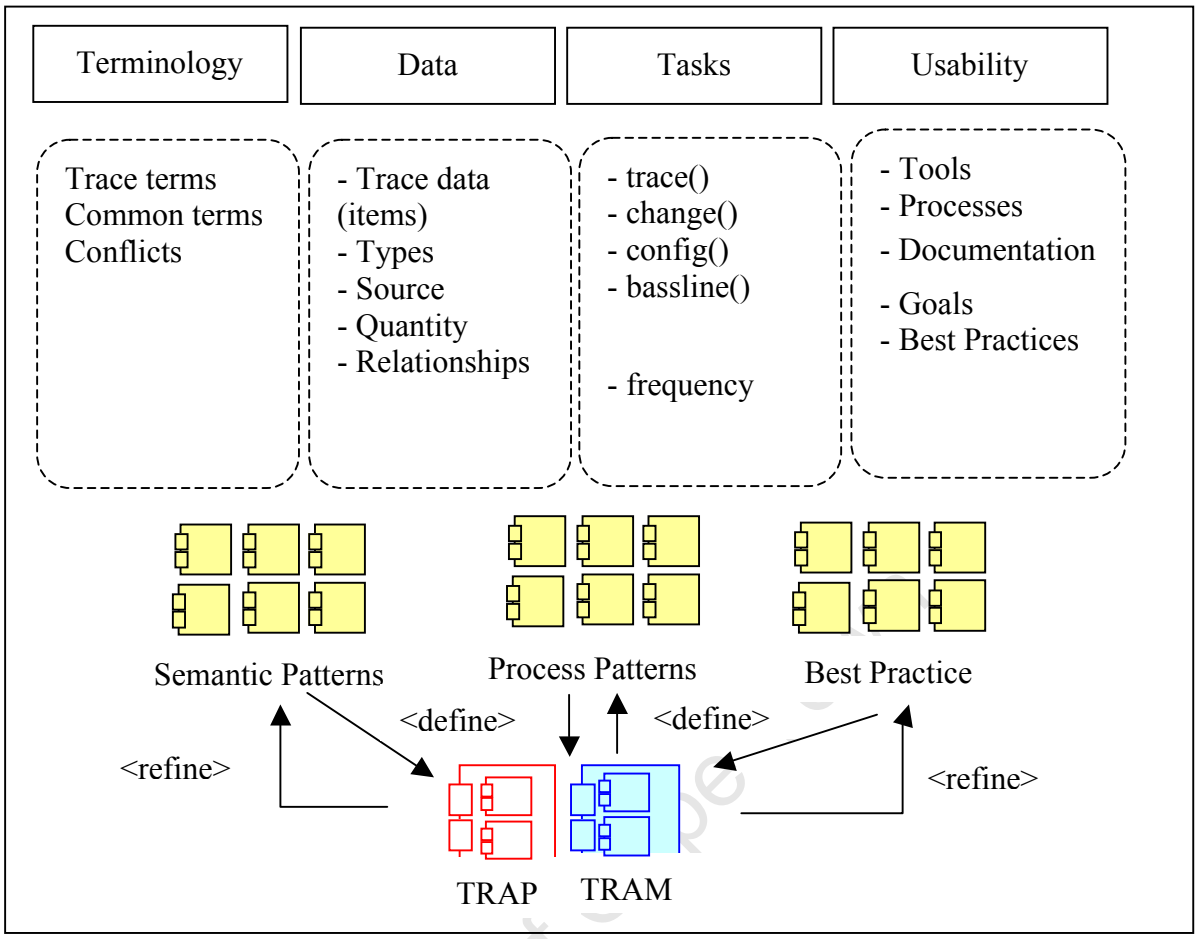



**Figure 9-6: Data Analysis**

Task analysis determines the goals the user is trying to achieve and the methods they use to achieve them. Analysis of existing approaches can help to highlight problems that should be resolved by the patterns. The pattern creator can use the identified tasks to determine both the data and the interaction mechanisms needed to support the tasks. The tasks are usually represented as operations in the different models. The results of task analysis will include the data items and specifically the data attributes relevant to the tasks. Depending on the results of the task analysis the data source and structure may or may not be relevant to solving the task. The task frequency can be used to determine which tasks, and the data attributes required to support those tasks, should be given priority when designing the pattern. Various user attributes must also be considered when designing a pattern. One of the goals of this research is to provide a software engineering role with a process that allows them to create designs that combine novelty, perceptual effectiveness, and usability. Pattern heuristics help to implement traceability but at the same time give them the freedom to be creative. Heuristics are not fixed rules that must be obeyed; they simply describe desirable characteristics.

In the pattern design stage the designer must determine an appropriate set of model elements, their interactions and composition elements. This defines the graphical aspect of the pattern. Using the data gathered during the data gathering, the researcher creates prototypes of TRAP and TRAM as specified in chapters 7 and 8. During this phase further patterns are identified that emerge from the models being created.

As would be expected we encountered a number of problems when identifying and creating the traceability patterns. For example, how to integrate data and process model elements into one pattern; which are described in two separate models. We used a number of modelling experts for sanity checks and for gaining clarity on certain issues that arose. In some cases the identification of the patterns acted as a catalyst for discussion with the OSS-RC Requirement Manager.

### 9.5.3 Created & Evaluate

Once the pattern creator has analysed and designed all the components they can then start to complete the pattern. The implementation of the patterns is the activities that take place after the pattern has been designed. Since the development of the traceability pattern is iterative the evaluation may lead to information gathering, analysis or design stages being revisited

Evaluating of the patterns to see if it is capable of supporting the user in their tasks and meet all of the desired usability criteria. Ideally we would like to empirically evaluate all proposed patterns and the underlying process. To do this correctly would require a complete development team including managers, traceability experts, software engineers, process specialists, and users. Even if this were possible, we could only compare the relative merits of each pattern based on the measures taken. This may be useful but it is not the purpose of this research to determine whether or not one pattern is better than another. Instead, we are proposing that traceability patterns are an effective and efficient approach for capturing common recurring practices, for promoting communication and reuse, for describing semantics and process aspects of traceability and therefore can be used as an aid for capturing knowledge and experience which can be used for better training experiences.

However, we develop a number of usability evaluation techniques including cognitive walkthroughs, focus groups and usability inspections, and so on. A Cognitive walkthrough method is a usability inspection method used to identify usability issues in a piece of software or web site, focusing on how easy it is for new users to accomplish tasks with the system. The decision regarding which technique to use is a difficult one that depends on many factors including time, type of user, and the usability factors you are most interested in achieving. One method of evaluating patterns is to test each one with users under controlled conditions. The pattern then undergoes statistical analysis in order to answer usability questions such as which patterns took the least time to learn, which patterns the users preferred and so on. This depends of course on the context that the patterns are set. We will revisit the evaluation of the patterns in Chapter 11.



## 9.6 PATTERN TEMPLATE

### 9.6.1 Pattern Template

Patterns are defined using pattern templates called forms. Early on into our investigations we created a pre-defined template. A pattern is defined by specifying the values of the form fields for that particular pattern. In his books on patterns Alexander offered a pattern form from which he specified patterns.(Alexander et al., 1977)

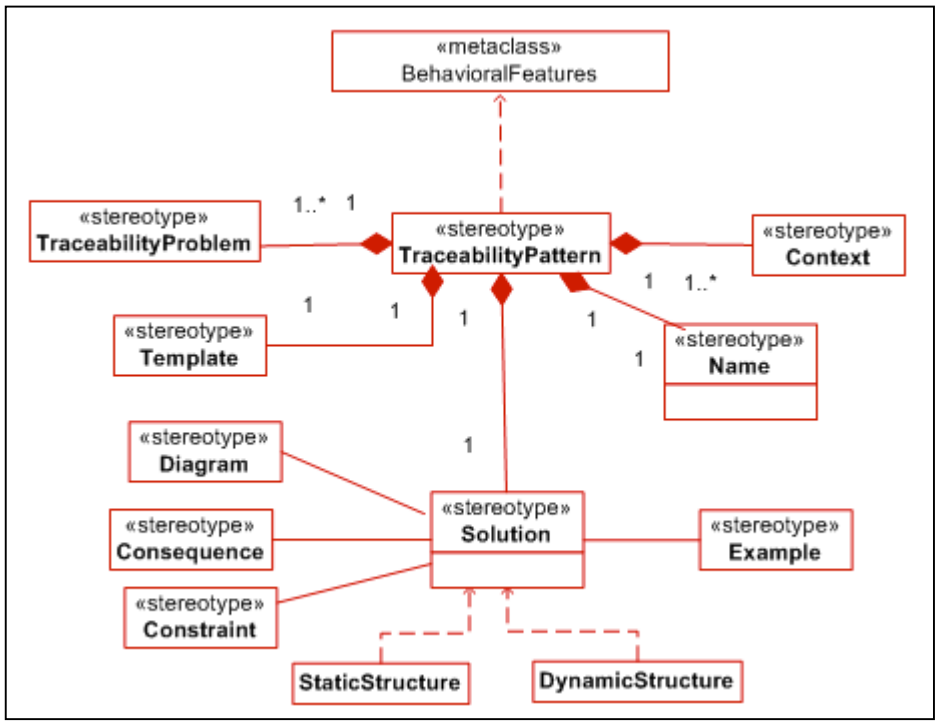



**Figure 9-7: Model of Template**

As shown in Figure 9-7, *Model of Template*, the UML 2.0 infrastructure specification defines behaviour as an observable effect of an operation or event, including its results. A behavioural feature is a dynamic feature of a model element, such as an operation or method. In UML 2.0, the *Behavioral Features* sub-package of the *Abstractions* package specifies the basic classes for modelling dynamic features of model elements. Behavioural patterns are concerned with the assignment of responsibilities between objects, or, encapsulating behaviour in an object and delegating requests to it.

In Figure 9-7 we illustrate a model of the template that we used. Patterns are defined using pattern templates called forms. A pattern is defined by specifying the values of the form fields for that particular pattern. The Static Structure is a diagram that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships. The dynamic behavior aspect of any pattern describes the interactions, sequences and behavior of the pattern's solution. While these elements are not necessary here we highlight them on the pattern template model to illustrate that a traceability pattern has both a static and dynamic aspect to them. In Table 9-3, *Pattern Template Fields*, we describe each field of the patterns.

| Pattern Heading | Brief Description |
|---|---|
| **Name** | Each pattern must have a meaningful name, which is easily explained and understandable. This helps with pattern classification or categorization as well as improving communication between different roles. Because the patterns are named, individuals can use those names to easily refer to that experience. |
| **Problem** | The problem describes the traceability problem that the pattern aims to solve and the conditions that must be |

| | |
|---|---|
| | met before it makes sense to apply the pattern. |
| **Context** | The context describes the preconditions under which the problem and its solution recur and the patterns applicability. |
| **Solution** | The proposed solution in text form. |
| **Diagram** | Christopher Alexander maintained that the sketch is the essence of the pattern. Coplien (Coplien, 1996) suggests that more specific graphical representations are by definition better while Herman states that the use of conventional UML diagrams leads to over specification and a consequent loss of the abstract nature of patterns. (Herrmann et al., 2003)<br><br>The diagram should contain the static traceability structure being described, the participants who use the pattern and their collaborations to illustrate how they solve the traceability problem. It should add clarity to the prose. |
| **Consequence** | A description of the consequences might include an acknowledgment of the trade-offs involved in selecting a particular traceability pattern. They are used to evaluate the patterns |
| **Constraint** | A constraint is a semantic condition or restriction. The rules thus specify constraints over attributes and associations which are defined in the metamodel. |
| **Example** | The implementation describes an example of the pattern in a traceability environment, for example how to set up requirement types and document types in Rational RequisitePro. In general, the implementation section is considered a non-normative suggestion, not an immutable rule or requirement. |

**Table 9-2 Pattern Template Fields**



## 9.7 TRACEABILITY PATTERNS

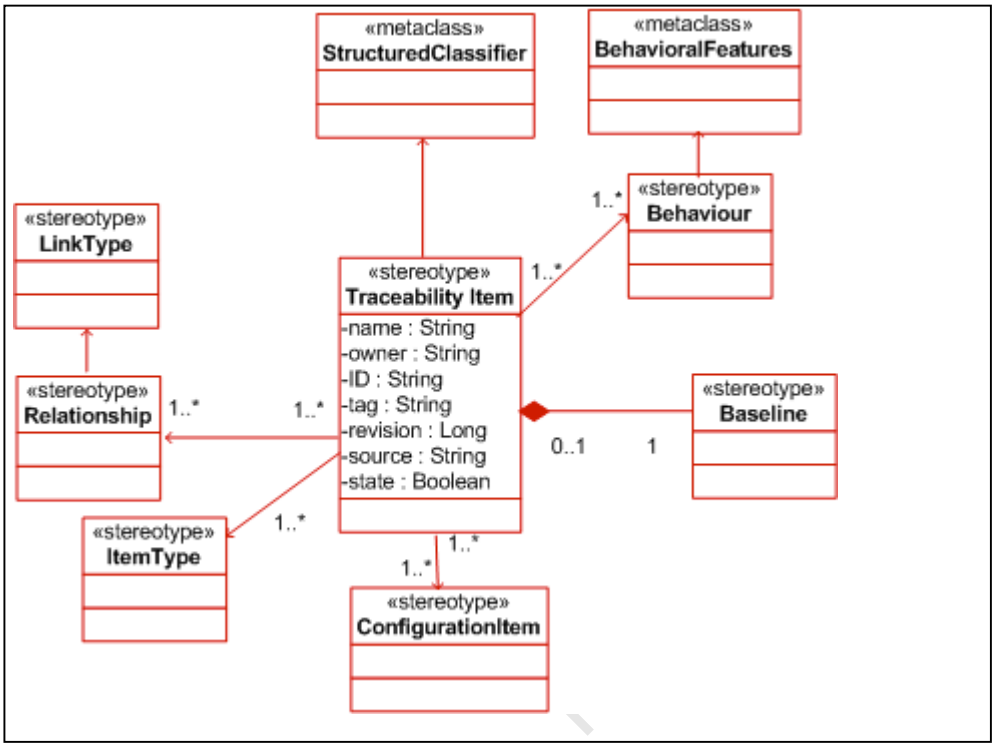
### 9.7.1 Semantic Patterns

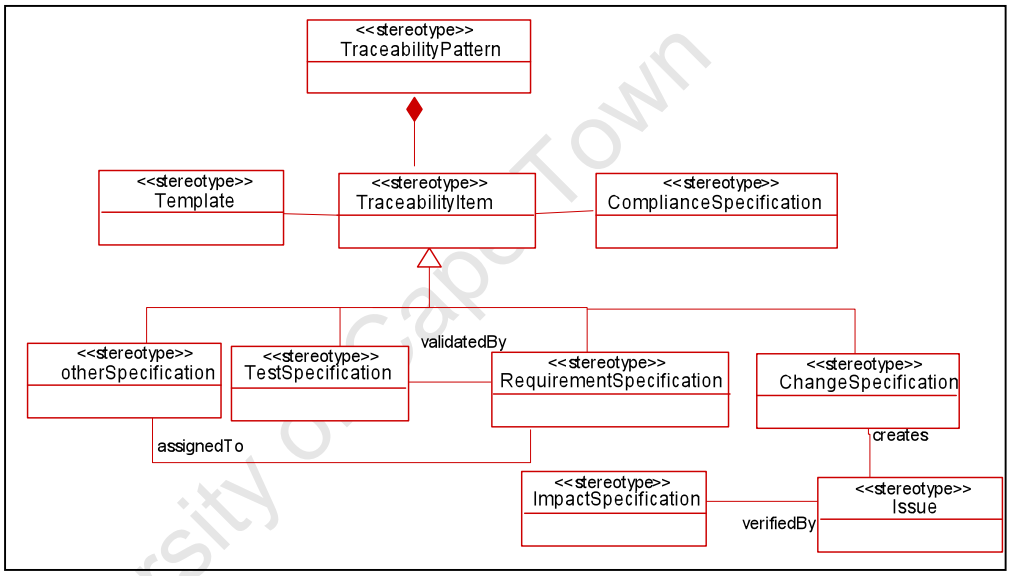
The TRAM describes the *traceability data concepts* and the *relationship between the data*.

A semantic pattern describes how to describe and manipulate traceability data. Using a model as an example adds a higher layer of abstraction for describing the concept. Semantic patterns are a means to communicate knowledge at an epistemological level of representation. Semantic patterns are used by software engineers to communicate, document and explore traceability alternatives by using a common vocabulary. They also

decrease the complexity of implementing and understanding traceability data and its manipulation. Additionally, semantic patterns offer solutions to common problems, help a novice to "act" more like an expert. In Figure 9-8, *Create Traceability Item*, we illustrate the a semantic pattern that emerged from the TRAM.

| Pattern Heading | Brief Description |
|---|---|
| **Name** | Create Traceability Item |
| **Problem** | Defining traceability items is one the most rudimentary steps in implementing traceability and in many cases an area where basic errors are made. In this pattern we address the creation of the pattern, the rules for creating the attributes and attempt to create a conceptual understanding of creating relationships. |
| **Context** | While this is a basic pattern which high-end users will generally not need, perhaps because of an understanding of the basic concepts of traceability or because they use a traceability tool to set the rules for creating traceability items. However, as our survey illustrated low-end users may not fully understand even the simplest concept. This pattern is therefore to be used by novice traceability practitioners |
| **Solution** | As shown in Figure 9-8, Create Traceability Item, a traceability item is any specification that can impact the system to be developed, for example a model, a diagram, a use case, a non-functional requirement, a change request, a test specification, or any other specification in the development cycle that impacts the overall system. A traceability item has a type (functional, non-functional) and relationships with other traceability items and each relationships has a type (process, product). After a traceability item is baselined, it becomes a configuration item,<br><br>Traceability items are located in word documents, spreadsheets, URLs, and traceability tools. They should be created using a pre-defined template. Traceability items have a unique name, identity, a revision, a specific type and attributes such as author, person responsible, origin or rationale, release number, status, priority, cost, difficulty, stability, and risk. All items should be under version control with each revision creates a separate item. It is possible to go back to a previous revision of an item and check the relationships to other items for that specific revision. |

| | |
|---|---|
| **Diagram** |  <br>**Figure 9-8: Create Traceability Item** |
| **Consequence** | This pattern should only be used when creating a requirement using the MAR's traceability tool. Patterns MUST have at least one relationship, have a type and belong to a baseline. |
| **Constraint** | When a traceability item is being set up it must satisfy a number of conditions. |

| | |
|---|---|
| **Example** | Configuration management is the discipline of identifying the components of an evolving system for the purpose of controlling changes to these components. Configuration management controls the configuration of a product from definition, development, build and maintenance. It is essential to integrate configuration management with traceability management for better product concurrence. A *ChangeSpecification* is a traceability item specifying a proposed change to another traceability item which is also under configuration control. Configuration control concerns the activity of controlling changes after a *Baseline* has been established. A Baseline is a version of a configuration established at a point in time when only controlled changes are allowed. A *Baseline Specification* documents the traceability items in a configuration version. The Baseline Specification documents which item belongs to which baseline. When a Change Request is received *Impact Analysis* investigation is undertaken.<br>  <br><br>Figure 9-9 Constraints on Traceability Item<br>Below is an example of modelled from two real Traceability Items found in the MAR's traceability database. Requirement RAN, a functional requirement was part of the OSS-RC R5 (Baseline V1.5) project and is linked via a product relationship link to the non-functional requirement, Requirement CNS. |

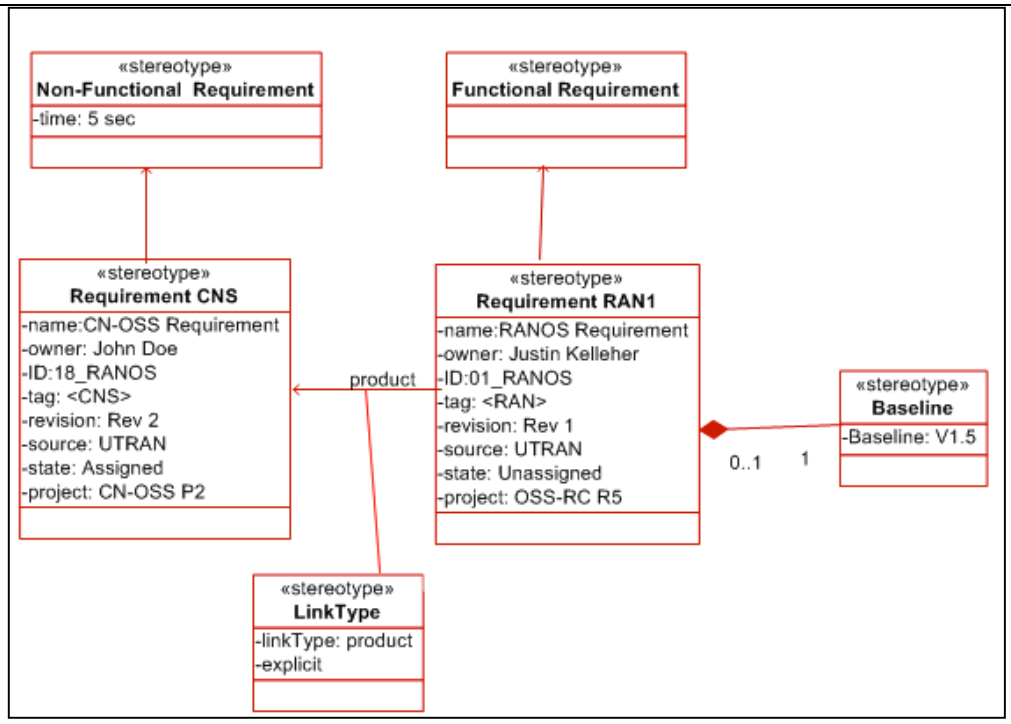



Figure 9-10: Example from OSS-RC

In this Figure 11 below, Example from Tool, we illustrate that a traceability item must have a revision, attribute and at least one traceability relationship.

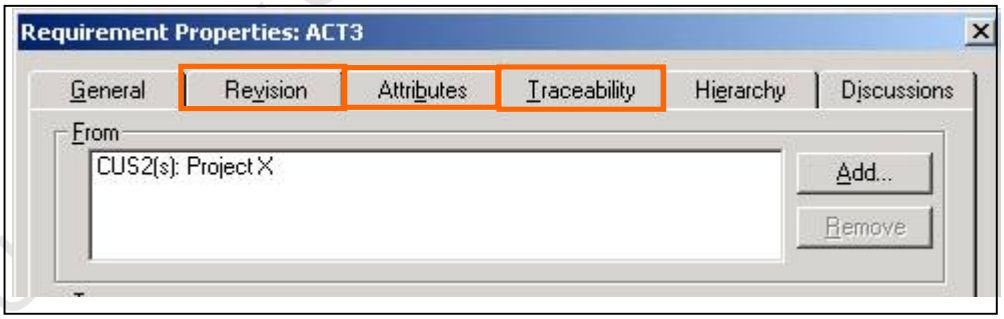



Figure 9-11: Example from Tool

### 9.7.2 Semantic Pattern from Literature

In the IT industry we often remark how similar a problem is to something we solved a decade ago on technologies long since retired. A good example of this Gotel and Finkelstein's work from the early 90s who wrote the seminal paper entitled, "An Analysis of the Requirement Traceability Problem".

In this paper they identified that many of the problems that arise in traceability were pre Requirement Specification. This paper discusses many problems that are still prevalent today. Perhaps if their paper was described by a pattern then the communication and

distribution of this basic principle is easier to understand. In the below example, we demonstrate this paper in one simple pattern.

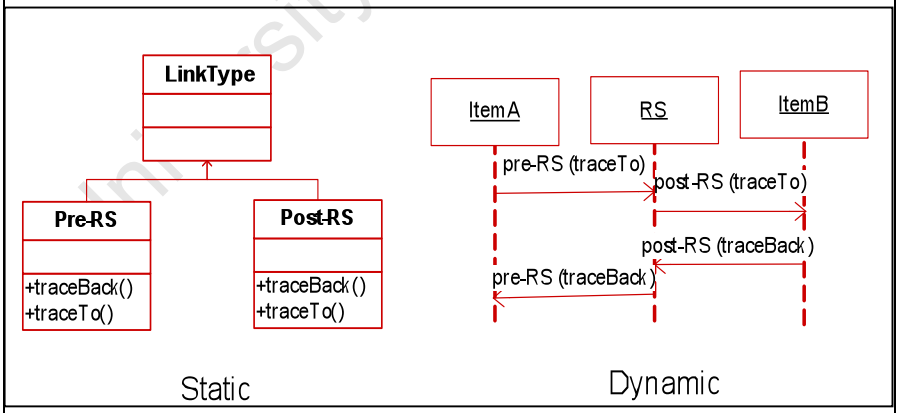
| Pattern Heading | Brief Description |
|---|---|
| **Name** | Pre-Requirement Specification Traceability |
| **Problem** | The majority of the problems attributed to poor requirements traceability are due to inadequate pre-Requirement Specification (RS) traceability. They also describe the consequences of poor pre-RS traceability and offer, but this is very hard to extrapolate from the complex literature. |
| **Context** | The reasons for the problems with poor pre-RS traceability are:<br>1. No agreement on the end-user requirements, resulting in a tendency to focus only on their immediate and visible needs.<br>2. Information (e.g., tacit knowledge), cannot always be obtained, and the quality of that which is varies. Deliverable driven cultures can discourage gathering certain information.<br>3. The documentation of required information is no guarantee of its traceability. That which is structured, so it is traceable in many ways, provides no guarantee it will be up to date.<br>This pattern highlights the importance of Pre-RS Traceability and describes the implementation of Pre-RS traceability. |
| **Solution** | Pre-RS traceability is concerned with those aspects of a requirement's life prior to its inclusion in the RS. |
| **Diagram** |  |

**Figure 9-12: Static and Dynamic Diagrams that illustrate Pre & Post RS**

The strength of this pattern is in its sketch that we created. In the Static diagram we illustrate that Pre & Post RS are associated to link types, while the dynamic diagram shows a sequence diagram of how you use pre and post RS.

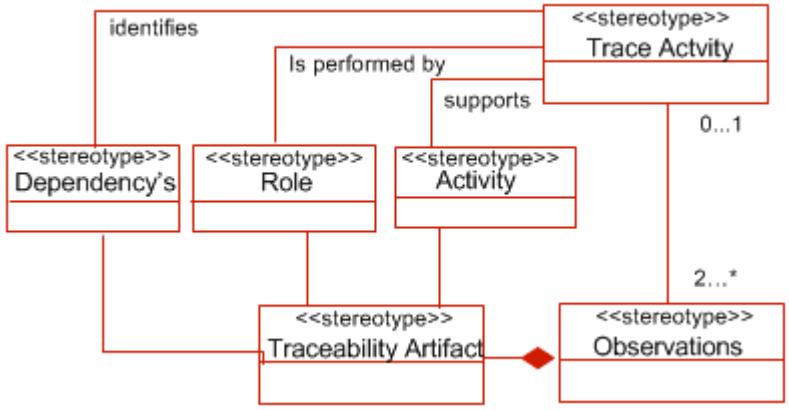
### 9.7.3 Process Patterns

Software process design is a common issue faced by many organizations. The introduction of process patterns provides an effective solution to the challenge. (Ma and Wang, 2006) Various traceability knowledge and techniques that would encompass a traceability process such as role and responsibility descriptions, tasks, scheduling, phases, resource assignment, are all actually required to build-up a detailed process for practical applications. Skilled requirement engineers or project managers have these techniques as their everyday knowledge, and they can use this knowledge for customizing the target traceability process. We believe some of traceability process technique can be explicitly clarified and categorized, so that inexperienced traceability users can easily reuse them to customize a process or even implement traceability.

Process patterns are in some ways similar with what Alexander calls generative patterns. Alexander describes generative patterns as: "These patterns in our minds are, more or less, mental images of the patterns in the world: they are abstract representations of the very morphological rules which define the patterns in the world…..But the same patterns in our minds are dynamic. They have force. They are generative. They tell us what to do; they tell us how we shall, or may, generate them; and they tell us too, that under certain circumstances, we *must* create them."(Alexander et al., 1977)

In this section, we attempt to formalize traceability process patterns. We also try to facilitate these patterns for process evolution. In this approach, process patterns are used as templates of process development.

| Pattern Heading | Brief Description |
|---|---|
| **Name** | Trace Activity |
| **Problem** | This pattern describes the process elements used to carry out traceability. It describes the role, the dependencies and the observations. |
| **Context** | You are working for a large multinational company. This company has work-groups with its members dispersed over various countries. This groups have to work together to fulfil their jobs, therefore efficient communication and information sharing is a vital necessity for them.<br><br>It is not possible that all members of a work-group meet frequently, since this would be too time consuming. To prevent information overload, the members of a work-group need an easy way to discuss the processes involved in traceability. This pattern describes at a high level, the roles, the dependencies, the source of the data, the activities that need to take place and so on. |
| **Solution** | A *Role* is either a physical project resource or tool based resource which performs a traceability *Activity*. For example a Role is a project manager, tester, maintenance engineer or a tool (RequisitePro, Doors). Traceability items are located in word documents, spreadsheets, URLs, and traceability tools. |

| | |
|---|---|
| | We call this workspace the Traceability Artefact. Traceability patterns are derived from two or more similar observations being made either from the Trace Activities or from the Traceability Matrix. |
| **Diagram** |  <br>**Figure 9-13: Process Pattern** |
| **Consequence** | Before a role applies this pattern they should understand that the have responsibilities for the following activities:<br>- Identify a traceability item in the Traceability Artefact.<br>- Perform an activity on the traceability item<br>- Identify traceability relationships with other traceability items.<br>- Make observations on the regularity or common occurrence of this activity. |
| **Constraint** | - The user needs to know where the traceability items reside in the Traceability Artefacts.<br>- It is assumed that the role understand that they have traceability responsibilities.<br>- The observations are made by the role, the process team or the requirement manager. |

### 9.7.4 Other Process Examples (in brief)

Without providing a template for each pattern in this section we briefly describe a number of other process patterns.

#### 9.7.4.1 Simple Role Pattern

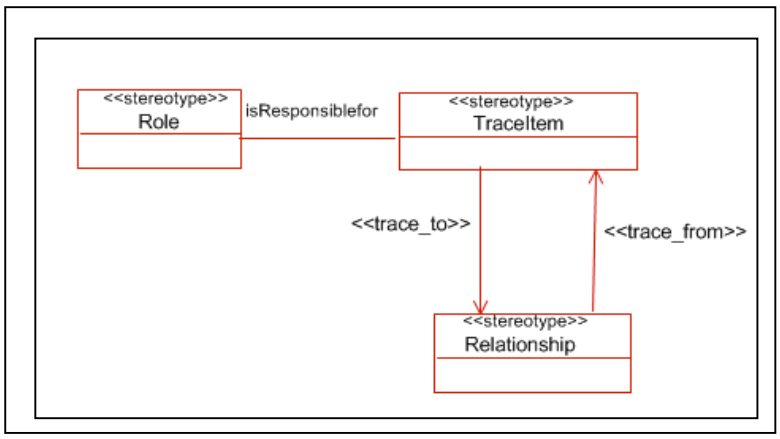



Figure 9-14, *Role Pattern*, depicts simple traceability pattern. A role is responsible for traceability items, by setting relationships ("trace_to" and "trace_from") relationships. The "Role Pattern", acts as a conceptual pattern in the reifying of activities and the traceability items and hence the creation of the traceability links.

**Figure 9-14: Role Pattern**

#### 9.7.4.2 Extracted from Ericsson Unified Requirement Engineering Process

In order to investigate if we could transfer some the process in operation in Ericsson we made a simple sketch as shown in Figure 9-15 below, *Ericsson Unified Requirement Engineering*.

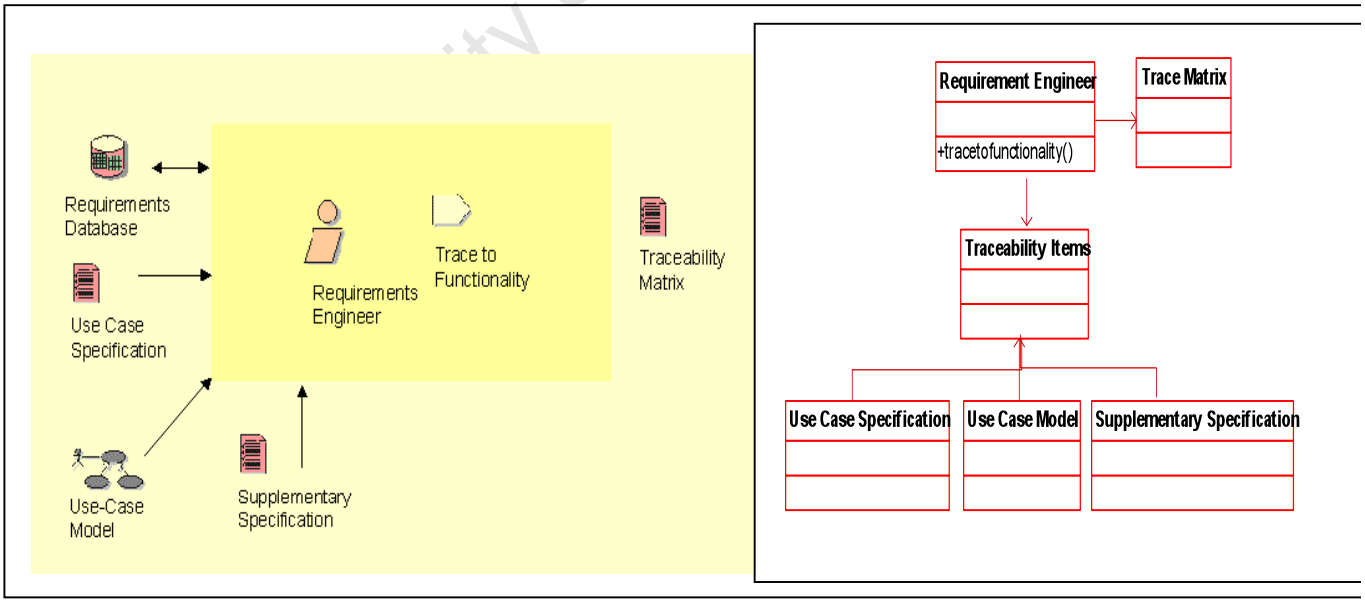



**Figure 9-15: Ericsson Unified Requirement Engineering**

### 9.7.5 Combine Process and Semantics

In Figure 9-16 below, *Real-Life sketch from OSS-RC R6 Project*, we illustrate how process concepts and semantic concepts can be integrated into one pattern. This is a

conceptual pattern, created by Eoghan Lynch, Requirement Manager (OSS-RC R6) during one of the pattern usability workshops.

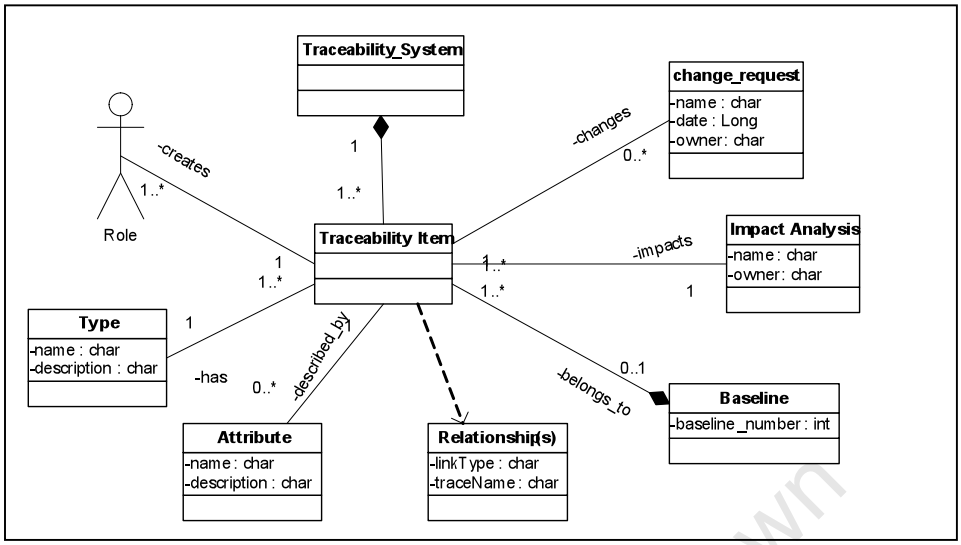



Figure 9-16: Real-Life sketch from OSS-RC R6 Project

### 9.7.6 Empirical Patterns (Case Study)

The below pattern is a valuable pattern that we documented in the OSS-RC R3 project which describing the traceability links between the testing artefacts in Ericsson.

| Pattern Heading | Brief Description |
|---|---|
| **Name** | Traceability Testing Pattern |
| **Problem** | This pattern addresses how to trace between the different testing traceability items in the OSS-RC R3 project. |
| **Context** | This pattern should be applied by any testers involved in the creation of the Test Specification, Test Cases and Test Instructions. The following is a description of the traceable types that make up the Test Management System (TMS). The core traceable types represent the stages of test case development. First a Test Case Specification (TS) is created, describing the purpose of the Test Case and which Test Phases it should be tested in. Based on the TS, one or more Test Instruction (TI) are created. Each TI describes the processing steps needed to implement the Test Specification, specific to a Test Phase. Finally, a Test Instruction may refer to one or more Test Procedures (TPR). |
| **Solution** | The Traceability should be:<br>1. ***Test Specification – Test Instruction***<br>2. ***Test Instruction- Test Procedure***<br>The Test Specification document is named by using the full tag of the software requirement and the test specification number and the sequential number. Example: WINR_SR1_001. The Test Specification Header is tagged as TS requirement and assigned a |

unique identifier by the system. Attributes may be entered using the Test Specification Modification dialog, or by using the Test Specification Attribute Matrix View shown below. The Attributes for the TS are: TS Approved (Yes, No), Priority (High, Medium, Low), Type (COM, NEG, REGR, FUNC, INTER, DR, CHAR, TR_AC, MISC), Status (NEW, MOD, RET), TS ID (Text), Env (SUB, SYS, TAR), Lev (Network, Node, Subsystem)

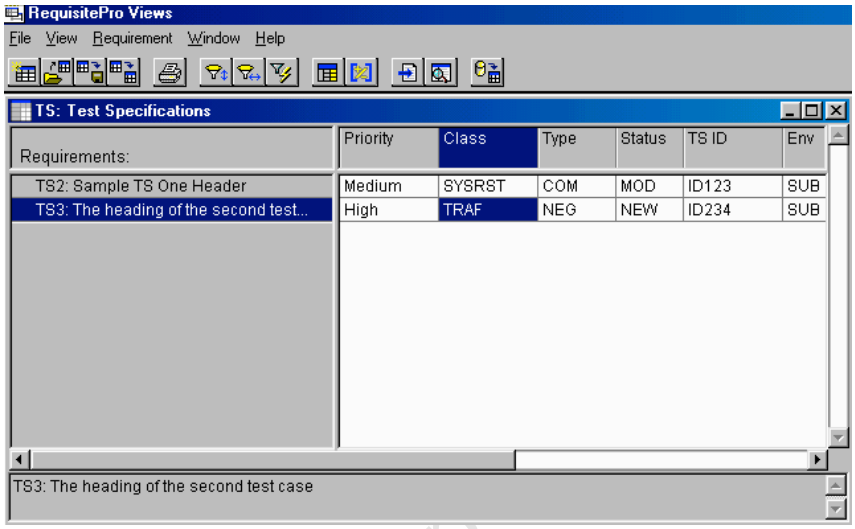



**Figure 17: Test Specification Attributes**

**Test Specification -> Test Instruction**

A Test Specification may be traced to one or more Test Instructions, using the "Test Specification to Test Instruction" Traceability Matrix View. This means that if the Test Specification changes, the Test Instructions traced to it become suspect.

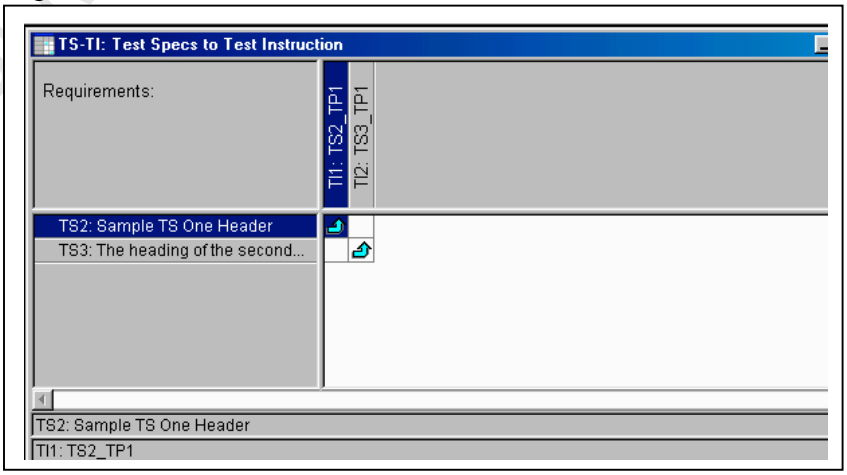



**Figure 9-18: Test Specification traces to Test Instruction**

*User Defined Test Instruction Attributes*: TI Approved (Yes, No). System defined attributes of particular interest include the source of the TI (the TS the TI Traces From), and the Test Procedures it references (Traces To). The person who last updated the TI and the date this occurred is also shown.

**Test Instruction-> Test Procedure**

*Traces To*: Test Instructions are traced to Test Procedures, using the Test Instructions to Test Procedures: Traceability Matrix View

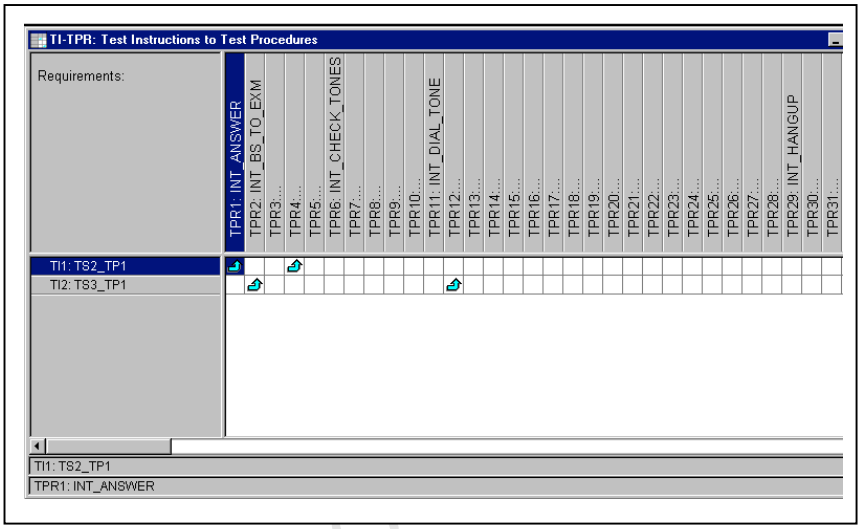



**Figure 9-19: Test Instruction traces to Test Procedure**

Test Instruction provides the steps needed to implement a Test Specification, specific to a Test Phase.

*How to Enter*: Test Instructions are entered in a Test Instruction document. Each Test Instruction is identified by a combination of tag the Test Specification it is related to and the Test Phase it is designed for.

**Constraint**

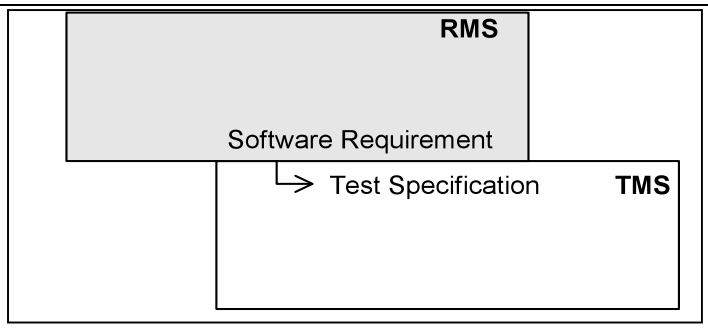



**Figure 9-20: Test Specification Must Trace-Up to Software Requirements**

As shown in Figure 9-20, *Test Specification Must Trace-Up to Software Requirements*, the Test Specification must "trace_up" to the Requirement Specifications. One Software Requirement may

| | |
|---|---|
| | have one or more Test Specifications. The Test Specification document is named by using the full tag of the software requirement the Test Specification is verifying, plus a sequential number. |

### 9.7.7 Empirical Patterns (Case Study)

Once again without describing the entire pattern template we illustrate in the next sections a number of empirical traceability patterns that emerged. These could also be called *Traceability Best Practice Patterns.*

#### 9.7.7.1 Name: Tracing Work Packages

**Creating traces between Work Packages and Requirements**

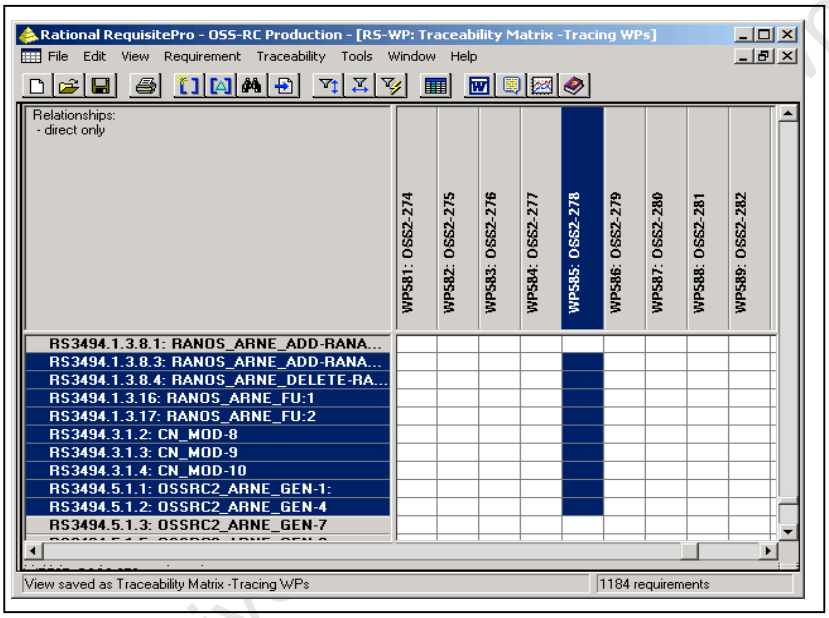



**Figure 9-21: Trace-From work Packages to Requirements**

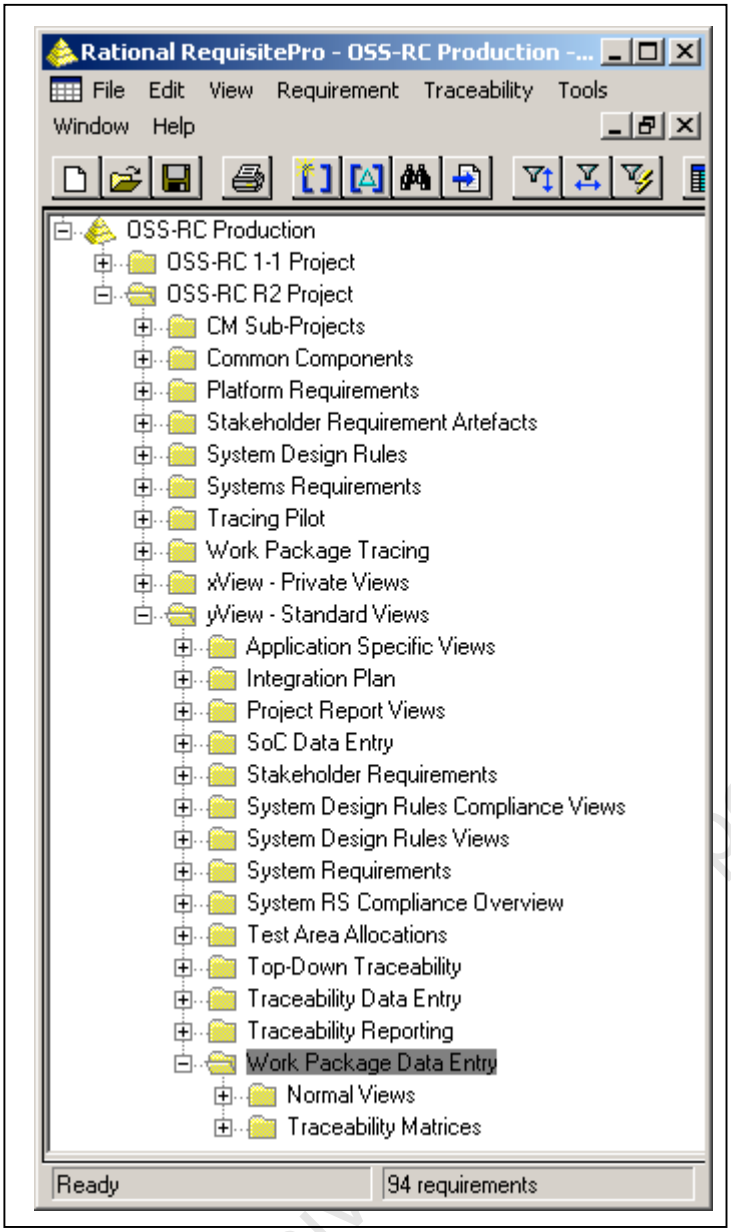



**Figure 9-22: Traceability Items in OSS-RC R2**

As illustrated in Figure 9-21 above, *Trace From Work Packages to Requirements*, the tracing relationships between work packages and requirements are:

- Requirements trace from Work Packages
- Work Packages trace to Requirements

It is important to note that trace relationships will have to be created in this way to keep the database consistent. The rules will be

1. When creating a trace to a work packages from a requirement, always choose the "Traced from" option.
2. When creating a trace to a requirement from a work package, always choose the "Traced to" option.

Rule 1 will always be used when adding these traces as the views have been created to work with requirements.

In Figure 9-22, we show a screen-shot taken from Requisite Pro of all the Traceability Items in the OSS-RC R1 project.



### 9.7.7.2 Main Requirement Specification -> Detailed Requirements (OSS-RC R5)

This pattern was documented in the OSS-RC R5 project. The first link in the requirement traceability chain is the traceability between Input (MRS) Requirements and Detailed Requirements. This traceability is the key to the Detailed Product Requirement List report that is generated from the Requirement Baseline. The MRS sets the scope for all development within the OSS-RC projects. It provides a consolidated view of the OSS-RC node functionality that will be delivered in each new release. Focal Point is the source of the majority of Input Requirements that are input to the project MRS.

The recommended way of working is to build the MRS incrementally in MAR's. MAR's will become the master source for an input requirement. This pattern is the responsibility of the Strategic Product Manager.

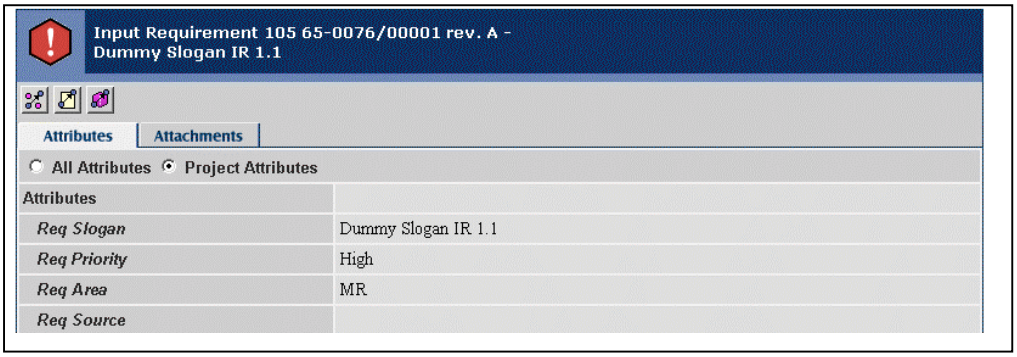



**Figure 9-23: Screen-shot taken from MAR's traceability tool**

### Pattern 2: Derived Requirement -> Parent Requirement

All derived requirements must have parent child relations with their source requirements. In Ericsson, for example, platform and common component requirements are derived from node (BSS/UTRAN) OSS requirements, which are created as child requirements of the BSS/UTRAN node requirement. Derived requirements need to be connected to requirement structures in a baseline.

## 9.7.8 Best Practices (Ericsson)

The difference between empirical findings and best practice patterns is negligible. However, they capture experience based practices, that contain know-how or knowledge that has been proven to be valuable in a specific situation and should be applied in future implementations of the practice.

### 9.7.8.1 Pattern: Statement of Test Coverage – SoTC

All requirements in the project requirement baseline shall be connected to at least one test case. A report shall be generated to verify that all requirements are connected to a test case. This is the responsibility of project management.

### 9.7.8.2 Pattern 4: Statement of Verification –SoV

A statement of verification should be provided on a per shipment basis for all requirements pertaining to the shipment. The ambition should be for 90% of requirement to have a SoV.

The Statement of Verification is issued to indicate the level of test fulfilment (in percentage) for a specific requirement regarding successful implementation and test.

## 9.8 BENEFITS OF PATTERNS

At first glance it may seem as though patterns are just structured guidelines that can be used to describe traceability concepts and practices in pattern format. However, patterns have a number of advantages over guidelines. Welie et al. (van Welie and Trætteberg, 2000) and Borchers (Borchers et al., 2001) emphasise three important differences between patterns and guidelines. The first difference is that patterns record all the information that would normally be required to use a guideline. The context, problem, and solution are all made explicit and a rationale is provided explaining why the solution works. A pattern might also incorporate several guidelines. The second difference is that patterns must

provide a proven solution and examples of the solution in practice. Finally, patterns can be organised into pattern languages allowing the designer to quickly navigate and refine solutions.

Perhaps one of the most significant benefits of patterns is their ability to allow both traceability experts and users to communicate using a common language. This is possible because patterns can be referred to by their title and the use of examples allows non-experts to understand what could potentially be a complex idea. The examples also serve as a means of showing the user a potential traceability solution.

The Centre for Excellence for Traceability describe some of the challenges faced by the traceability community, for example, the challenge of not having a central traceability knowledge base, the lack of traceability certification programs, the lack of understanding of the human factors in the implementation of traceability. We believe that traceability patterns address many of the challenges described by the centres technical report. Of course, other benefits include the better communication of the properties of traceability and the promotion of reuse, where novices can gain experience from patterns developed by experts.

## 9.9 TRACEABILITY PATTERN TOOL (TRAPT)

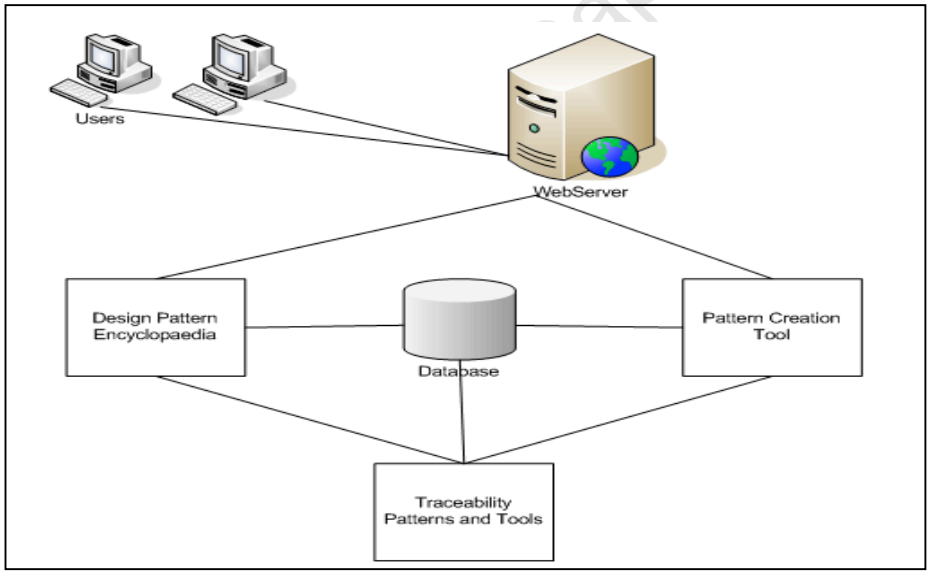
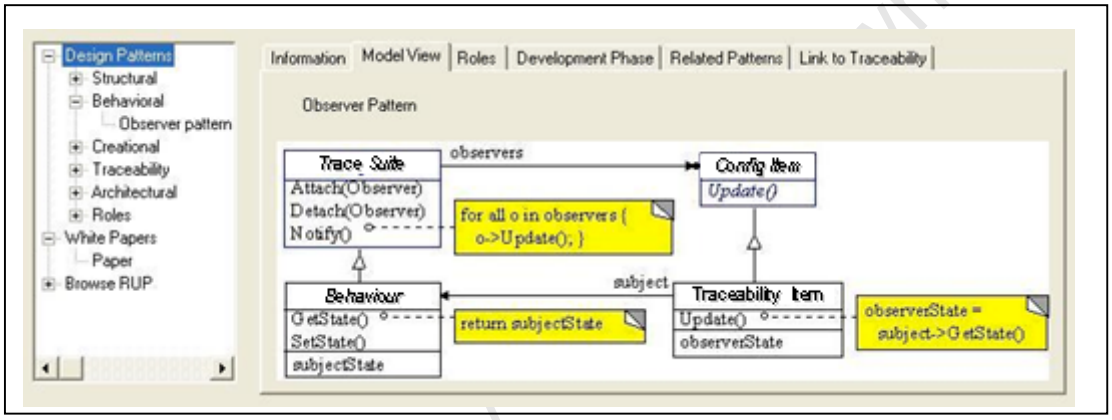



**Figure 9-24 TRAceability Pattern Tool (TRAPT)**

As shown in Figure 9-24, *Traceability Pattern Tool (TRAPT),* the TRAPT tool comprises of three major parts: the Pattern Encyclopaedia, the Pattern Creation Tool and the Pattern Database. The tool was created as part of a student research project at the University of Cape Town. (Kelleher et al., 2004)The pattern database holds all pattern details. The Pattern Encyclopaedia and Creator both access the database. The Encyclopaedia retrieves pattern information from the database. The Creator retrieves pattern information from the database as well as inserts new or reviewed pattern information into the database.

The objectives of the Pattern Encyclopaedia are to enable users to learn about patterns and traceability and to help practitioners search for patterns that would solve a traceability problem. This allows the users to apply the patterns effectively. Learning about patterns and traceability is facilitated by providing information related to traceability, including definitions, applicability and examples. Academic papers and articles are used as a supplementary source of information.

The functionality of the Pattern Creator can be logically split into pattern definition and pattern management. Pattern definition regards the selection of a pattern form and then the definition of the pattern in terms of the fields of that form. Within this functionality the pattern creator is able to enter text and to insert diagrams. These diagrams can either be loaded from external files, or created using the integrated diagramming tool. We included a pattern creation process or workflow through which the patterns must pass. Within this workflow users play roles in moving the pattern toward publishing. Documents can be assigned to patterns.




Due to time constraints we did not carry out testing of TRAPT in any controlled experiment. However, a number of informal testing workshops were organised with undergraduate students. Furthermore, two system demonstrations with industry experts were conducted. The aim of the demonstrations was to get some measure of feedback from practitioners. The experts indicated strong enthusiasm for the TraPT tool.

## 9.10 DISCUSSION AND LESSONS LEARNED

Developing reusable traceability patterns and frameworks is not in itself a silver bullet. Traceability is a communication driven practice and as discussed in earlier chapters, communication across large organizations is difficult. When implementing traceability in particular, the problem is increased due to the lack of a single standard mechanism for persisting knowledge about proven traceability practices which has often resulted in, at best, inconsistent reinvention of practices in different tools, using different processes causing duplication of efforts and in many cases a loss of important traceability information.

Much has been written about providing suitable solutions to the problem, but in our opinion knowledge management would solve many of these problems. This knowledge management based solution should include a standard notation for describing proven traceability practices that incorporates information such as when the particular practice is applicable, tradeoffs associated with using it and a solution that is reproducible. As discovered during the empirical studies the solution also needs to provide a standard

vocabulary to describe traceability, a searchable repository for publishing, sharing, and locating the patterns and a way of bringing together any models related to this domain.

Patterns provide an effective means of communicating traceability best practices and for solving recurring traceability challenges. For example, the Center of Excellence for Traceability technical report could be described in its entirety using traceability patterns, outlining the problems and the challenges faced, giving examples of the problems where possible. Traceability patterns use a template that incorporates a pattern name, the context in which the pattern exists, a description of the problem the pattern solves, a solution to the problem, and consequences or tradeoffs that arise from using the pattern. We give the traceability patterns names so that the solutions they encompass can be communicated and discussed between success critical resources, across geographical locations in a formalized manner. Over time, the names establish a common understanding of the key characteristics of their implementation.

A pattern such as Create Traceability Item, which simply describes how to create a traceability item, can appear trivial on the surface. However, when you consider aspects such as; the rules for creating the attributes, the relationships to other items, the types and the fact that each item must belong to a baseline, you get the idea of why this pattern is valuable to almost any traceability practitioners.

We have presented a generic, pattern-based approach to the formal specification of traceability patterns. The approach has been supported by a number of different examples to patterns discovered during the case study. In the survey chapter we presented some of our findings using a pattern based approach. During this project we learned the following lessons when creating traceability patterns:

- *Lesson 1:* All problems discovered during the empirical study can be represented in pattern format. This formalises the approach to data gathering and presentation. By following this approach presenting the findings and discovering commonalities between identified problems was made easier.

- *Lesson 2:* By translating formalized traceability into natural language, a better basis for discussions with Ericsson was achieved, as this translation could be performed in a uniform way. This was most evident when the OSS-RC R6 Requirement Manager, created a pattern and used it in discussions with new novice traceability users.

- *Lesson 3:* The pattern-based approach is scalable in the sense that more patterns can be added in an iterative manner. This is particularly true when creating models of traceability. For example, Create Traceability Item is the foundation pattern for any modelling effort and further patterns can be built on top of this foundation pattern.

- *Lesson 4:* The traceability patterns provide a consistent approach for describing traceability terminology. Due to the sheer quantity of traceability patterns we did not demonstrate the use of patterns to describe the most basic traceability terms. However, patterns could be created to standardise the language used around the traceability concepts of relationships, link types and item types, to mention a few however, a complete catalog of terms could be created to standardise the language as a pattern language.

- *Lesson 5:* Using patterns can lead to a substantial degree of reuse. With a set of "good" patterns being available, the formalization of matching traceability practices is straightforward, reducing the overall effort and leading to improved readability.

- *Lesson 6:* Patterns Are Decisions, Not Prescriptions. Traceability patterns represent a set of decisions or agreements that were met within an organisation or project. A good pattern should describe the forces which affect the decision whether or not to adopt the particular solution it offers. Traceability patterns provide a good mechanism for gaining agreement on aspects of traceability. They should not be applied before they have gone through a decision process. For example, some of the patterns illustrated in this chapter are generic enough to be used by any project; however, the empirical patterns should only be used within the OSS-RC domain.

- *Lesson 7:* The traceability patterns traverse the entire software development lifecycle. These patterns can be used to describe traceability scenarios from high level organizational aspects of traceability to lower level testing practices.

- *Lesson 8:* The power of the pattern was most realized when the actual practitioners created their own.. This was evident when we first proposed patterns to the OSS-RC Requirement Manager, who was slow to use this approach in the beginning. However, after assistance with the creation of the first pattern, he commenced using the pattern structure to communicate different aspects of traceability. Moreover, the pattern template proved very valuable during the survey because they had a formalized approach for capturing the problems. Patterns are written by practitioners which should be used by practitioners. This is a key ingredient to success.

- *Lesson 9:* In one experimental session with Ericsson we investigated if patterns could be used to replace the OSS-RC R6 Requirement Management Plan. The answer was a resounding yes. The RM plan describes the baseline strategy, the change request strategy, the traceability items, the attributes, the relationships and the tooling strategy. In only a matter of a few hours we had replaced the RM Plan with a number of easy to use patterns. Moreover, in future OSS-RC projects only the patterns need to be changed to reflect changes in the traceability strategy. This saves a lot of time in creating new RM Plans. Gaining agreement on the patterns would also be made easier. Furthermore, the traceability corporate strategy could be better described with a number of patterns.

While the traceability patterns discussed in this chapter are still in its infancy there are a number of areas that should be focused on in future research:

- *Cost Benefit Analysis:* Patterns address the difference between theory and real-world practicality. Cost-benefit analysis (CBA) is a method to reduce uncertainty during decision making and planning. While we did not address cost best analysis, patterns do in our opinion provide a novel approach for assisting executives in making IT investment decisions on traceability. One could add to each pattern expert opinion data, historical data or measurements that would assist in the cost benefit decisions. For example, if a project has 5000 requirements, using the Create Traceability Item pattern, we could calculate how long it takes to execute each pattern and then calculate the cost (in man-hours) by estimating the amount of time spent executing each pattern. By carefully managing the patterns with a tool, executives have new measurements to base their decisions on future IT investments. Measuring the success or failure of particular pattern can assist in all decisions related to traceability. The functionality of TRAPT could easily be adapted to include cost based analysis.

The cost estimation is a vital link in the success or failure of traceability. The price of investing in software based traceability tools can be based on the success or failure of the pattern. Issues such as usability, learning and training costs all affect the decision on traceability.

In Figure 9-25 below, we illustrate a simple example of cost benefit analysis. For example, once again if a project has 5000 requirement and creating each traceability item using a spreadsheet takes 15 minutes per traceability item, however, by using a traceability tool this reduces the time to 10 minutes then the cost saving could be calculated.

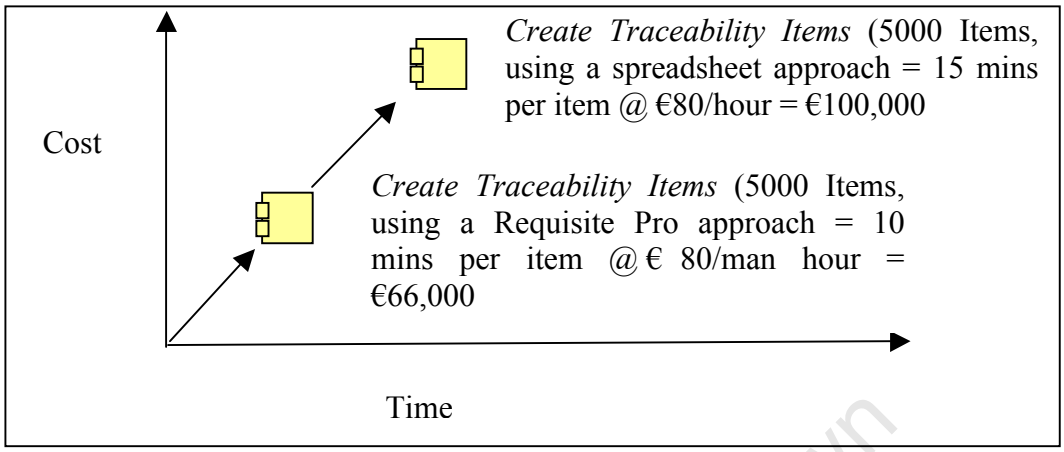



**Figure 9-25 Cost Benefit Analysis**



Constraint Languages: The traceability patterns that we propose are formalised approaches for grouping rules and decisions in relation to traceability We use profiles to extend the UML vocabulary and the patterns define a common vocabulary for traceability users, therefore as we demonstrated in this study, it is possible to use profiles to define a pattern vocabulary in UML. By using these profiles to represent patterns it is therefore not required to use a constraint language or notation to describe the constraints. However, we could have used a language like Object Constraint Language (OCL) to represent the patterns. The Object Constraint Language developed by IBM is a declarative language for describing rules that apply to UML models. It is a precise text language that provides constraints on any model, especially useful for models that cannot be expressed by diagrammatic notation. We used UML models or sketches where possible, but we could have used a language like OCL to denote either the models or the constraints that applied to the patterns. In some of the patterns, where it was difficult to represent the solution using a sketch OCL could have been used. In future research projects we would recommend integrating a constraint language or a temporal logic language to the pattern approach, thereby formalising the solution further.

For example, we could use OCL language to define the context or the limit in which the statement is valid, the property that represents some characteristics of the context, the operations for manipulating or qualifying the property and keywords like “if-else” statements that are used to specify the conditional expression. In Figure 9-26 below we represent one class in OCL. This simple example only demonstrates that OCL could be used to describe constraints in UML.

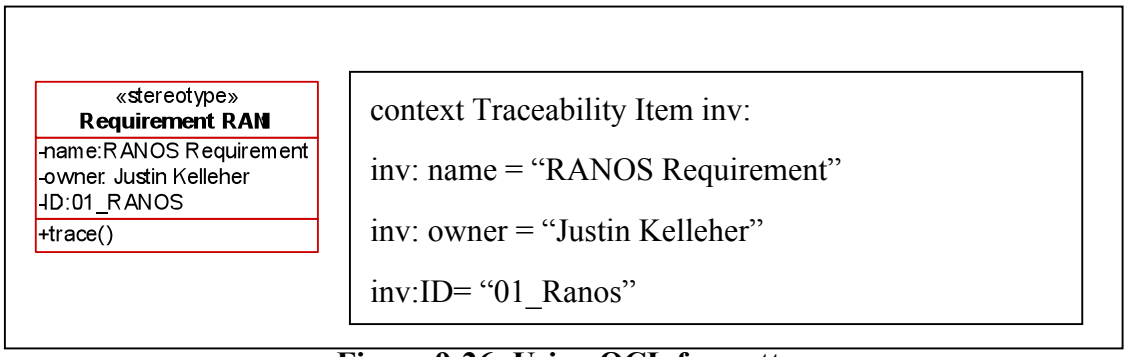



**Figure 9-26: Using OCL for patterns**

## 9.11 CONCLUSION

In this chapter we introduce traceability patterns as a novel pattern based approached at describing and communicating many aspects of traceability including the traceability data, the traceability process elements, the empirical data, best practices and even the bad practices. While the development of these traceability patterns is in its infancy there are many positive applications including their use in the creation of a knowledge base of traceability best practices, in the development of traceability training certification programs and the capturing of many human aspects of traceability. While we did not fully investigate the use of the patterns for assisting with cost benefit decisions we firmly believe that the patterns provide a powerful approach for practicing this. We hope that in future research that further applications of traceability will be investigated and further evaluations of their suitability in different contexts will be developed.

# Chapter 10 CASE STUDY REVISITED: MAJOR CHANGES AND HOW THEY EFFECTED TRACEABILITY IN OSS DOMAIN

## 10.1 INTRODUCTION

The software engineering field's accelerating rate of change makes post-project reflection an essential activity for organizations to evaluate and learn for future projects. Today, corporate Darwinism means adapting to change; changes in the marketplace, changes to customer expectations and changes in technologies. Even the best intentioned IT companies often fail because executives get carried away with the latest technology trend often overlooking the need to learn from the past experiences of project team members. To learn you must ask questions. To hear you must listen. Documenting organisational and developmental changes is a valuable practice both to organisations and to the research communities. With shorter lead times and quicker time to markets, organisations have little time for reflection, instead keeping their focus on the next development.

In this chapter we reflect on the changes that occurred within the OSS-RC domain. We begin with a brief review of our findings in 2004. We address the changes that occur and the implications that these changes had on the traceability discipline. We review how the introduction of a new propriety traceability tool impacted the overall practice of traceability and how this tool impacted the attitudes of the engineers towards the importance of traceability.

The primary purpose of this chapter is to report the changes that occurred to the traceability practices in the context of the Ericsson's OSS product line between the period of 2004-2007.

## 10.2 THE PROBLEMS THAT WERE IDENTIFIED IN 2003

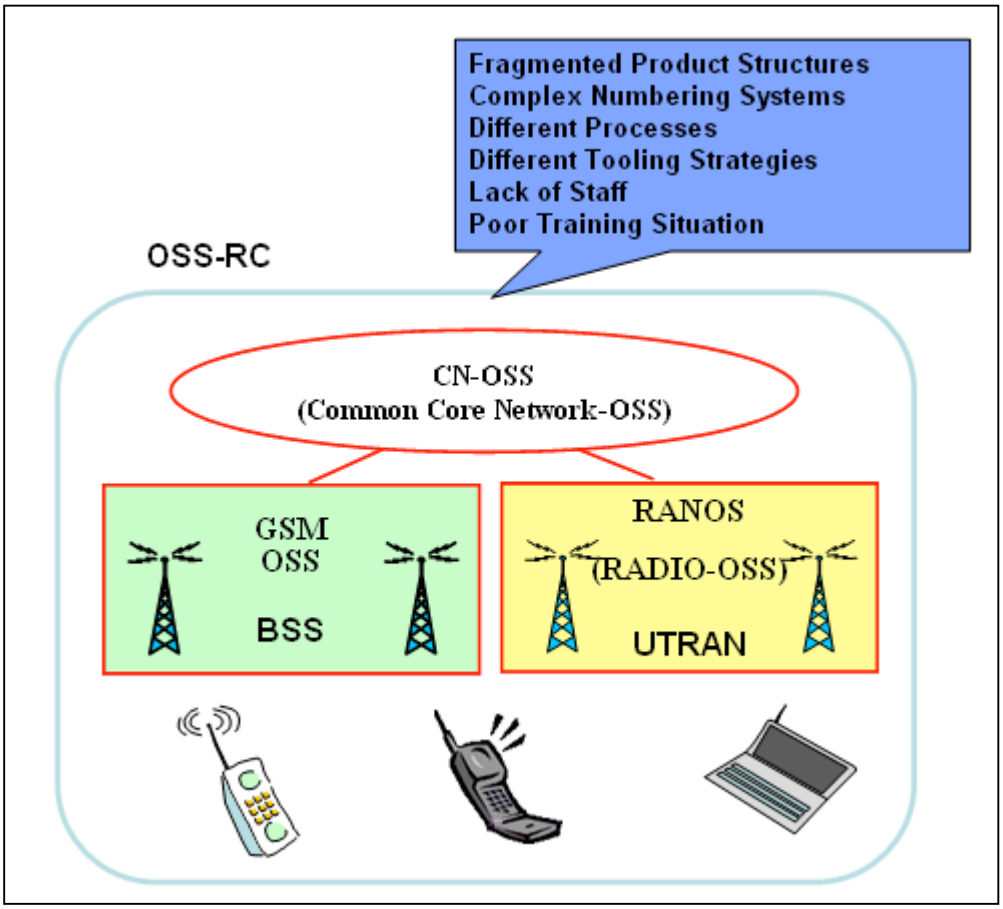



**Figure 10-1 Problem Overview (2004)**

In Figure 10-1 above, *Problems Overview 2004*, we illustrate the main problems discovered in 2003. The main focus of the case study was to gain an understanding of the factors that influence traceability. This can only be made possible by investigating the changes that occurred and the impacts these changes had on traceability in general. In Chapter 4 we described the case study research method, the results of on-site interviews in 2004 and the observations that we made in this period. One-on-one interviews with 27 people were conducted with many different types of engineers including the project manager's, requirement manager, configuration manager, system architects, designers and test personnel to investigate the factors that influence traceability per role basis and gain an understanding of the attitudes of the different roles to traceability.

Also in Chapter 5 we introduced Ramesh's conceptual model of the factors that influenced traceability. Ramesh put forward the theory that institutional contexts and strategic conducts influence each other over time. Ramesh further subdivides the institutional context into organizational, environmental and system development contexts. We reused Ramesh's model by describe traceability from an institutional, organisational and environmental perspective in the context of OSS products.

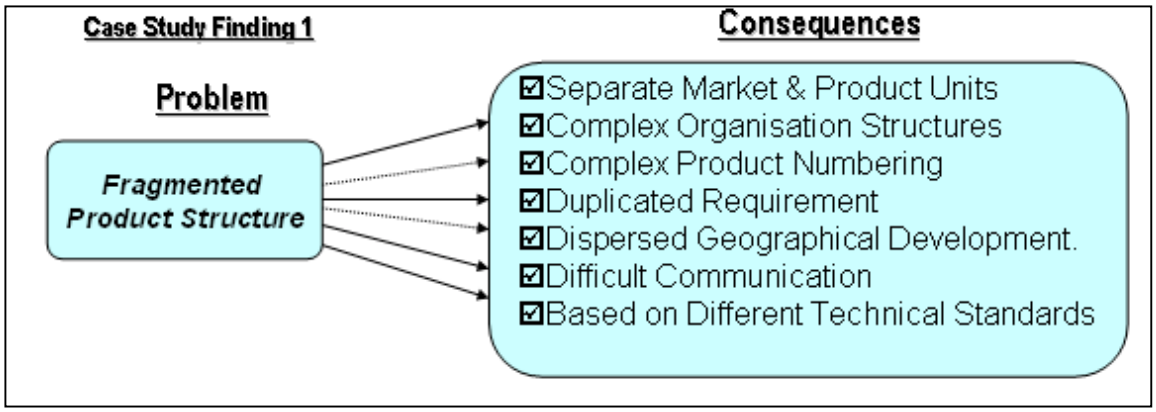



**Figure 10-2: Fragmented Product Structure**

***Finding 1 Fragmented Product Strategy:*** The OSS product-line was sub-divided into three separate but related products, namely RANOS, GSM-OSS, CN-OSS. As illustrated in Figure 10-2, *Fragmented Product Structure*, we describe the consequences of such a fragmented product approach which influences the discipline of traceability either directly or indirectly. These consequences are best described as:

- Separate Market and Product Units: Each product had separate market units and product management units with different product strategies and development approaches. The source of the requirements and the traceability items is complex. To overcome this each traceability item must have a source identifier attribute.

- *Complex Organisation Structures*: Is one of the major levers or challenges that influences traceability. Often leading to a lack of uniformity in the definition of the roles across organisation, for example, the responsibility of a requirement manager in one organisation is different to the responsibilities in another. Unclear or dysfunctional reporting, for example, the method by which organisations report major changes or impact analysis. Poor visibility or sharing of information between organisations, for example, poor visibility into each others requirement database.

- *Complex Numbering Systems:* By their very nature, enterprise applications like the OSS-RC, are complex. They have many internal modules and interfaces with many applications across diverse geographical locations. As the number of versions of both your application and the applications it interfaces increases, managing these interfaces becomes much more difficult. A key ingredient to successful product development and the implementation of traceability is consistent product and document numbers.

- *Duplicated Requirements:* Due to the tight technological coupling between the products in many cases requirements were duplicated across all three product development units. Poor management and control of these requirements increased the risk of duplication in development effort or loss of information on changes incurred in different projects.

- *Dispersed Geographical Development:* In all sixteen design houses were involved in the development of OSS products.

- *Difficult communication* between the success critical roles.

***Finding 2 Lack of Uniform Process:***

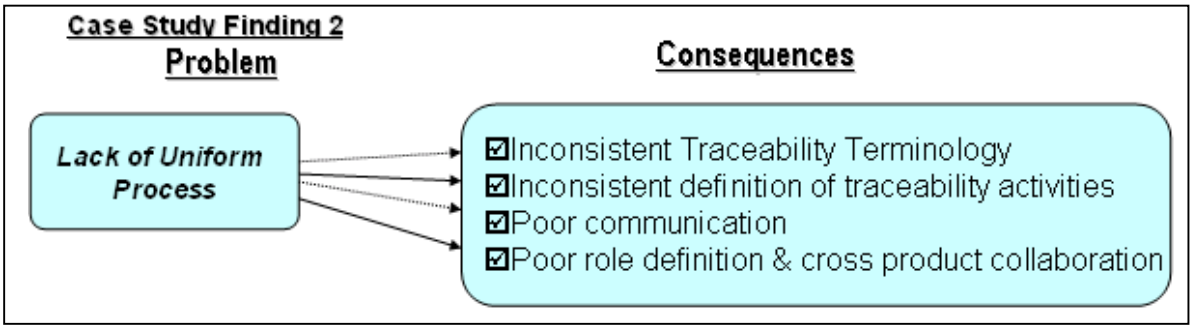



**Figure 10-3: Lack of a Uniform Process**

In Figure 10-3 above, *Lack of a Uniform Process*, we illustrate the effects of a lack of a uniform process, including, the inconsistency in terminology being used, poor definition of traceability activities, poor communication and poor role definition.

**Finding 3 Decentralised Tooling Strategies**

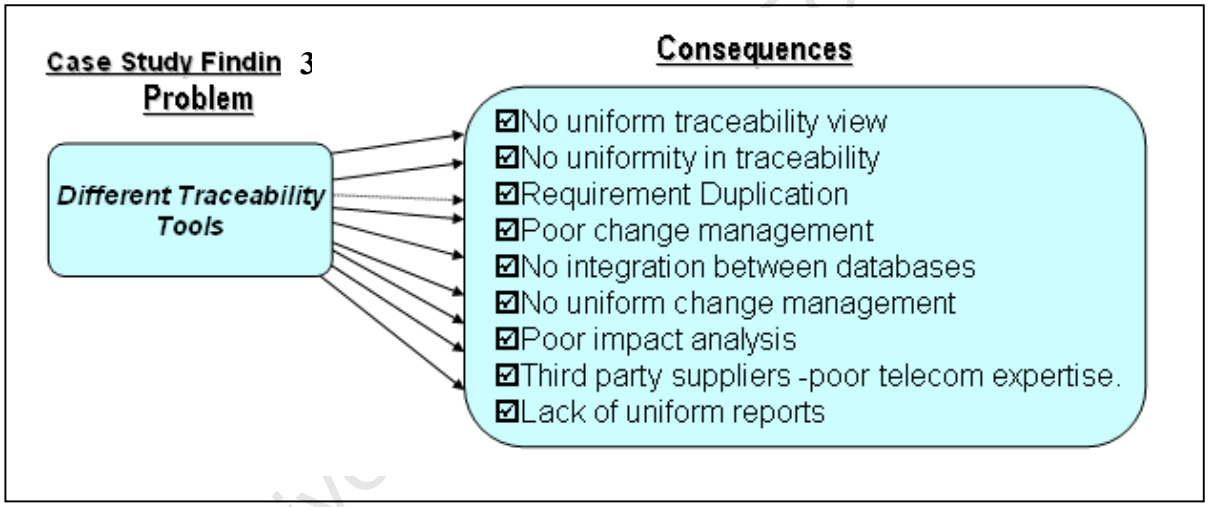



**Figure 10-4: Decentralised Tooling Strategy**

As shown in Figure 10-4 above, *Decentralised Tooling Strategy*, different tooling strategies caused many problems for the development units. In 2003, the traceability tooling situation for the product development units is illustrated in Figure 10-5 below, *Tooling Situation in 2003*. GSM-OSS and RANOS used Rational's RequisitePro while CN-OSS used Telelogic's DOOR's.

| Product | Traceability Tool |
|---|---|
| RANOS | Rational RequisitePro |
| CN-OSS | Telelogic Doors |
| GSM-OSS | Rational RequitePro |

**Figure 10-5: Tooling Situation in 2003**

***Finding 4 Staff Shortages:***

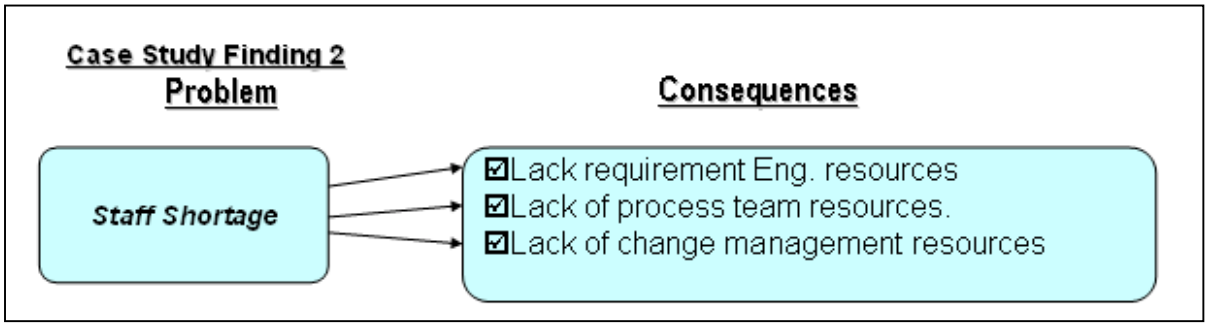



**Figure 10-6: Staff Shortage**

The U.S. economy entered a recession in March 2001 with a general consensus that it ended sometime around December 2001. We now know that during the investment boom of the late nineties that firms vastly overspent with an increase in employee headcount in an effort to satisfy a level of demand for their products which simply proved to be unsustainable. Cutbacks in IT investment and job losses were endemic during this recession period. Ericsson instituted a regime of restructuring in 2001, which reduced its workforce from 107,000 to just over 60,000 in 2002. By June 2003, Ericsson announced its eighth consecutive quarterly loss and further job losses were predicted.

During the early interviews it became clear that a lack of staff bonuses and pay increases was leading to a higher attrition of competent staff. Staff shortages, has a huge impact on any organisation. The effect of this towards traceability was great. In this case it lead to poor allocation of resources for less essential activities and traceability fell into this category. It leads to a lack of support around change management issues, a resistance to change by users and project managers, poor review processes and eventually poor implementation of traceability.

**Finding 5 Poor Training Situation**

As part of the third party supplier agreements, only training delivered by certified instructors could be delivered. The certification process was expensive and time consuming and forcing the product development unit to use the third party supplier instructors. Overall many projects were unwilling to train all their staff with the requirement management tools. Trained mentors were identified in each project however the time needed to complete such mentoring was not allocated to each resource.

## 10.3 MAJOR CHANGES TO THE FACTORS THAT INFLUENCE TRACEABILITY (2003-2007)

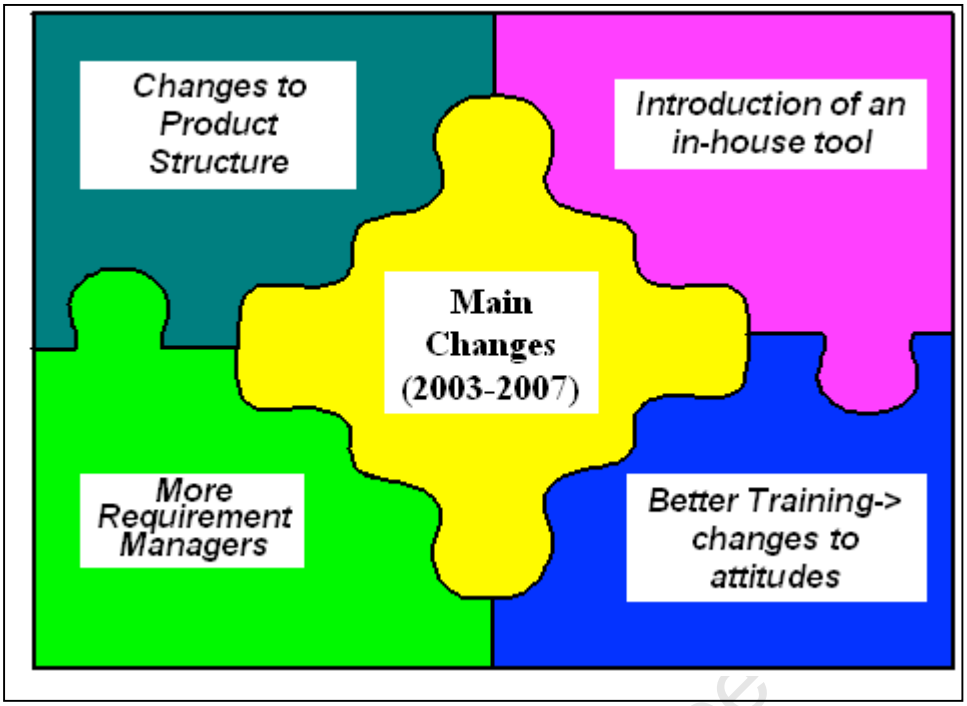



**Figure 10-7: Major Changes 2004-2007**

In Figure 10-7 above, *Major Changes 2004-2007*, we illustrate the major changes that took place between 2003 and 2007.

### 10.3.1 Changes to the Product Structure

In 2002 the OSS anatomy was divided into three products GSM-OSS, RANOS (Radio Access Network OSS) and CN-OSS (Core Network OSS). Each product was further decomposed into sub-products or functional components which are developed by projects. The projects are further divided into sub-projects. The Product Development Unit-OSS (PDU-OSS) was responsible for all aspects of the development of the product line from strategic product management, operational product management, systems (or architectural) development, design, implementation, test, integration and verification, supply chain management, third line support and product maintenance. However, a PDU could be dispersed over many sites, with product management in Sweden and projects located across international boundaries.

At the start of 2002 there were sixteen design houses developing OSS's. The "three-to-one" product strategy consolidated the number of design houses to three by the end of 2002. The Product Development Unit (PDU) for all OSS-RC development became the responsibility of LMI-Athlone, Ireland. As shown in Figure 10-8 below, "Three-to-One" Product Strategy, the integration of the three products occurred over three separate releases. CN-OSS in OSS-RC 1.0, GSM-OSS in OSS-RC R2 and RANOS in OSS-RC 1.2.

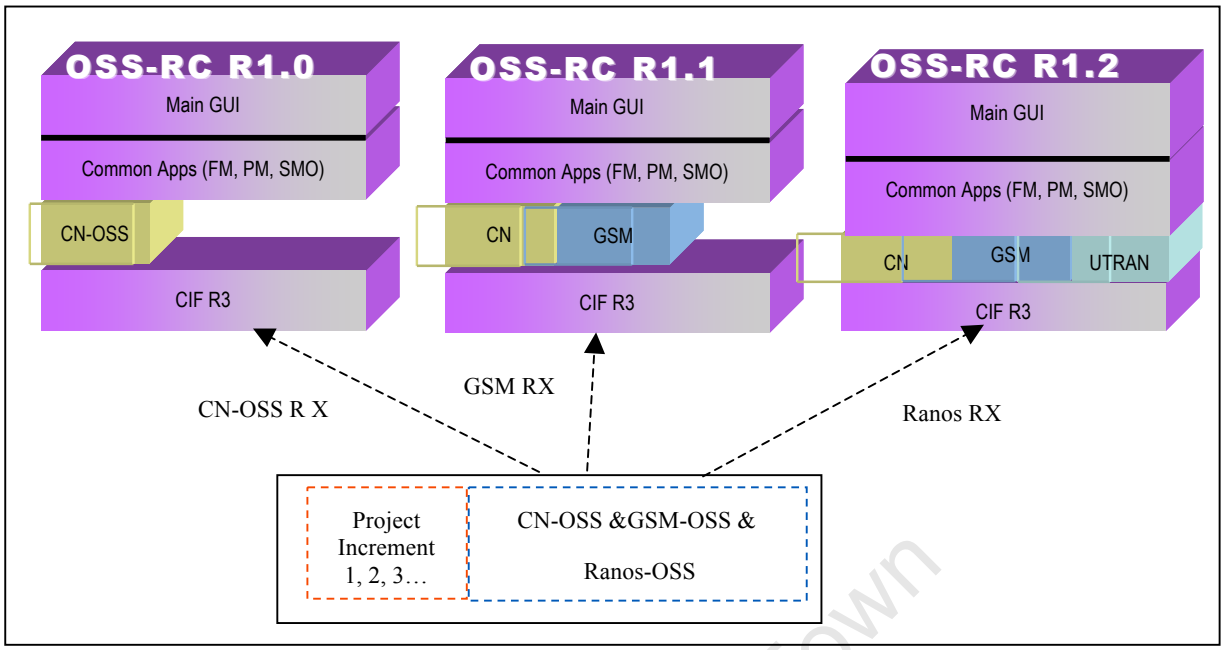



**Figure 10-8: "Three-to-One" Product Strategy**

The implication of such a strategy on traceability was immense. As shown in Figure 10-9 below, *Change to Product Structure*, this lead to a consolidation of development units, the integration of market units, less complex organisational structures, a unification in product numbering, easing the possibility of duplicated requirements, better communication, the consolidation of process team leading to better change and impact analysis management. While the product consolidation strategy was market and standard driven it had a major impact on the simplification of all software engineering practices including traceability.

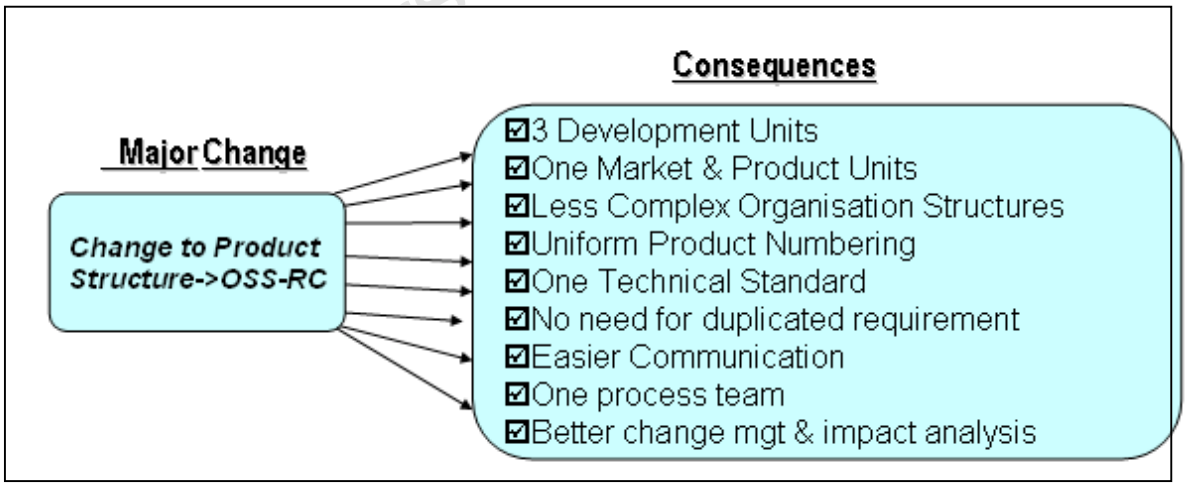



**Figure 10-9: Change to Product Structure**

### 10.3.2 Change to Tooling Environment

In 2004 the introduction and mandating of the a new traceability tool strategy had many positive effects on the practice of traceability. Firstly, all market units consisting including the stakeholders were using Telelogic's tool Focal Point. Secondly, the overall OSS Product Development Unit was mandated to use MAR's (see Figure 10-10, *Tooling Situation 2004*)

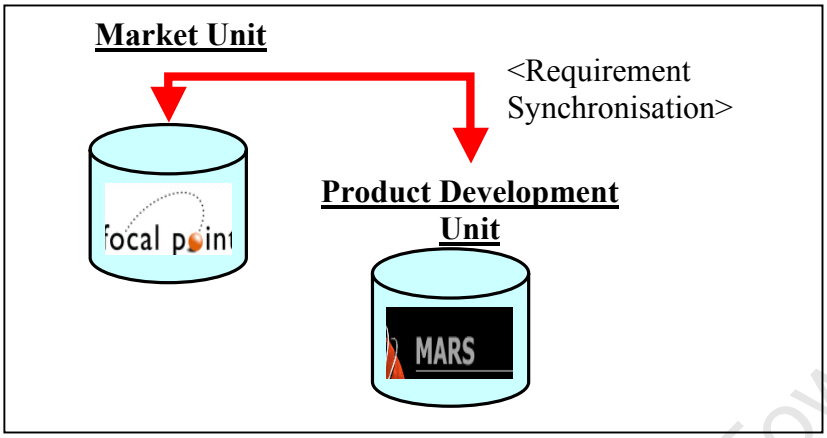



**Figure 10-10: Tooling Situation 2004**

#### 10.3.2.1 MAR's

MAR's standing for MAtrix Reference System is developed on top of the Matrix platform and customized to support the developing projects within Ericsson. It was mandated by Corporate Methods and Tools. As shown in Figure 10-11 below, *MAR's Overview*, we see that MAR's tool consists of three major parts, Requirement Management, Configuration Management and Test Management, where the Configuration Management includes both Baseline and Change handling. MAR's was sponsored by Ericsson R&D Products Methods and Tools and is owned entirely by Ericsson.

In 2007, MAR's had approximately 2000 registered users in all of Ericsson. MAR's was used for OSS projects from OSS-RC R4 onwards. This elevated many interoperability problems, where the stakeholders and the projects had visibility of the same items. Using the same tool facilitates information exchange. MAR's is divided into "Vaults". Each vault can be product development unit specific. Each project in the vault is a separate instance in MAR's.

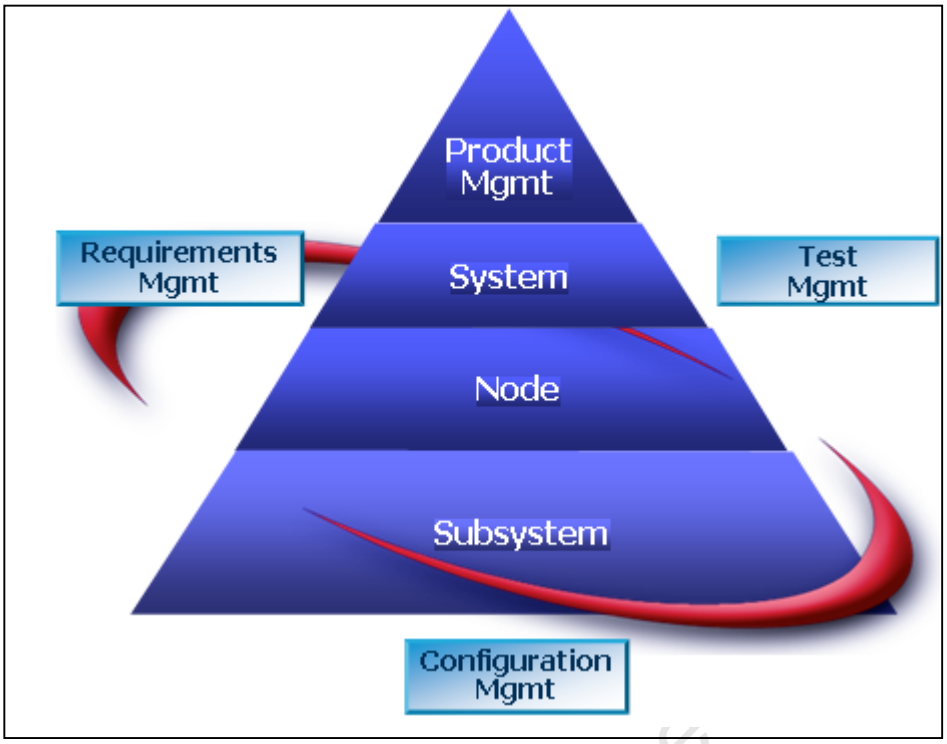



**Figure 10-11: MAR's Overview**[11]



### 10.3.2.2 Focal Point

According to the Ericsson R&D IS/IT Management corporate decisions Focal Point shall be used by the Product Management, for release planning, prioritization and traceability between product requirements. MAR's is used by the projects for project management of change requests, baseline and revision management. As shown in Figure 10-12, *Synchronisation between Focal Point and MAR's*, we illustrate that it is possible to sychronise between the two tools. Focal Point focuses on the early stages of requirement management, like release planning, product positioning and prioritization. The information shared between Focal Point and MAR's consists of a certain number of attributes that are sent from one system to the other. Some attributes, such as slogans and description are always synchronized and are under revision control while others may be transferred only from one system to the other without having a new revision created. During the project lifecycle, MAR's will automatically transfer detailed information from the Configuration Management to Focal Point for a synchronized requirement.

[11] This diagram was extracted from the MAR's supporting documentation

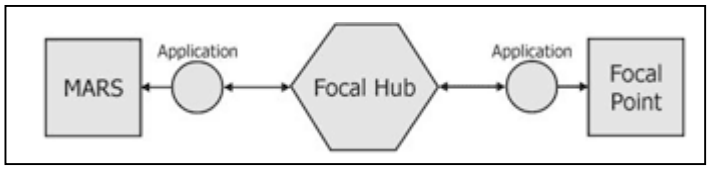



**Figure 10-12: Synchronisation between Focal Point and MAR's**

Focal Point is used for prioritization of requirements and acts as a container for all product requirements. When a project is started, the prioritized requirements are synchronized to MAR's. When a project is finished, any "spill over" requirements shall be transferred back to Focal Point again. Focal Point is always the master of the synchronized requirements. Solution Architects work only in Focal Point except in relation to change management issues. Project management works only in MAR's. A project management person in MAR's has at least one of the roles project manager, requirement coordinator or configuration manager. In Figure 10-13 below, *Focal Point Inputs Requirements to MAR's*, we illustrate that the stakeholder's requirements move to the PPF (Product Planning Forum) and then into MAR's. The PPF is a forum for managing the strategy and control of the products.

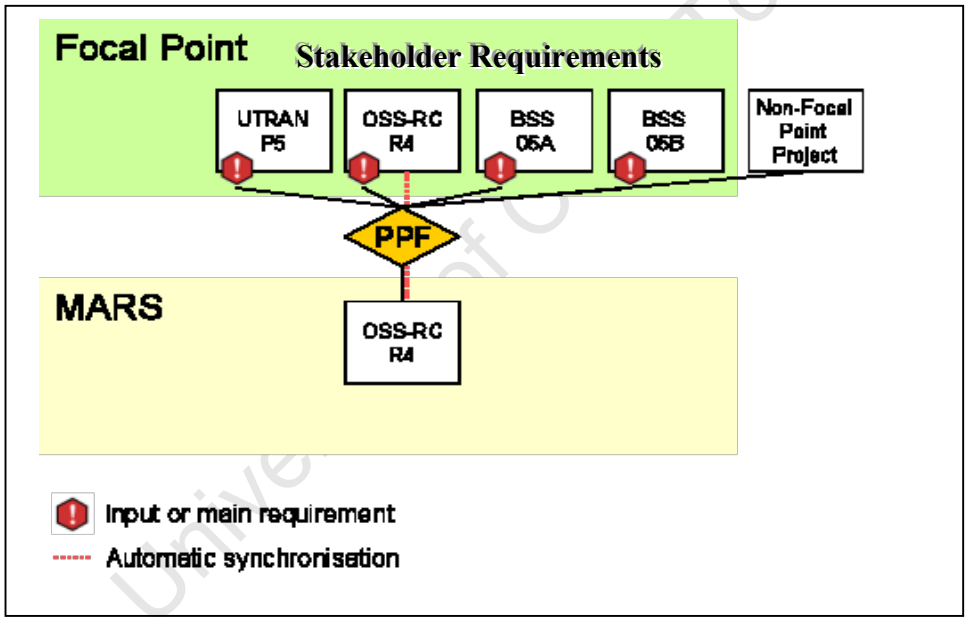



**Figure 10-13: Focal Point Inputs Requirements to MAR's**

### 10.3.2.3 Impact of Changes in Tooling Strategy on Traceability[12]

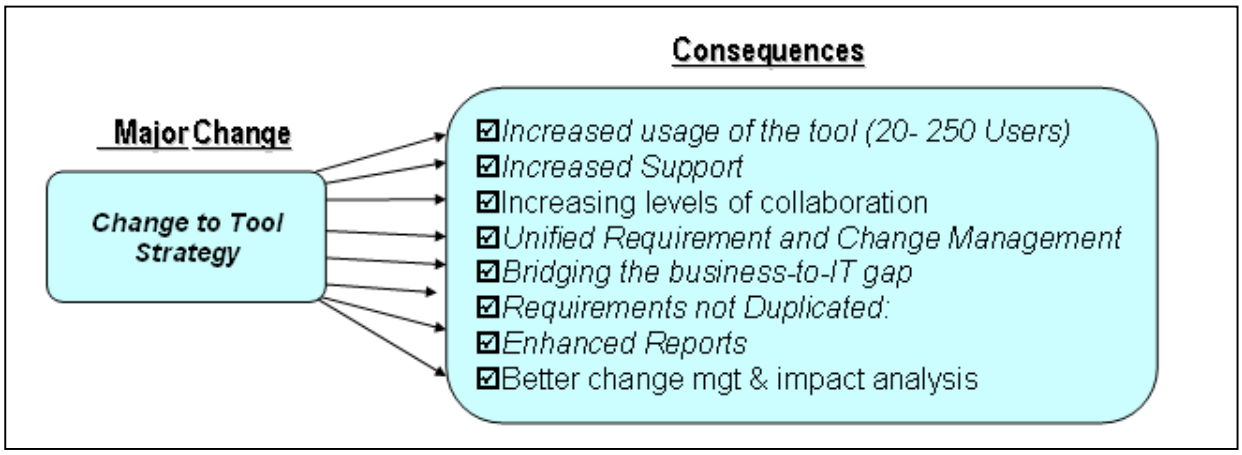



**Figure 10-14: Impact of Change in Tooling Strategy on Traceability**

In Figure 10-14, *Impact of Change in Tooling Strategy on Traceability*, we illustrate the changes that were captured during the interview sessions in 2007, on the influences of the change of a tooling strategy on traceability. These changes are described as:

- *Increased usage of the tool:* In 2004, there were less than 20 people using Requisite Pro in the RANOS project. By 2007, there were over approximately 250 people using MAR's. On further investigation the feedback that we received was that the tool was more efficient, was easier to use and the requirements were kept up to date.

- *Increased Support*: MAR's was supported by internal IT staff, therefore if there were any problems with the tool or a need for tool configurations, the support had quick turn around times with problems.

- *Increasing levels of collaboration* between roles and organisations. Because everyone shared the same view, increased collaboration was reported by many users. The collaborative nature of requirement engineering and traceability was a major improvement from earlier efforts with 1) wider geographic distribution of traceability practices and 2) enhance real-time communication among individual team members.

- *Unified Requirement and Change Management:* Using MAR's and its synchronisation with Focal Point, requirements, change requests and impact analysis are unified into the same view. These factors made all aspects of traceability easier to maintain and use.

- *Bridging the business-to-IT gap*: The alignment of the business view or Market Unit with the Product Development Unit into the same workspace. This is achieved by



[12] On reflection I think the demand for traceability definitely has increased, and that while research into traceability has continued this is not reflected in the industrial tools. For example, in 2009, working for a Telco's called COLT in London I was the primary stakeholder in bringing a new requirement management tool into the organisation. Before selecting a tool I carried out a tool survey, similar to the one carried out in this research, to identify a suitable tool. What was most alarming to discover was that the traceability functionality had not improved at all in the industrial tools I assessed.

synchronising Focal Point and MAR's. This alignment includes the ability for business requirements to drive the product development and to combine business and development together.

- *Requirements not Duplicated:* Requirements were no longer duplicated across three separate databases reducing the risk of development duplication or even requirement loss.
- *Enhanced Reports*: Running reports from a single repository meant that the coverage of the report did in fact give a true reflection of the state of the traceability situation. With requirements, change requests and impact analysis in the same repository, manager's have a better insight into project risk, status, change and trends.

### 10.3.3 Increased Number of Requirement Managers

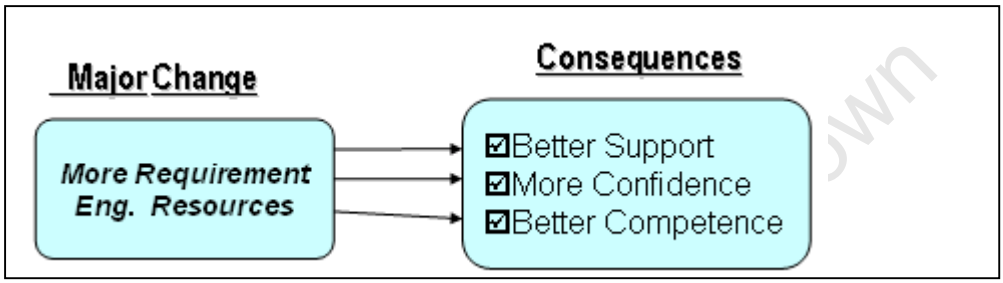



**Figure 10-15: Increased Number of Requirement Managers**

In 2004, one resource was allocated to Requirement Engineering. By 2007, this number had increased to four. Three of these roles could be best described as administrative or instructive in nature; carrying out desk-to-desk support to ensure that all users have access and support with the tool. The other resource had a more management function with tasks that included the creation of the Requirement Management Plan, working with the process team to ensure that the requirement process satisfied the development organisations needs and being involved in CMMI assessments. Using open discussions with the end users, the following points were documented as a result of the new resource allocation: (see Figure 10-15, *Increased Number of Requirement Managers*, above)

1. Users of MAR's all indicated that there was much better levels of support. The desk-to-desk approach gave a personal feel to getting tooling problems solved. There was overwhelming praise for the increase support

2. More confidence with the system. With the quicker turn around times with any issues with the tool, users had a higher level of confidence in using the tool.

3. Better Competence: Users were more willing to learn new functionality as the technical support was readily available to them.

### 10.3.4 Changes in Training Practices

In 2004, a notable problem was the shortage of resources receiving training in requirement engineering and traceability. This was due to the previously discussed problem of entering into licence agreements with third party suppliers which included contractual agreements mandating the use of certified instructors to deliver the training. With the

development of MAR's, a new era in training provision was developed. The Ericsson's intranet had all available MAR's product and introductory training information on-line. A new course was developed with modules covering Requirement Management, Change Management and Impact Analysis as well as the necessary tooling training. Furthermore, a trainer was provided by an internal training organisation. The availability and cost effectiveness of this course dramatically increased the number of trained resources in MAR's; which had a huge impact on the overall organisations traceability competence, confidence in the system and the attitudes towards traceability.

### 10.3.5 Attitudes towards Traceability

A notable change in attitude was documented in the duration between 2004 and 2007. When starting this research the upper management team all understood the importance of traceability, however, there was poor attitudes documented by many of the designers and testers. By 2007, a major swing in the improvement of attitudes was observed. We will look at this further in Section 10.4.

### 10.3.6 Changes to Process

The Ericsson Unified Requirement Engineering Process (EUREP) was not used by most projects rather using informal approaches, after 2003. In 2004, OSS-RC commenced the use of IBM's RMC (Rational Method Composer) process modelling tool. RMC enables the management, configuration and deployment of process models. While traceability was defined using these process models it still did not define all the roles and their responsibilities for all software engineers in the OSS-RC product development unit. However, with the introduction of MAR's, which had strong process capabilities incorporated into the tool, there was less of a need for better process development.

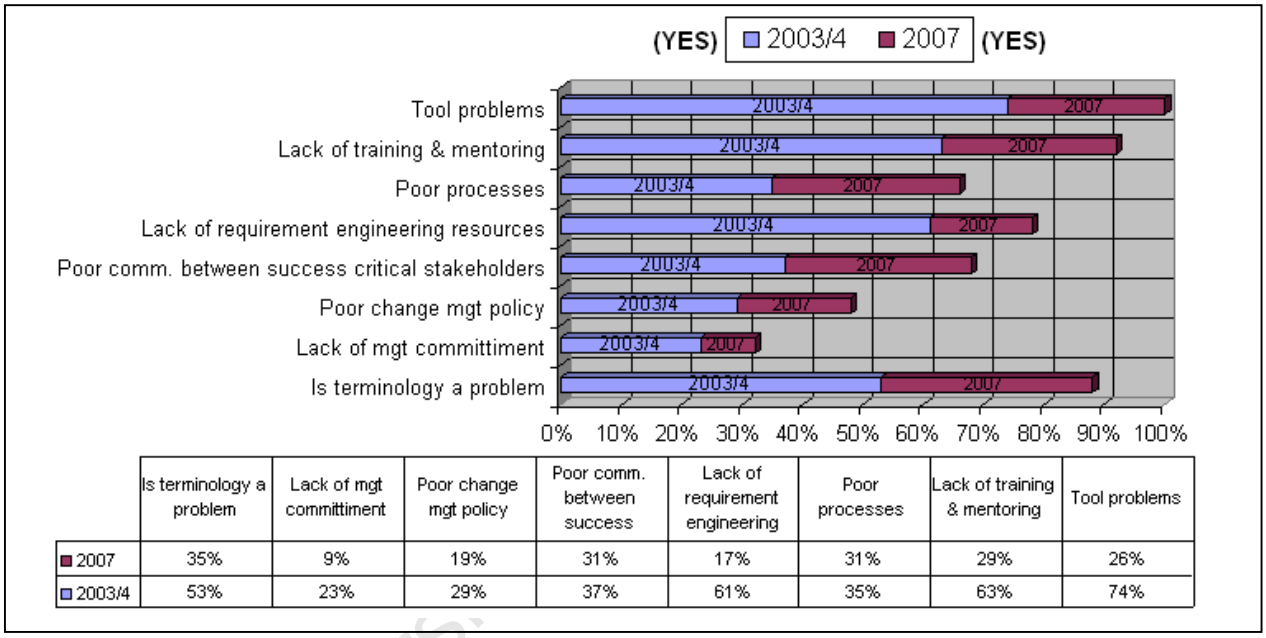
## 10.4 RESULTS FROM INTERVIEWS (2007)

In 2004 we asked the following questions during interview with 27 resources involved in the product development lifecycle:

1. Have you a problem with the tooling solution?
2. Do you feel you receive adequate training to traceability?
3. Do you feel that the process describes traceability sufficiently?
4. Do you feel that there are adequate resources for requirement management tasks?
5. Do you feel that communication between the success critical resources is a problem?
6. Do you feel that the change management policy are sufficient?
7. Do you feel that a lack of management commitment is a problem?
8. Do you feel terminology is a problem when working with traceability?

In the Figure 10-16 below, *Results from Interviews*, we compare the "yes" responses in 2003 with those received in 2007. There were some variations in the interview data. Only 23 people were interviewed in 2007, and only 17 were the same as those interviewed in

2004. The results clearly show that in all cases there were changes in attitudes towards traceability. One can see that poor communication between the success critical resources made little change from 37% in 2004, to 31% in 2007. Furthermore, there was little change in the number of respondents who believed that the processes did not support traceability sufficiently, with 37% in 2004 with this figure reducing only to 31% in 2007. However, the most resounding changes were in relation to the tooling environment. In 2004, 74% stated that they had tooling problems, however, with the introduction of MAR's this number had reduced to 26%. The change in requirement engineering allocation also provided some positive changes in attitudes with many of those interviewed responding positively to the new support.




| | Is terminology a problem | Lack of mgt committiment | Poor change mgt policy | Poor comm. between success | Lack of requirement engineering | Poor processes | Lack of training & mentoring | Tool problems |
|---|---|---|---|---|---|---|---|---|
| 2007 | 35% | 9% | 19% | 31% | 17% | 31% | 29% | 26% |
| 2003/4 | 53% | 23% | 29% | 37% | 61% | 35% | 63% | 74% |

**Figure 10-16: Results from Interviews**



## 10.5 LESSONS LEARNED FROM CASE STUDY IN OSS

The researcher used ethnographic techniques to research traceability by joining the OSS-RC product development unit from 2004-2007, gaining an insight into individual behaviour and the organizational context for traceability. In essence, traceability ethnographers immerse themselves in organisations for months or years, to obtain the relevant data.

Unfortunately, there are many ways in which ethnographic observation can go wrong: it is easy to misinterpret observations, to disrupt normal practice, and to overlook important events. We recommend the following guidelines for preparing for the evaluation, performing the field study, analyzing the data, and reporting the findings in our experience include the following:

**Preparation**

- *Understand organization policies and work culture.*
- *Familiarize yourself with the product family and its history.*

- *Set initial goals and prepare questions*.
- *Gain access and permission to observe or interview*

**Field Study**

- *Establish rapport with managers and users*.
- *Observe or interview users in their workplace*.
- *Review documentation* including Requirement Management Plan, process definitions, tooling documentation, relevant development documentation (including systems, design and test)
- *Document your visits.*

**Analysis**

- Compile the collected data in numerical, textual, and visual diagrams.
- Quantify data and compile statistics.
- Interpret the data.
- Refine the goals and the process used.

**Reporting**

- Prepare a report and present the findings.

While these notions seem obvious when stated there was a notable lack of documentation on how to carry out a case study into traceability. For example, deciding what information to capture was one of the obstacles that we needed to overcome. Traceability traverses so many disciplines it is often difficult to decide what data should be recorded from the different users.

## 10.6 AUTHORS REFLECTION

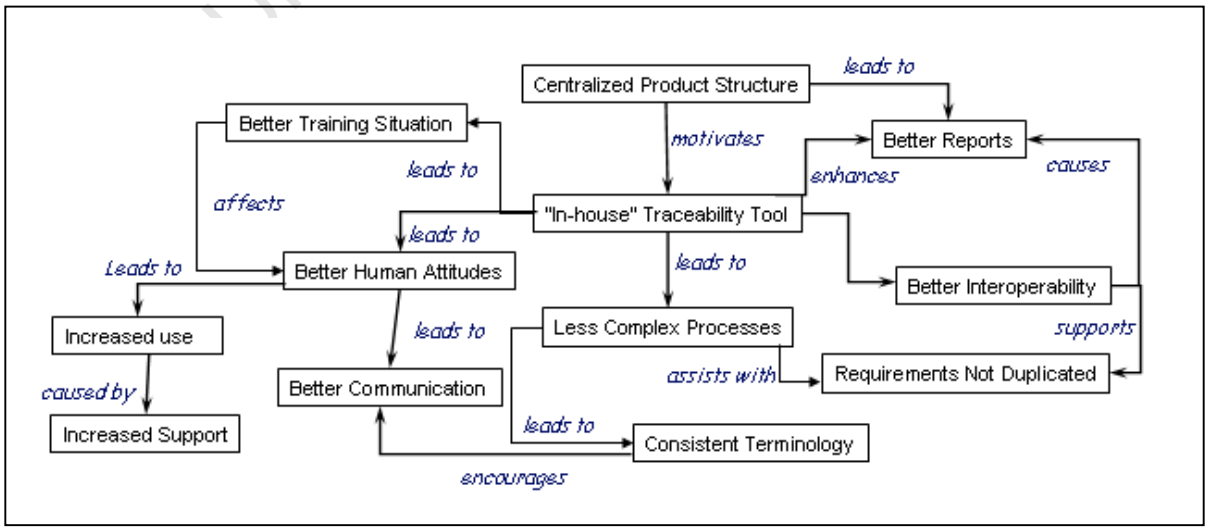



**Figure 10-17: Taxonomy of case Study Findings**

In Figure 10-17 above, *Taxonomy of Case Study Findings*, we illustrate the findings of the case study and the relationship between the factors that influenced traceability. The problems we encountered were not dissimilar to the problems that many organisations face. However, developing telecommunications enterprises is a very complex business. The complex telecom standards, the interrelationship with legacy systems, the number of interrelated product families, the magnitude of requirements, complexities brought about by major cross project and cross continent development in an ever changing market place makes development of these large enterprises difficult.

The first major change that heavily influenced traceability was the consolidating of the product structure from three separate products, with separate market units, product units and development units. This was made possible because of merging telecom standards, but also because of a consolidation strategy brought about by the poor economic climate of the early days of the 2000s. The impact of this change in product structure had a major influence on the geographical distribution of the project, going from 19 to three development houses. The dictum that traceability is easier to manage and control across fewer sites, though simplistic holds true. The ability to centralise the organisation of the product development team onto one site makes a huge difference to the success of traceability. Project meetings were more effective, seamless communication was reported, leading to better overall traceability practices.

The next big change to the traceability situation was the introduction of an in-house propriety tool. While corporate Ericsson invested heavily in the development of this tool the impact on local traceability practices was incredible. For the product development unit the tool synchronised with the stakeholder's requirements stored using Focal Point. The tool had better change management and impact analysis functionality, better interoperability between geographical separated sites, better reports, better support than its predecessor Requisite Pro. With the availability of such an environment the number of users increased from 20 to 250 users. Because the tool was propriety there was no restriction on formal or informal training practices or the free distribution of training material. On-line manuals and simple click and play tutorials were made available to all project participants. On-site trainers were identified to run shorter, carefully tailored courses for different audiences. The reduced cost and simpler course logistics lead to better management commitment to release staff for courses.

Another factor or key to success was the change in resource allocation for requirement engineering. In 2004, only one resource was allocated to requirement management. By 2007 four resources were available. In 2003, the requirement manager looked after management issues like the creation of the Requirement Management Plan, participation in change control processes and ensuring the requirement engineering processes met the needs of the users. There was little time for support, training or mentoring. In 2007, four resources were assigned to requirement engineering, the three extra resources providing support and personal mentoring on all aspects of the tool. We observed that the requirement manager was senior and attended the systems and product management meetings, interfaced with the configuration management team and all stakeholder related matters. Furthermore, the requirement manager worked closely with the process development team to ensure that the processes satisfy the needs of the organisation. The other resources had less experience and are best described as requirement engineers or requirement administrators working closely with the project team members. The approach of having one senior requirement manager and a number of requirement engineers worked well for Ericsson.

Another less important change that took place was the change to the process strategy. Between 2004 and 2007, Ericsson Unified Requirement Engineering (EUREP) was dropped. A process modelling expert was employed and in early 2008 the Requirement Engineering process moved from CMMI level 1 to CMMI level 3. The main factors that brought about this change in quality award was the introduction of the process modelling approach and better documentation of the roles and their corresponding relationships.

In conclusion, we believe that Ericsson's succeeded in improving their requirement engineering and traceability practices by returning to their proven approach from the eighties and nineties. They consolidated their product line leading to simpler organisational structures. They developed an in-house traceability tool to overcome the problems of interoperability, poor integration of all requirements into one solution space and lack of consistent reporting across all sites. The development of a propriety tool lead to many advantages especially in the rollout of training and on-site support. We firmly believe that the success of traceability can only be fully realised when there is regular training, better training and support documentation and better on-site mentoring. Mentoring is only made possible when there are more resources assigned to requirement engineering tasks, which can only come about with greater management commitment on the importance of requirement engineering and traceability.

## 10.7 THE FUTURE FOR OSS

Next-Generation Network (NGN) is a generic term used to describe the new IP (Internet Protocol) Networks that are emerging. With NGN one network transports voice, data and all sorts of media such as video using packet technology across the internet. The International Trade Union has set up a NGN Management Focus Group with a goal of organizing and undertaking a centralized approach regarding specification of NGN related Fault, Configuration, Accounting, Performance, and Security Management interfaces. A key element in the operation of the focus group is that it engages key individuals from various organizations with management expertise and specifications applicable to NGN to collaborate in and contribute to this activity. NGN will seriously impact the product lines that Ericsson approaches. Without question IP networks are the future and the management of this networks through OSS is perhaps the future direction for Ericsson.

Development on 4G is underway. The goal of 4G is to replace the entire core network with a single worldwide cellular network completely standardized and based to IP for video, packet data utilizing Voice over IP (VoIP) and multimedia services. The newly standardized networks will provide uniform video, voice, and data services to the cellular handset or handheld Internet appliance, based entirely on the Internet Protocol. In simple terms 4G will solve the problem: if a consumer can do it at home or in the office while wired to the Internet, that consumer must be able to do it wirelessly in a fully mobile environment.

So what does all this mean? The future for OSS's is new standards, new rules, new requirements, more demanding customers, new complexity, new challenges, new hardware and new systems. Without breaking confidentiality agreements in this paper we can only hypothesise what the future for OSS development and the challenges for traceability hold for Ericsson. We believe that for the next couple of years, fine tuning of the current proved methods and tools will continue. The next step will be automation of traceability links, brought about by new developments in structure languages leading to a sharp reduction in the manual approaches we have today. We are sure that complex organisational structures,

large quantities of requirements, complex change management and difficult requirement engineering and traceability challenges will continue for the foreseeable future. The future for traceability is unknown but we hope that this case study will assist at least with our understanding of how changes made both organisational and environmentally heavily influence the attitudes that software engineers have towards traceability and impact its success for the future.

# Chapter 11 TRACEABILITY FRAMEWORK VALIDATION & ASSESSMENT

## 11.1 INTRODUCTION

The purpose of the chapter is to report the results of the validation, test and assessment of the Traceability Framework, in both the laboratory and the field, in order to test the hypothesis that:

*"On the basis of data gathered from industrial sources and previous research efforts, traceability practices are in need of a structured, systematic and disciplined rule-based modelling approach to overcome the problems being encountered in the field"*

*And that "the package of the TRAM and TRAP models, plus the patterns provide a flexible basic package, easily adaptable to a wide range of users, with potential to overcome many of the problems in the field."*

The primary objective of this chapter is to validate this hypothesis by gathering evidence that supports the utility of the TRAM and TRAP models and to demonstrate that traceability patterns are a useful approach for describing certain aspects of traceability.

It is of primary importance for the reader to appreciate that the approach adopted for validation, test, assessment and evaluation was both exploratory and experimental. This chapter tells the story of how we moved from the stage of initial exploratory efforts in our 1st year of research, 2004, through the development of the TRAP and TRAM models and the traceability patterns.

This process involves a number of, sometimes conflicting, aims, between simplicity and conformance to standards, which must be resolved if a model or modelling approach is to be widely adopted. Firstly, models should provide a simplified solution to a complex problem. They are abstractions of real-world situations and designed to be *understandable, useable* and *manageable*. If they are going to be widely accepted then they must conform to widely accepted standards.

We followed a layered approach, with each layer describing a different abstraction or perspective. As illustrated in Figure 11-1 below, *Validation of the Model Layers*, a number of different validation techniques were employed. Although the OMG provide a solid foundation for creating models at different levels of abstraction, we argue that the specifications are weak with regard to validation, and do not address fundamental issues. For example, the metamodels (M2 Layer) are platform and domain independent. Therefore, the questions arise, who and when should the models be validated and what criteria should be used to verify their completeness, correctness or conformance to the standards? Because the M2 models are domain independent we decided that the validation should take place in a domain independent or *laboratory setting*, while the lower level models (M1 & M0) are defined from a users perspective therefore these models should be tested in the industrial context that they were defined. For example, carrying out cognitive walkthroughs and participatory workshops in Ericsson with personnel that were involved in the definition of

the models.

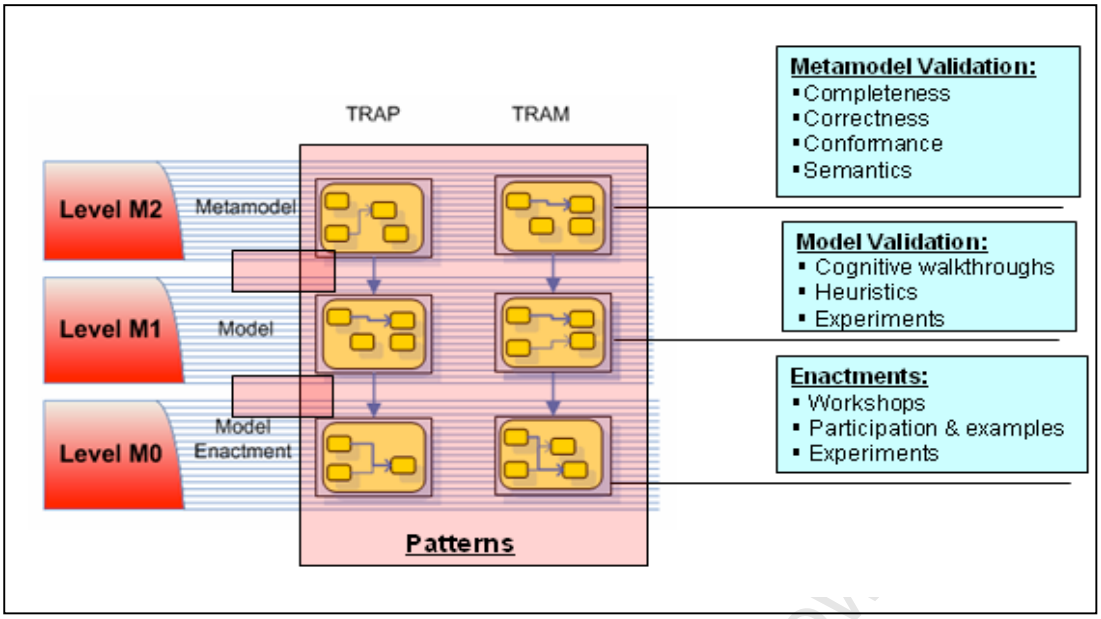



**Figure 11-1 Validation of the Model Layers**

Our evaluation sought answers to two simple questions from two different perspectives:

- Does the proposed solution address some of the problems faced by the traceability user community in small, medium and large organisations?
- Does the proposed solution offer reusable components that be further developed in future generations of research?

Our evaluation of the value of the models is built up from both perspectives. In Section 11.7 we describe the laboratory and field tests that we carried out on the Patterns. Finally, in the last section, 11.8, we conclude with a critical assessment of all the evidence from the Test and Evaluation work, both in the lab and the field, in order to see whether the hypothesis has been proved *in principle*.

## 11.2 TEST & EVALUATION METHODS

### 11.2.1 Basic Approach

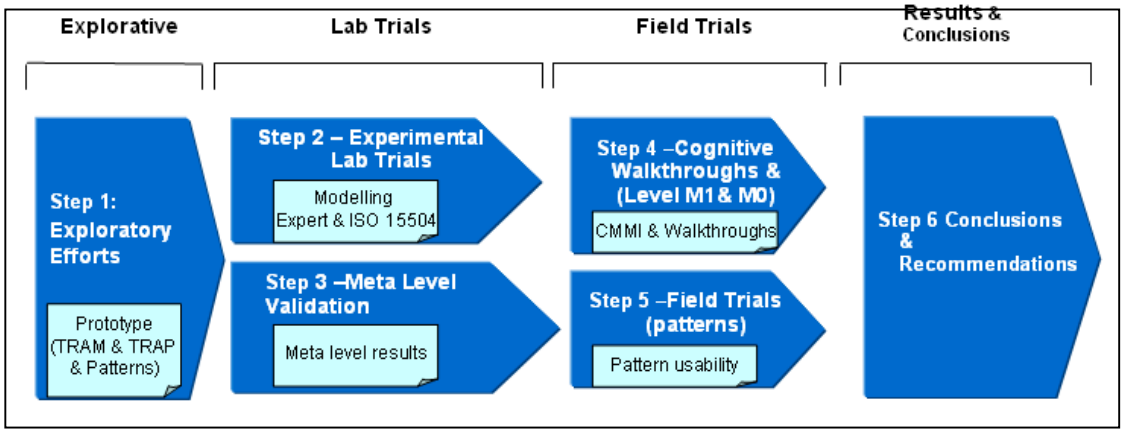



**Figure 11-2: Test & Evaluation Approach**

In Figure 11-2 above, *Test & Evaluation Approach*, we illustrate the test and evaluation approach that we followed from the initial exploratory efforts, through a series of lab and field trials. While, the above method is portrayed as a simple sequence, many of these validation steps were iterative, and in some cases conducted in parallel. This is especially true in steps three, four and five, where validation results required adjustments to the models or a pattern before another validation exercise was attempted. The four phases are briefly described below as:

- *Explorative:* Explorative learning and development plays a major role when users are creating new concepts. Therefore, we used exploration environments to support us with self-directed learning. At the end of this phase, we reported the results of our initial findings to both academicians and practitioners using their feedback to give direction for the next stage of the development.

- Lab Trials: These involved validation work undertaken in the laboratories at the University of Cape Town. When industrial reviewers were required they generally visited the university to carry out the reviews. There were two main activities or type of lab trials undertaken as follows:

  - Experimental Lab Trials: The main experimental lab trial was to assess the TRAP process framework using the ISO 15504 framework. The main objective of this experiment was to benchmark the capabilities of TRAP against those of the Rational Unified Process (RUP).

  - Meta-Level Validation: This consisted of a number of different validations and verification workshops that were undertaken to evaluate the TRAM and TRAP M2 models.

- Field Trials: The work proposed here turns to the user. The phase explores the concepts of usability and qualities of the framework in use, and their relationship to end-users learning to use the framework, in a case study approach. A summary of the test components can be found in Table 11-1 below, *Testable Components*.

| Testable Component | Type of Tests |
|---|---|
| TRAM & TRAP Level M2 | Laboratory based testing, for example conformance and correctness against the standard, and extensibility testing. |
| TRAM & TRAP Level M1 &M0 | Cognitive walkthroughs with the user community, testing the components under the criteria, usability, understandability, completeness and confidence. |
| Patterns | Executing laboratory and field trials and assessing the patterns for learnability, usability, understandability and structuredness. |

**Table 11-1: Testable Components**

### 11.2.2 Test Criteria, Testable Requirements, and Methods

In this section we briefly describe the basic testing method used throughout the lab and field trials. A fundamental objective of testing is to establish that the properties of the solution possess a particular required characteristic. In this work it is useful to think of models and patterns as being very similar in kind. There are three recurring test questions:

1. Is the model or pattern *valid?*
2. Is the model or pattern implemented in a true and *verified* manner?
3. Is the model or pattern *useful*?

*Validation* of models is an absolute essential if it is to be widely accepted. Because models are a simplification and abstraction of reality, modellers look for three properties when testing namely:

- Completeness: A validation assessment must determine if the model represents all of the properties that the user requires to pursue traceability: the traceable artefacts, the key relationships and dependencies, the work flow and the roles and rules.
- Correctness: A validation assessment must identify if the models representation of these properties matches sufficiently all aspects of traceability behaviour to adequately serve the different users.
- Confidence: A validation assessment must explicitly quantify the confidence that the user can place in the applicability of the model.

We will discuss these criteria in more detail in Section 11.5.3, Walkthrough of Profiles.

*Verification* of models is concerned with proving that the model or pattern is implemented in the intended manner: that is, it is programmed in the correct manner and that the algorithms have been implemented properly. This includes both meeting the model designer's intention and that of conformance with the standard.

In principle, one would expect the Traceability Framework to comply with most or all aspects of OMG's standards, because:

- The models in the traceability framework are based on the UML 2.0 specification, the Meta-Object Facility (MOF) 1.4 and the principles of Model Driven Architecture defined by the OMG.
- The TRAM and TRAP models follow the principles of UML Profiles as defined in the UML infrastructure volume of UML 2.0.

However, because the TRAM and TRAP models made extensive use of extension mechanisms in UML, we thought it wise for modellers with expertise in UML and OMG specifications to assess the models for conformance, using a simple rating metric: *good, fair* or *poor.*

Another important modelling consideration arises out of the desire that TRAM and TRAP should not be a complex monolithic model. The hope is that traceability practitioners will, adapt the versions of TRAM and TRAP presented here and develop them further to suit their own needs. For this reason *adaptability* and *flexibility* were considered a very important property of the TRAM and TRAP models from a *reusability* standpoint.

Consequently, the test and evaluation sessions have attached considerable importance to assessing two further properties:

▪ ***Adaptability/Reusability*:** This requires consideration of whether the UML extension mechanisms employed in TRAM and TRAP can be further extended, adapted or simplified for different degrees of rigour in traceability practice. To remain consistent with the above evaluation metrics we also use *good, fair* and *poor* to describe the results obtained.

▪ ***Expressiveness***: This means the ability of the models to cope with more fundamental changes, but these are more a function of the flexibility of the UML modelling language itself. UML provides extensions to the language and allows the creation of new model elements through the use of stereotypes, which make it inherently expressive. As before we have adopted a simple rating scheme of good, fair, poor, but our results show that too much expressiveness can be a bad thing if it becomes incomprehensible.

The ultimate test for any model, however, is, is it *useful*? That is the most fundamental question that one can ask in any evaluation. If a model is useful then it has value. A key consideration in usability, arguably the most important factor, is the ease with which the user can understand the model. If a model is too complex for a potential user to understand, then the user is likely to reject it as too complicated. In seeking to evolve a widely useable model, one must therefore attach great weight to *understandability*. A model element rated as Poor for *understandability* is almost surely headed for major modification or outright rejection, so it is a vital criterion for model improvement.

As illustrated in Table 11-12, *The Requirements and Test Questions*, the basic approach, therefore, was to give the modellers and users alike a set of basic questions. In general, users are more concerned about the questions under the Validation column, whereas modellers are concerned with both Validation and Verification. These questions were put to users and modellers in guided inspection sessions in both the lab and the field. Guided inspection is an inspection or review technique carried out by a human that provides a detailed examination of a design or in our case a model, which is "guided" by test cases. The use of test cases meant that the inspection process could address more than just the syntax of the model being reviewed; in addition, we could address the *understandability* and *usability* of the models or patterns. The test cases come from test plans that are a required part of any software development process.

| Requirement Name | Brief Requirement Description | Test Question Validation | Test Question Verification |
|---|---|---|---|
| Req. 1: Empirical Data | This requirement ensures the solution framework satisfies the problems discovered during the Case Study and Survey | Does the solution framework satisfy the problems identified during the case study and industrial survey? | Does the framework meet its intended requirements in terms of the methods employed and the results obtained? |
| Req. 2 TRAM: Traceability Concepts/ Semantics | To capture the main concepts of traceability in a semantic data model, for example traceability items, types and relationships. | Does TRAM encapsulate all the main concepts as used by a traceability user? Would you use the TRAM in your organisation? | Do the models conform to underlying standards that it was designed against? |
| Req. 3 TRAP: | The TRAP has the | Does the TRAP | Does the TRAP |

| | | | |
|---|---|---|---|
| Traceability Process | fundamental objective of describing and modelling a software traceability process. It includes the phases, the engineering roles, the activities and any element necessary to describe a traceability process. | capture the main process components to describe a traceability process in a real world situation? Would you use the TRAP in your organisation? | conform to the underlying standards it was designed against? |
| Req. 4 Adaptable & Reusable | This requirement investigates whether the framework is adaptable to real world scenarios. | Is the framework both reusable and adaptable for real world scenarios and future work by research communities? | Is the framework adaptable and reusable using the extension mechanism as defined by the underlying standards? |
| Req. 5: Patterns Empirical Patterns | We investigate empirical patterns in real-world situations | Are the patterns understandable, reusable and adaptable for real-world scenarios? Are the patterns easily understood and do they provide benefits to an organisation interested in improving traceability practices? | Do the patterns conform to widely accepted pattern practices? |
| Req. 6 Model Patterns | We investigate if the patterns that emerge from the TRAM and TRAP are correct and complete. | Does the model represent and correctly reproduce the behaviours of the real world situation when implementing traceability?<br>Would you use the patterns in one of your organisation? | Do they conform to agreed model based patterns? |

**Table 11-2 The Requirements and Test Questions**

Each Test Case was presented in a structured format as in Table 2 below, Test Case. Testing ascertains the capability of a product by answering a series of questions. The test objective is a named element describing what property the thing under test is expected to possess in order to pass or be rejected. A test case is a specification of one case to test the

system, including what to test with, under which conditions, with which input and which desired result.

| | |
|---|---|
| **Test Case Name (or ID)** | Name or identified |
| **Test Case Objective** | Verify that a certain condition has been met. |
| **Test Type** | The type of test that was executed, for example whether it was a test for completeness or an empirical based test. |
| **Test Background** | This field describes the background to the test execution, for example the context of the test situation. |
| **Pre-condition** | The state of the system, or the activities the user must carry out before executing the test. For example, the user must review the M2 models before testing an M1 or Mo model. |
| **Input** | The input into the test case, for example, the user selects something or enters something to start test execution. |
| **Results** | The results obtained, what impact the results had on the solution or changes made to overcome the identified problems. |
| **Observation of Researcher** | Any key observations made by the researcher. |

**Table 11-3: Test Case**

As this is project researching traceability, we demonstrate in Figure 11-4, *Requirement to Test Traceability Tree*, the relationship between the requirements and the test cases.

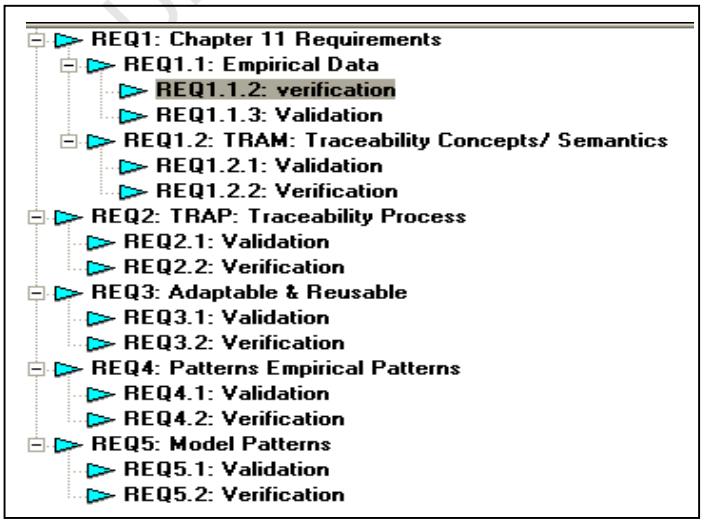



**Figure 11-3: Requirement to Test Traceability Tree**

## 11.3 STEP 1- EXPLORATIVE DESIGN & ASSESSMENT

### 11.3.1 Summary of Activities

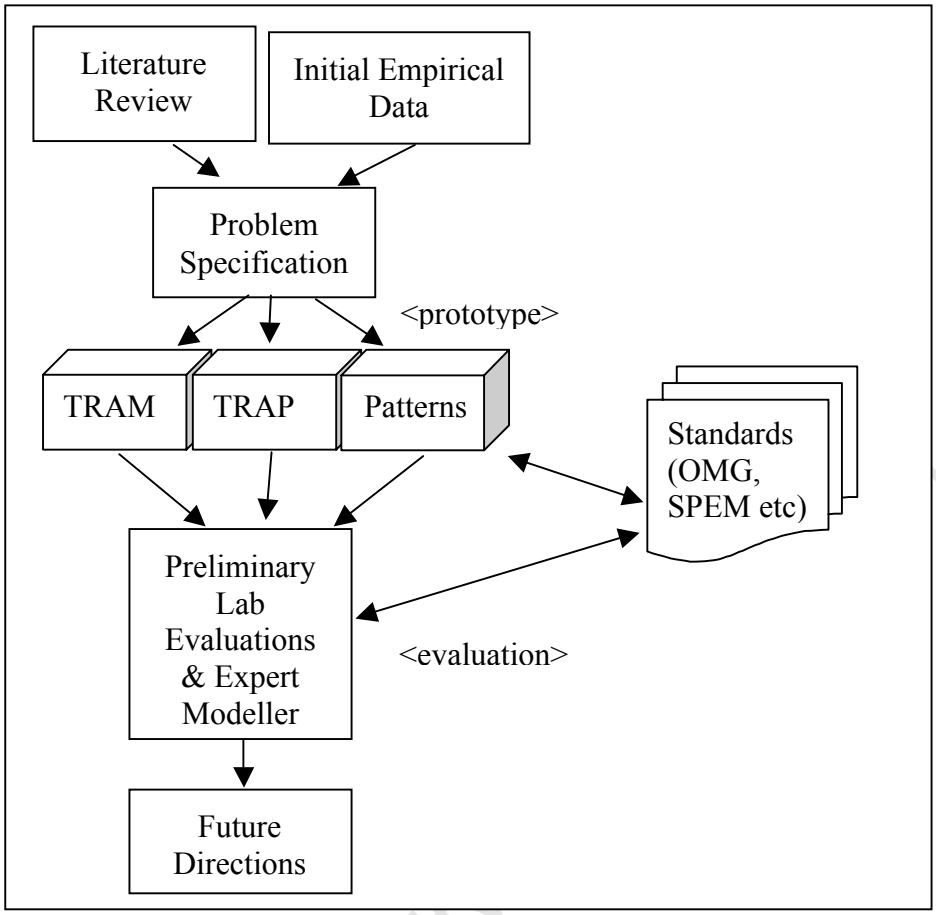



**Figure 11-4: Explorative Design and Assessment**

### 11.3.2 Exploratory Work in 2004

During the period 2004-2005, the author began several exploratory pilot projects with the aid of graduate[13] and post-graduate students[14] at the University of Cape Town. These were conducted in the offices and laboratories at the University of Cape Town. In Figure 11-4 above, Explorative Design and Assessment, we illustrate a workflow diagram of the explorative design and assessment activities. In Table 11-4 below, *Explorative Activities and Outcomes*, we demonstrate the main activities, the inputs to the activities, the resources involved and the outcomes from each activity.

[13] A number of final year projects were completed. One project created a prototype of the TRAPT, while another investigated the implementation of traceability using Service Oriented Architectures. On evaluation the use of SOA's for creating traceability services using Web Services was later discarded due to lack of suitability.

[14] Working collaboratively with a Master's student from the Royal Institute in Stockholm, Sweden, we created prototypes of TRAM and TRAP and began explorations into Patterns.

| Activity | Inputs | Who was Involved | Outcome |
|---|---|---|---|
| TRAM & TRAP Metamodel | Initial Empirical Data, Literature, OMG Specifications | Researcher & Modelling Expert | Evaluated Level M2 Prototype & lessons learned |
| Create Pattern Template | Pattern Literature | Researcher | Template & Template Model & lessons learned |
| Traceability Patterns | Empirical data, literature | Researcher & postgraduate student & OSS-RC Requirement Manager | Initial Simple pattern examples & lessons learned |
| TRAPT | Pattern tool survey, pattern literature | Researcher & undergraduate student | Prototype TRAPT tool |

**Table 11-4: The Explorative Activities and Outcomes**



### 11.3.3 Lessons Learned

In Table 11-5, *Lessons Learned from Explorative Activities*, we briefly describe the lessons learned during the explorative stage:

| Lesson | Brief Description |
|---|---|
| TRAM and TRAP Lessons Learned | ▪ Too many model elements used<br>▪ The models were too complex<br>▪ Users who did not have any knowledge of traceability concepts found the models difficult to understand. |
| Template Lesson | ▪ More than one template was required. For example, a different template for capturing the survey information was needed in comparison to the template for the model based patterns. The template for the survey had to capture the survey information. Furthermore, implications on the solution was an important criteria for each pattern represented because it illustrates how the findings impacted our decision with the solution. |
| Patterns Lesson | ▪ Initial results showed that the patterns helped communicate complex traceability concepts in easy to understand and easy to use approach.<br>▪ Different classifications of traceability |

| | |
|---|---|
| | patterns was required<br>▪ Further work was needed on evaluating patterns.<br>▪ A process for capturing and evaluating emerging patterns was required. |
| TRAPT Lesson | ▪ The tool needed more information on generic patterns<br>▪ Further developments on modelling patterns were required. |

**Table 11-5: Lessons Learned from Explorative Activities**

## 11.4 STEPS 2 & 3 - LAB TRIALS

### 11.4.1 SUMMARY OF ACTIVITIES

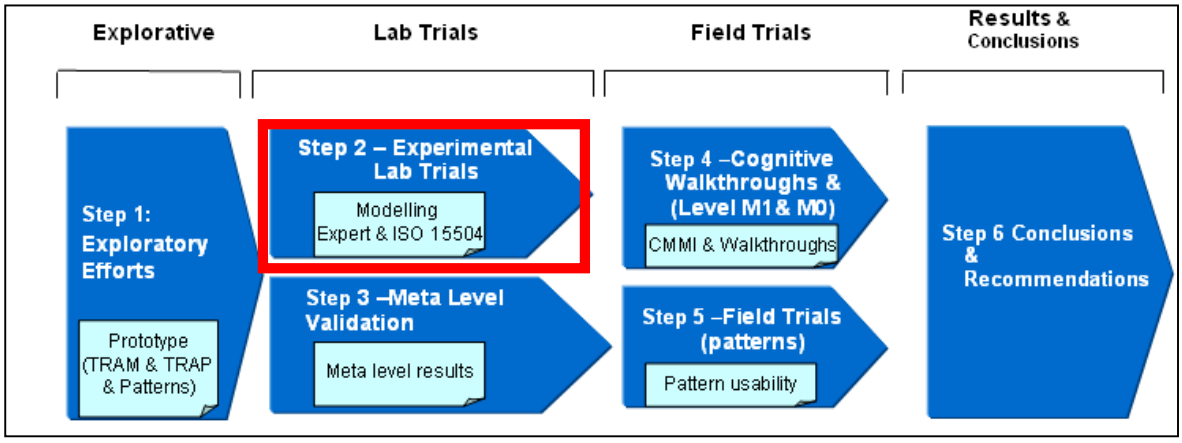



**Figure 11-5: Assessment Method**

In Figure 11-5 above, *Assessment Method*, we illustrate where in the assessment method this section addresses.

At this stage two different laboratory trials were executed. The first was to compare the capabilities of TRAP against the capabilities of a commercially available process, the Rational Unified Process, using the ISO 15504 assessment framework. Secondly, we assess the meta-level models using experimental techniques that we describe later.

### 11.4.2 Step 2: Benchmarking of RUP & TRAP to ISO 15504[15]

In late 2004, we evaluated that an early assessment of TRAP was needed to give us an indication of the capabilities of the TRAP and to assist us with its future direction. At this stage TRAP consisted of M2 models and a number of traceability workflow diagrams that defined roles, responsibilities, phases and traceability diagrams. The primary objective

[15] This results from this work were presented at EuroSPI in 2005 (KELLEHER, J. (2005a) A Method for Modelling a Mature Process. *Software Process Improvement · 12th European Conference, EuroSPI 2005.* Budapest, Hungary, Springer.)

of this assessment was threefold. A primary consideration of this early evaluation was to gain an understanding of what an international standard specified about traceability. For example, what traceability practices does ISO 15504 recommend? An understanding of this would help us greatly in further designs of the TRAP. Secondly, we needed to understand how to carry out an assessment, of the TRAP but also how to assess the TRAM and the traceability patterns in future assessments. On analysis of the ISO 15504, it was evident that this standard would give us a framework for carrying out assessments. Finally, to gain an understanding of the capabilities of the TRAP by benchmarking it against the capabilities of the Rational Unified Process (RUP). We evaluated that RUP was a suitable framework to compare TRAP against, because it defines traceability workflows, it uses SPEM as the underlying process metamodel (Bencomo, 2005) and that we had access to the process and tools through the Rational University Program.[16] In Figure 11-6 below, *RUP metamodel*, we illustrate an extract of the RUP metamodel which is based on SPEM.

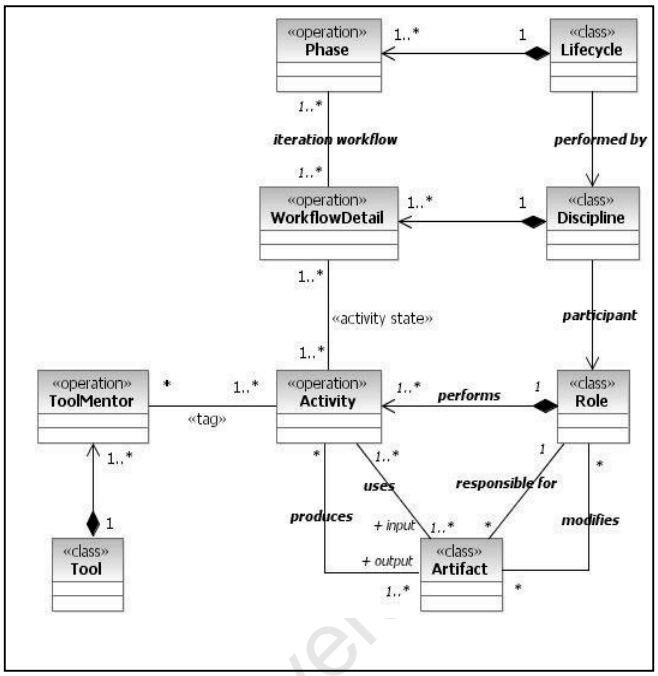



**Figure 11-6: RUP Metamodel**

We selected ISO 15504 as the evaluation standard for the following reasons:

- It encourages self-assessment. While we used an independent assessor in the final assessment we learned how to carry out preliminary self assessments in a laboratory setting which gave us an understanding of how to carry out an assessment.
- It produces a set of process ratings rather than a pass/fail result. This is essential when comparing two processes.
- It addresses the adequacy of the management of the assessed processes.
- It takes into account the context in which the assessed processes operate.
- It is appropriate across all application domains and sizes of organization.

The ISO/IEC 15504, is sometimes referred to as SPICE (Software Process Improvement and Capability dEtermination) is a process assessment framework developed by the Joint Technical Subcommittee between ISO (International Organization for

[16] This program also provided us with licences for the traceability tool, Requisite Pro

Standardization) [17] and IEC (International Electrotechnical Commission). ISO 15504 is a process framework that consists of a process reference model and a process assessment model. The Technical Report (TR) document for ISO/IEC 15504 is divided into 9 parts (separate documents). However, as shown in Figure 5 below, ISO 15504 (Part 2 & 5), there are two core parts namely; ISO 15504-2 (Part 2) and ISO 15504-5 (Part 5) (ISO/IEC, 1998 )

The ISO 15504-2 defines a reference model that has two main dimensions: the process dimension and the process capability dimension. The process dimension is sub-divided into five sub-processes called process categories namely; the customer-supplier, engineering, supporting, management and organization. To contextualize this to our study the five sub-processes would collectively make up the Product Development Lifecycle (PDLC). Inside each process it defines base practices and management practices.

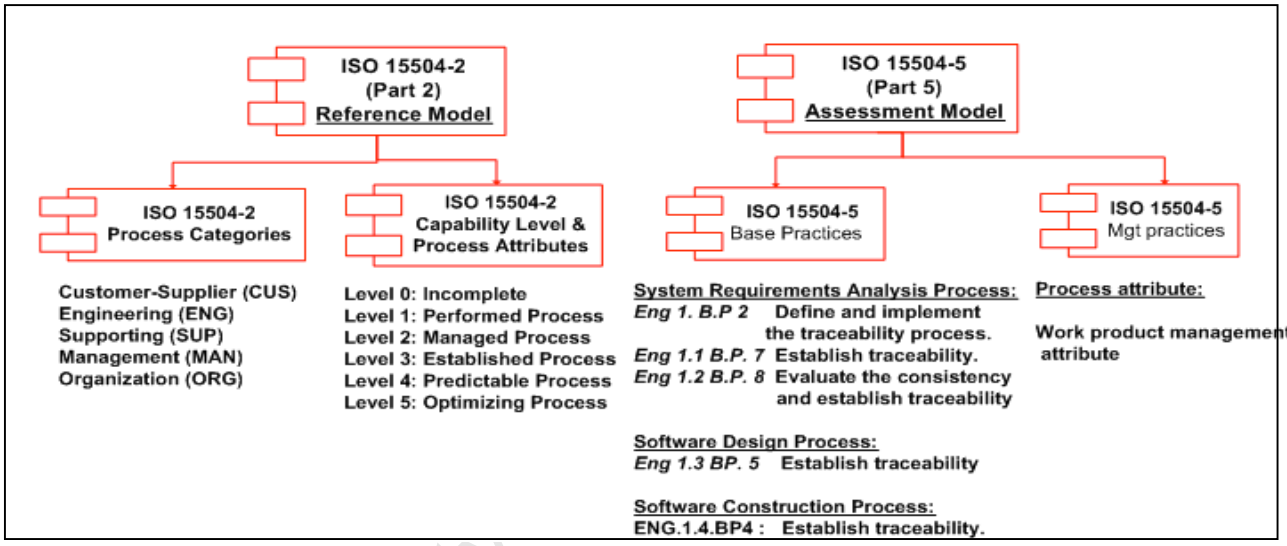



**Figure 11-7: ISO 15504 (Part 2 & 5)**

Each of the processes defined by ISO 15504 (customer-supplier, engineering, supporting, management and organization) has a process capability level as follows:

- Level 0: Incomplete
- Level 1: Performed Process
- Level 2: Managed Process
- Level 3: Established Process

[17] ISO standards are developed according to the following principles: 1. *Consensus*: The views of all interests are taken into account: manufacturers, vendors and users, consumer groups, testing laboratories, governments, engineering professions and research organizations. 2: Industry wide: Global solutions to satisfy industries and customers worldwide. 3. Voluntary: International standardization is market driven and therefore based on voluntary involvement of all interests in the market-place. To date, ISO's work has resulted in over 16 000 International Standards, representing more than 620 000 pages in English and French (terminology is often provided in other languages as well).(ISO/IEC (2007a) How are ISO standards developed?)

- Level 4: Predictable Process
- Level 5: Optimizing Process.

The process capability is determined by measuring the nine process attributes which are: Process Performance, Performance Management, Work Product Management, Process Definition, Process Deployment, Process Measurement, Process Control, Process Innovation, and Process Optimization. To assess your process or organisation you must assess each of the nine attributes above against the commonly called four point N-P-L-F rating scale:

- Not achieved (0 - 15%)
- Partially achieved (>15% - 50%)
- Largely achieved (>50%- 85%)
- Fully achieved (>85% - 100%).

The ISO 15504-5 (part 5) describes the assessment model. It describes a Base Practice (BP) as a software engineering or management activity that, when consistently performed, contributes to achieve the purpose of a particular process. A management practice is a management activity or task that addresses the implementation or institutionalization of a specific process attribute.

The ISO 15504 document suite also provides a set of categories in which the assessors can place the data that they collect during their assessment. The result is that the assessors can give an overall determination of the process capabilities or in our case to benchmark TRAP against the capabilities of RUP.



### 11.4.2.1 The Assessment Method

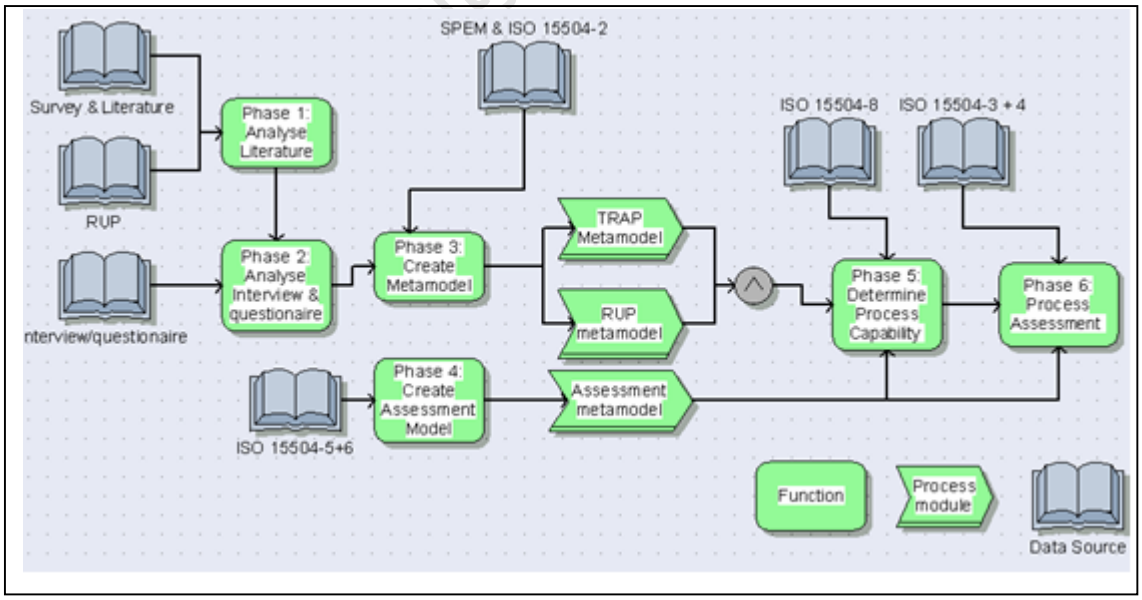



**Figure 11-8: Assessment Method**

In Figure 11-8, *Assessment Method*, we illustrate the steps or phases that were followed, the input artefacts and the outputs from each step. At Phase 2, we review the initial interview and questionnaires that were gathered in Ericsson. We cross checked the

problems that emerged with the objectives of the TRAP models. This ensured that the models were addressing the problems. At Phase 3, we created a model of the RUP traceability workflow, including the roles, work products, activities, guidelines and responsibilities. At Phase 5 we assess the capabilities of the TRAP against the capabilities of the RUP. The inputs to each phase and the outcomes are described in Table 11-6, Inputs and Outputs from Each Phase.

| Development Stage | Data Inputs | Outputs |
|---|---|---|
| Phase 1: Review Literature | ACM, IEEE, Academic research focus groups, experience documentation, ISO, Carnegie Mellon etc | Repository of reusable literature or best practices for TRAP |
| Phase 2: Interview + Questionnaires: | Review the interview and questionnaire from focus groups on process modelling and software traceability | Identify critical traceability problems and create document list for TRAP process blueprint |
| Phase 3: Create TRAP + RUP Metamodel: | The RUP + TRAP process metamodel ISO TR 15504 Software Process Engineering Metamodel (SPEM) | RUP, and TRAP metamodel (work products, activities, guidelines, work flows, best practices) |
| Phase 4 Create Assessment Model: | ISO 15504 Software Process Engineering Metamodel (SPEM) | TRAP and RUP process capability and maturity assessment report |
| Phase 5: | Questionnaire | TRAP and RUP process capability and maturity assessment report |
| Phase 6: | Review and validation of work with focus group | Approval report |

**Table 11-6 Inputs and Outputs for Each Phase**

In Figure 11-9, *TRAP RUP Assessment*, we describe the assessment method that we followed with the assessor. The assessment steps were as follows:

- *Identify a suitable assessor*. One of the reasons that we decided to use the ISO 15504 was that we had access to an independent assessor in Cape Town.
- *Review the assessment input*. Before starting the assessment we review the TRAP M2 models and the workflow diagrams, and review the RUP metamodel, especially the traceability workflows.
- *Select the process instances*, for example, we select the *software design and implementation capabilities*, the *customer process capabilities* and the *engineering process capabilities* as the processes we are assessing TRAP against.
- *Determine the actual ratings*, by giving a capability level for each process instance.
- *Validate the ratings*. Recheck and validate that the results are a true reflection.

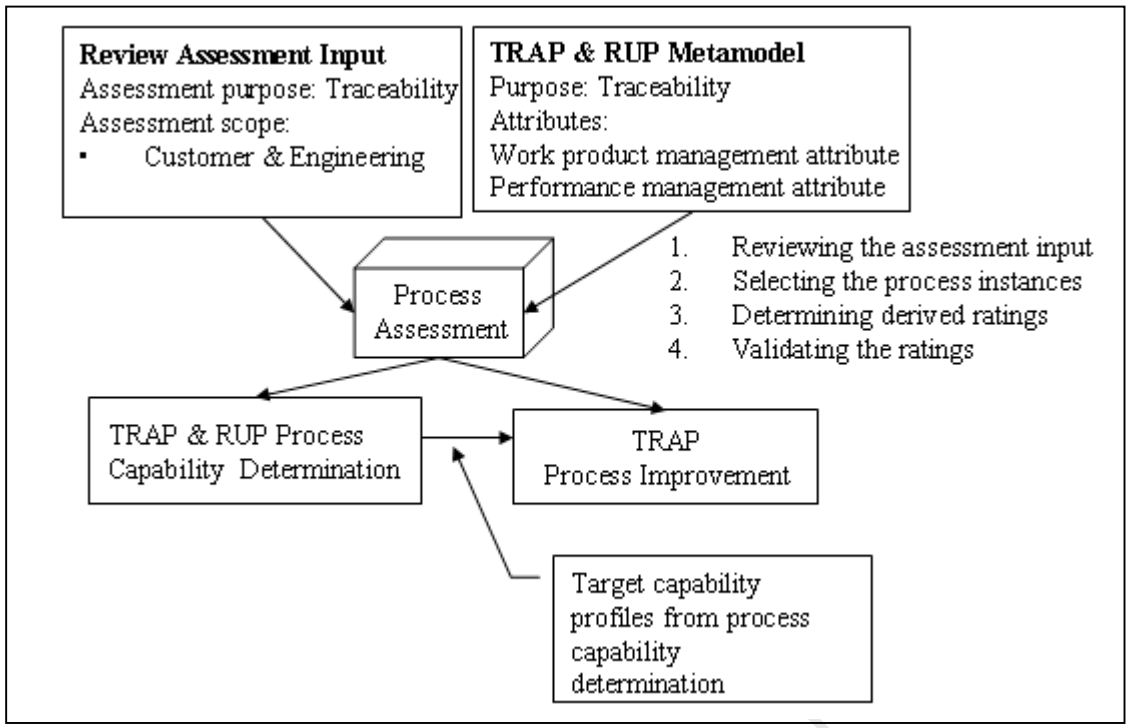



**Figure 11-9 TRAP RUP Assessment**

### 11.4.2.2 Results of Assessment

In all it took three hours to carry out the assessment. The only attendants were the researcher and the independent assessor. We assessed the capabilities of TRAP in the following processes, *software design and implementation, customer acquisition* and *requirement elicitation.* The results were as follows:

**Level 1** *The software design and implementation processes.* The assessor assessed the capabilities of TRAP and its traceability capabilities at the design and implementation level, to be at Level 1. This was a somewhat disappointing assessment result; however, it did give us a good indicator of an area for improvement for future developments in TRAP. While the TRAP possessed the ability to show traceability relationships between the different artifacts (which are called work products in ISO 15504), it did not sufficiently capture the different types of design artifacts. This was a continual weakness of TRAP. It is very difficult to capture traceability between design artifacts especially if it is between design elements and code. The assessor concluded that further work on TRAP was needed to improve the description of "what" to trace at the design and implementation level.

**Level 2** The customer acquisition and preparation process and the engineering process for the integration and testing of the software were determined at the capability Level 2. The TRAP process described the requirement types (for example, functional and non-functional) as traceability item types, which was one of the necessary ISO assessment criteria and it also described how to control the traceability items, create the dependencies and how to control change. The assessor commented that the models and the underlying processes described change sufficiently.

**Level 3** The *requirement elicitation process*, the *architectural requirement process* and the *software requirements process* were determined as Level 3. TRAP satisfied the work product (artifacts) management attribute and also the process resource attribute. TRAP

described the roles involved in software traceability, their corresponding responsibilities required for performing the traceability process.

The results for the RUP capability determination were:

**Level 1** *The customer acquisition and preparation process and the software design and implementation processes*. The customer acquisition process is poorly defined in RUP. However, the process performance attribute that the process transforms the identifiable input work products to produce identifiable output work products was true.

**Level 2** The customer requirements elicitation process, the system and software requirements process and the integration and test process were determined to have a Level 2 rating. We determined that the requirements and testing discipline are the two most mature process disciplines in RUP. The integration between the requirement management and test management environments was taken into consideration in its process rating.

Overall traceability is a poorly defined from an end to end perspective in RUP. For example, RUP describes the management of traceability dependencies in the requirements discipline but omits this practice in the business modelling discipline. In Figure 11-10 below, *TRAP versus RUP Capabilities*, we compare the maturity levels of the TRAP and the RUP.


**Figure 11-10 TRAP versus RUP Capabilities**

### 11.4.2.3 Lessons Learned from using ISO 15504 to evaluate TRAP

Firstly, applying an assessment framework to a "work in progress" is generally not a good practice as the results do not give a true reflection of the capabilities of the final process. However, this was not the main goal of the assessment at this stage. The primary objective of using ISO 15504 was to gain an understanding of how to assess a process, or in fact how to assess any component of the Traceability Framework using an international standard. In this respect using ISO 15504 was a very useful activity.

The immediate problem that you face when using ISO 15504 is the sheer magnitude or size of the framework. It consists of nine parts, described in hundreds of pages, which took the researcher many weeks to become familiar with the content. Furthermore, some aspects of ISO 15504, are more useful than others. As previously stated, it clearly defines

how to carry out an assessment. This aspect alone helped us to achieve a consistent method of assessing other aspects of our solution, be it during the cognitive walkthroughs or participatory workshops. Furthermore, it recommends traceability practices which assisted with future developments of the TRAP and the TRAM. For example, Base Practice seven dictates to establish traceability between the customer needs and the system requirements, while Base Practice eight instructs the user to evaluate the consistency between the software requirements and system requirements. This extra information greatly benefited our overall understanding of traceability.

The problem with ISO 15504 is the magnitude of information that it contains. The number of processes defined from customer acquisition, to requirement elicitation to software design and implementation and the large number of base practices is daunting. However, its purpose is to describe all processes in the product development lifecycle and in so doing provide an assessment framework. ISO 15504 certainly meets this objective.

Furthermore, gaining a better understanding of the capabilities of RUP, gave us an insight of the short-comings of commercial processes. One is left wondering, how a process that is so widely used and supported by a tool (Requisite Pro) could define traceability so poorly across its processes. The reality is that RUP fails to describe many of the important aspects of traceability. RUP only defines traceability responsibilities to a number of its roles. For example, it describes a traceability workflow in the requirement discipline and yet fails to mention it in the design discipline. This gives the wrong message to designers that traceability is not an important practice that they should carry out. Unquestionably, in the nineties, Rational made a major contribution to the requirement engineering discipline. They employed some of the greatest methodologist who standardised many of the requirement engineering practices that are common place; however, after researching traceability it is easy to see that RUP has many flaws.

While this assessment was carried out very early into the design of TRAP we did demonstrate that TRAP had good capabilities when compared to a commercial product. This insight assisted us to decide to continue with the development of TRAP. It also helped us with the scope of the TRAP and areas for future improvements. For example, overcoming the problems of traceability between design elements and implemented code was eventually reasoned to be outside the scope of this project. Overall, the ISO assessor gave us positive feedback on TRAP which also aided us to make the decision to continue this endeavour. The assessor also made it clear that the process framework would achieve a higher rating after further process instantiations were added especially from the Ericsson's project domain.

Would we recommend that traceability researchers use ISO 15504 to assess any aspect of a traceability process? The answer is "maybe", depending on what they want to achieve. If the researcher wants to gain a deeper understanding of traceability practices in the develop lifecycle, then ISO provides a perfect framework to gain this knowledge. Furthermore, if the researcher, like in the context of this study, wanted to gain experience in critiquing and assessing process models then ISO 15504 also provides excellent information on how to do so. However, if your objective is to compare the capabilities of your models against the capabilities of other process models, then this assessment framework is too complex and does not merit the effort that it takes. Overall, ISO 15504 is a complex framework.

Finally, traceability communities, in particular the Centre of Excellence for Traceability, emphasise the need for benchmarking and metrics on aspects of traceability.

We believe that a research group could adopt the ISO 15504 assessment framework, for assessing and benchmarking certain aspects of traceability. It comes with an in-built metric system and standard traceability practices with further research efforts added to the framework.

## 11.5 STEP 3- META-LEVEL VALIDATION

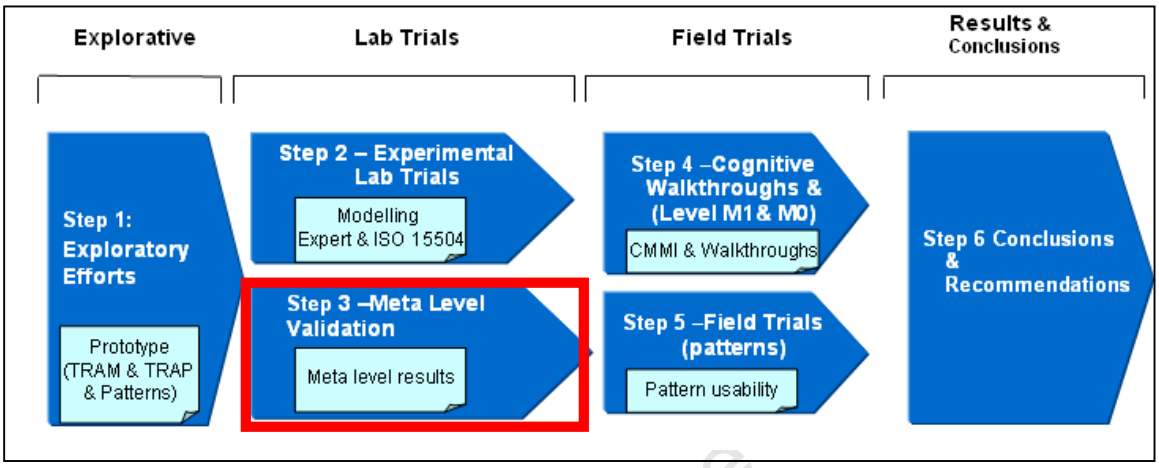



**Figure 11-11: Review of Assessment Method**

In Figure 11-11, *Review of Assessment Method*, we illustrate that we are addressing the Meta-Level Validation in this section. The steps involved in the evaluation of the M2 or meta-level models, are shown in Figure 11-12 below, *M2 & M1Evaluation Process.*

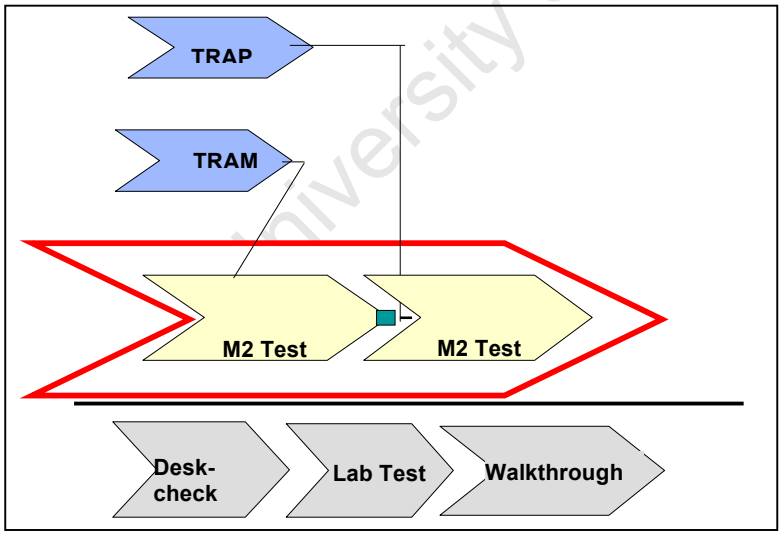



**Figure 11-12: M2 & M1Evaluation Process**

### 11.5.1 Deskcheck

In Table 11-7 below, *Deskcheck*, we describe the deskchecks process and the results obtained in table format.

| Test Type | Deskcheck |
|---|---|
| **Test Objective** | Verify that a models are syntactically correct |
| **Test Type** | Deskcheck |
| **Location** | University of Cape Town |
| **Background** | In 2004, we carried out a number of deskchecks during the early prototyping of the M2 TRAM and TRAP models. These simple reviews were carried out with research lab members at graduate and postgraduate levels. The researcher sent the models to the reviewer's who read the models and sent back with defects and comments. This type of informal review was useful during the early stages of the design. |
| **Number of Reviewers** | 6 undergraduates & 4 postgraduates |
| **Pre-condition** | The models had already been reviewed by the author. |
| **Input** | The TRAM and TRAP M2 metamodels |
| **Output** | Approved models |
| **The Results** | On the first initial deskcheck the semantic data model (TRAM) and the process model (TRAP) were integrated into the one model. It contained 30 model elements and the initial feedback from the reviewers was that two separate models should be created; one for process and one for semantics because the current models were too complex.<br><br>Another main outcome was the decision to use UML Profiles to extend UML rather than following a MOF metamodel approach. This decision was proposed by a postgraduate student carrying out research in UML. The UML Profile gave us more flexibility to extend UML for the many model elements not supported by the language. [18] |

**Table 11-7: Deskcheck**



[18] Staron states that creating a profile is making the existing, standard modeling language more suitable for a specific context and adaptable to particular purposes.

### 11.5.2 Lab Inspections (Model Checking)

As shown by Table 11-10, *Model Checking*, we carried out a number of assessments and model checks, in 2004, at the University of Cape Town. The primary objective of these modelling checking workshops was to verify the models and get the input from a number of sources on the modelling approach including modelling experts.

| Test Type | Lab Inspection |
|---|---|
| **Test Objective** | The goal of the inspection was to run a moderated inspection meeting, repairing any defects so that everyone on the inspection team could approve the models. |
| **Test Type** | Main test types were: Conformance, Completeness, Correctness |
| **Location** | University of Cape Town |
| **Background** | In 2004, we carried out a number of formal inspections. The moderator distributed a printed version of the models to each inspector, along with a checklist to aid in the review. At the start of the inspections, the moderator verifies that each team member is prepared. The moderator walks through each model and the inspectors indicate where there are defects. Each defect was either resolved or left as an open issue. The moderator adds each defect to the inspection log. The author repairs the defects identified in the inspection meeting. At the next inspection meeting the reviewer's verify that the defects were repaired. |
| **Number of Reviewers** | 4 undergraduates & 4 postgraduates & Modelling Expert[19] & Traceability Expert[20] |
| **Pre-condition** | All updates from deskcheck activity completed. |
| **Input** | M2 Profile models for inspection. |
| **The Results** | This proved to be a very useful activity. The main discussions were around the size and level of detail of the models. While the students believed that the level of detail was correct, the two industrial participants agreed that the *Profiles* should suppress the attributes and operations, where possible and that less model elements should be used as in its current state it was difficult to understand. In Figure 11-13 below, *Complex Original Profile*, we illustrate one of the original models that we presented. The reality was that the models contained too many modelling elements, for example, modelling elements for requirements, test cases, changes requests and design which we later integrated to become one model element, traceability item. |



[19] Thanks to Alida Del Porte
[20] Thanks to Mike Swift, Software Futures, Best Practices Team.

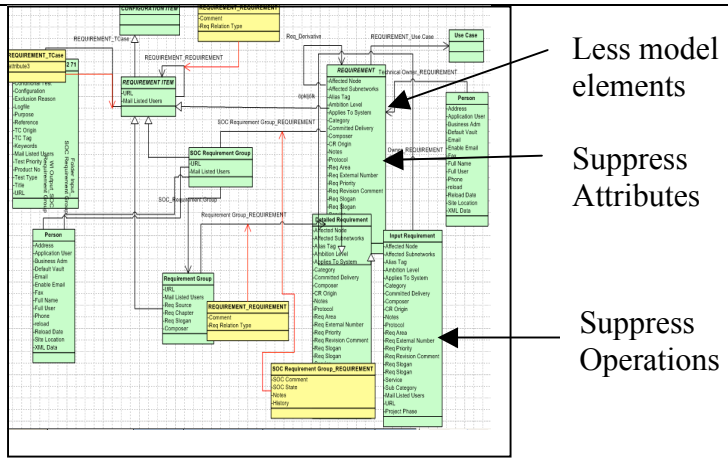



**Figure 11-13: Complex Original Profiles**

A number of completeness faults, for example missing connections between model elements were detected.

By having a modelling expert present in the testing session, intermediate problems with algorithms were identified. A lot of discussion in relation to TRAM was based around the UML Foundation Package which provided the basic infrastructure for exploring the static structure of traceability and TRAM represents a lot of the static data. The discussions on the TRAP were based around the UML's Behavioural Elements package which provides the linguistic elements for modelling the dynamic behaviour of the system or the process in this case. However, there are cases where the behavioural information had to be augmented with static information typically found in the Foundation Package.

| | Correctness | Conformance | Expressiveness |
|---|---|---|---|
| **TRAM Profile** | **Medium** | **Medium** | **High** |
| **TRAP Profile** | **Medium** | **Medium** | **High** |

**Table 11-8: Evaluation of M2 Profiles**

Table 11-8, *Evaluation of M2 Profiles*, summarizes the evaluation of the TRAM and TRAP profile against the specific criteria outlined above. Each criterion is defined by its scale (with the three levels, high, medium or low). On analysis of Table 11-8, it is clearly visible that the TRAM and TRAP profiles rate well in terms of expressiveness (since it is UML-based). However, the UML profile specifications were only rated with medium rankings for correctness and conformance against the OMG standards that they were based upon. However, these lower ratings were due to

| | |
|---|---|
| | the fact that we had to decide between correctness and conformance of the models versus ease of use and understanding. |
| **Comment of Researcher** | The main challenge in this inspection was the difference in opinion between the students and the practitioners. The most difficult task was keeping the test session focused on testing matters. Too frequently the discussions focused on the semantics of UML rather than on the models themselves. The researcher had to keep the focus of the inspections on the models and regularly illustrate the requirements we were testing against. |

**Table 11-9: Model Checking**

### 11.5.3 Walkthrough of Profiles

Before presenting the results of the walkthroughs, we again describe the evaluation criteria that we evaluated the models against:

- *Completeness*: A validation assessment must determine if the model represents all of the properties that the *user requires* to pursue traceability the technique, practice or even the philosophy. It should also identify the required properties that the model does not represent. In this case, the user should decide whether the information about the models capabilities is important to their acceptance decisions. We created Table 11-10, *Completeness Validation Ratings*, below to assist the reviewers with grading the models[21]. Each attendee was given a copy of the table below and the results were averaged at the end of the workshop.

| Tier of Validation | Supporting Information | Informal Validity Statement |
|---|---|---|
| 0 | Nothing | I have no idea |
| 1 | Simple statement of validity | It works. |
| 2 | Required model elements compared against the traceability contexts it represents | For what it represents, it is complete enough |
| 3 | Required model elements compares against the traceability context and the standards it is based upon | It compares to the context and the standards as defined in the conformance statement |
| 4 | Required model elements, the standards it is based upon and the assessment criteria imply that I would use this model in my organisations in it entirety. | I would use this model in my organisation. |

**Table 11-10: Completeness Validation Ratings**

▪Confidence: A validation assessment must explicitly characterize the *confidence that the user* can place in its information, particularly the error estimates. It should assign confidence rating to each error estimate in such a way as to represent all of the sources of uncertainty associated with the validation measurements. In Table 11-11 below, *Validation Rating*, we define a confidence rating table. It should also identify areas where either the model or the validation assessment cannot provide sufficient confidence to meet the user's requirements and suggest the means through which to increase that confidence. We intend that the assessed confidences reflect the totality of uncertainties associated with making a validation assessment.

| Tier of Validation | Supporting Information | Informal Validity Statement |
|---|---|---|
| 0 | Nothing | I have no idea |
| 1 | Simple statement of validity | It works. |
| 2 | Required model elements compared against the traceability contexts it represents | For what it represents, it is complete enough |
| 3 | Required model elements compares against the traceability context and the standards it is based upon | It compares to the context and the standards as defined in the conformance statement |
| 4 | Required model elements, the standards it is based upon and the assessment criteria giving complete confidence | I'm confident that this model is valid |

**Table 11-11: Validation Ratings**

The other evaluation criteria were reusability, understandability and expressiveness. The other main questions that we asked the assessors were:

- Does the model represent and correctly reproduce the behaviours of the real world situation when implementing traceability?

- Does the model meet its intended requirements in terms of the methods employed and the results obtained?

- Would you use the models in one of your organisation?

| Test Type | Deskcheck |
|---|---|
| **Test Type** | Profile Walkthrough |
| **Test Objective** | A walkthrough is an informal way of presenting a technical document in a meeting |

[21] A similar table was proposed by CARIMO, R. A. (2006) Evaluation of UML Profile for Quality of Service from the User Perspective. *School of Engineering.* SE – 372 25 Ronneby, Blekinge Institute of Technology, Sweden.

| | |
|---|---|
| **Location** | Software Futures (Cape Town) |
| **Background** | The researcher walked through the models because some of the participants believed they did not have the technical expertise to review the models without help from the researcher. |
| **Number of Reviewers** | Researcher, 2 undergraduates, 1 postgraduate, 7 practitioners (6 best practices team, one business development manager) |
| **Pre-condition** | The models had already been updated with the comments from the lab trials and the model checks. |
| **Input** | The updated TRAM and TRAP M2 Profiles |
| **Output** | Approved models. |
| **The Results** | As shown in Table 11-12 below, *Results from Walkthrough Model Check*, the main cause for concern was the lower than expected ratings for the TRAP both in confidence and completeness. This can be explained by the fact that few of the reviewers had any experience with process modelling and preferred standard process definition techniques like RUP. |

| | Completeness | Confidence | Reusability | Understandability | Expressiveness |
|---|---|---|---|---|---|
| **TRAM** | 3 | 3 | High | Medium | High |
| **TRAP** | 2 | 2 | High | Medium | High |

**Table 11-12: Results from Walkthrough Model Check**

In Table 11-13 below, *Test Discussion on TRAM Model Elements*, we summarise the main outcome on the TRAM elements.

| Test Object | Test Discussion |
|---|---|
| **TRAM** | |
| Traceability Suite | Level M2 should be platform or tool independent; however this element increases the level of understandability of the models to the end users and should be kept in the model. |
| Suite Configuration | The same discussion occurred as per the Traceability Suite,, however users agreed that this model element gave the users the concepts that a traceability suite has different functions. |
| Traceability Item | This is the most important element and the relationships to the other model elements are important. |
| Item Type | This model element was difficult for almost all people who participated in the validation phases. We had to describe functional versus non-functional requirements for people to understand what an Item Type was. We noted that in future |

| | |
|---|---|
| | modelling efforts, the attributes of functional and non-functional should be included to assist with the understandability of the Item Type concept. |
| Relationship | This model lead to a number of good discussions. Most people understood the concept of traceability relationships; however, there was some disagreement on the different types of traceability relationships. |
| Work Package | This model element was only added in during the case study with Ericsson. |
| Configuration Item | Some of those participating in the validation at this stage believed that configuration item and traceability item were the same model element. However, after describing that traceability items became configuration items after a baseline then everyone agreed that this was in fact a separate model element. |
| Behaviour | Many of the discussions that took place around this model element were about the relationship it had with the Behaviour Element in UML 2.0. |
| Configuration | A number of people at the participation workshops believed that this model element should be removed and the capabilities integrated with the Configuration Item. However, after discussion there was general consensus that this model element should remain in the model. |
| Baseline | General discussion on the name of this model element took place. One person believed that this should be moved into the TRAP model. However, it added to the semantic understanding of traceability and it was generally agreed that this element should remain. |

**Table 11-13: Test Discussion on TRAM Model Elements**



In Table 11-14 below, *Test Discussions on TRAP Model Elements*, we illustrate the main discussions that took place with regard to the TRAP components. In many cases the majority of the model elements were self explanatory while the main questions were related to the relationships between the elements rather than questions over the existence of the elements.

| Test Object | Test Discussion |
|---|---|
| **TRAP** | |
| Goal & Precondition | Every traceability activity must have a precondition, for example that the user has the responsibility to carry out the activity, and a goal, for example to set up a traceability link. The discussion concluded that many processes do not specifically define these to important variable and therefore that these were useful model elements. A lively discussion took place on what the Boolean |

| | |
|---|---|
| | values should be. It was concluded that this was very useful for defining the state of the traceability item after the traceability link had been executed. |
| Process | As we discovered using ISO 15504 and the RUP, processes are subdivided into disciplines or sub-processes. It was agreed that this was an essential model element. |
| Phase | All processes have phases, for example pre-study or feasibility. There was general consensus on this process element. |
| Lifecycle | A number of phases together become a lifecycle. The main discussion on lifecycle was the different types of lifecycles. For example, there are product development lifecycles and project development lifecycles. The product lifecycle incorporating more phases than the project lifecycle. The relationship between different lifecycles was discussed in detail. |
| Iteration | This model element was self explanatory and lead to very little discussion. |

**Figure 11-14: Test Discussions on TRAP Model Elements**



We concluded the participatory review with the question: Would you use the models in your organisation? One of the first responses was that the models provided an excellent framework for discussions related to process, traceability and traceability data. The model elements in both TRAM and TRAP, in general, captured all aspects of traceability while the relationships between the model elements were where the discussions took place. Therefore for providing a framework for discussion the models proved useful. Furthermore, the models provided a mechanism for reaching agreement between the different members that participated in the reviews. Therefore as a mechanism for gaining agreement and promoting communication this approach was very useful.

However, a number of the industrial practitioners concurred that in their experience the models were still too complex to be used by an entire organisation. They believed that while many engineers had modelling experience and would understand the models, however, without training on the models many of the users would lack the know-how to interpret the models if they did not have a guided walkthrough from an expert. It was therefore decided that a process team could use the TRAP models, for driving cognitive process walkthroughs, and for starting discussions and gaining agreement on process related matters while the requirement manager could use the TRAM for gaining agreement on matters related to the traceability data. They all agreed however, that the lower level models possessed better understandability qualities than the meta-level models. The fact that agreement was reached in this simple form validates our model approach.

At this stage we also did some evaluations on the patterns, which we discuss in later sections. For now it is suffice to say that we reviewed some of the patterns that had been developed by some of the participants.

## 11.6 STEP 4 - FIELD TRIALS (ERICSSON)

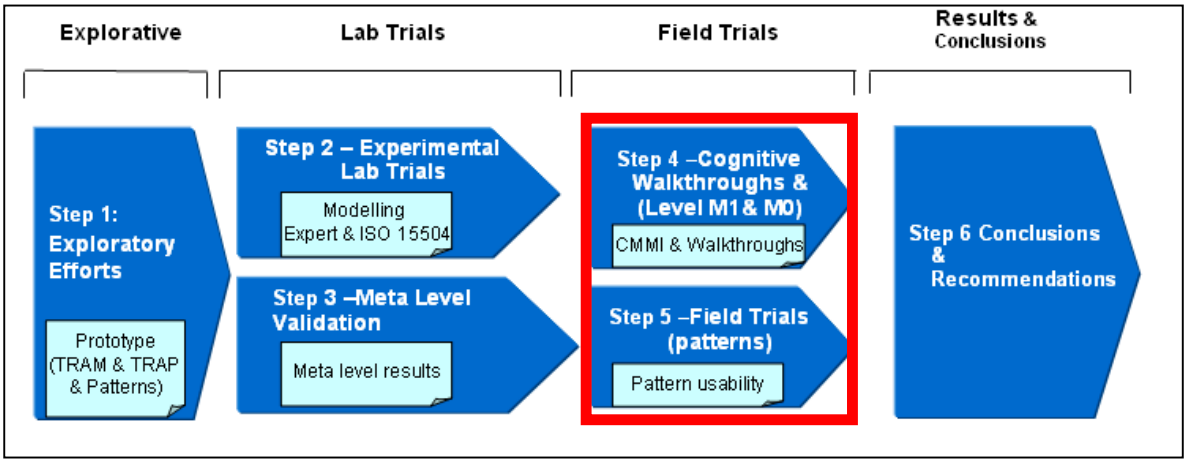



**Figure 11-15: Assessment Method**

In Figure 11-15 above, *Assessment Method*, we once again, illustrate where in the assessment process we describe in this section. In 2004, we began initial investigations at Ericsson, carrying out interviews and reviewing historical data from the different OSS-RC projects. During 2004, we created the TRAP and TRAM meta-level (M2) models. In early 2005, we started to address the creation of the lower level models (M1 and M0). On completion of the M1 and M0 models the next natural stage was to evaluate the components, except in this case we use CMMI due to the fact that Ericsson was using this assessment framework.

### 11.6.1 Assessing TRAP (M1 and M0)

As previously stated when it came to assessing the TRAP and TRAM in the context of Ericsson we came across a major obstacle, that Ericsson used CMMI to assess all their process models. In fact they had already completed a CMMI assessment in 2005 with another one due in 2008. The corporate decision to use CMMI in the evaluation of the OSS-RC processes would lead to inconsistencies in our evaluation approach as they insisted on using CMMI in our assessments. As already discussed, we had already assessed TRAP using ISO 15504. It was however clear, that if the models, especially the instantiation models or application models were to be accepted, then we would have to use the local CMMI assessors. The Capability Maturity Model was developed by the Software Engineering Institute (SEI)[22] and similarly to ISO 15504, it is used as a model for judging the maturity of the software processes, for identifying the key practices that are required to increase the maturity of these processes and to guide process improvement across a project, organisation [23] or in our case our research project. One of the primary goals of using the

[22] The Department of Defence established the SEI in response to a perceived crisis in software development related to escalating software cost and quality problems, at Carnegie Mellon University in Pittsburgh, Pennsylvania in the early 1980s. SEI began the development of a process improvement model for software engineering in 1988. In August 1991 the first version of the Capability Maturity Model for Software (SW-CMM) was published by the SEI (SEI (2007a) Concept of Operations for the CMMI. Carnegie Mellon University, Pittsburgh, Software Engineering Institute.

[23] The SEI created the first CMM designed for software organizations and published it in a book, The Capability Maturity Model: Guidelines for Improving the Software Process.

CMMI model is to have processes that are repeatable, defined, managed, and optimized, which aligned with our objectives for the traceability framework that we propose. It integrates separate organizational processes for example business processes with software development processes. (Paulk et al., 1995)

A maturity level is a defined evolutionary platform for organizational process improvement. A maturity level consists of related specific and generic practices for a predefined set of process areas that improve the organization's overall performance. The maturity level of an organization provides a way to predict an organization's performance in a given discipline or set of disciplines. Each maturity level determines an important subset of the organization's processes, preparing it to move to the next maturity level. There are five maturity levels, each a layer in the foundation for ongoing process improvement, designated by the numbers 1 through 5; Initial, Managed, Defined, Quantitatively Managed, Optimizing.

With this background we set about assessing the TRAP and TRAM components of our solution framework. The assessment was carried out with the OSS-RC R6 Requirement Manager and the OSS-RC R6 Process Modeller. The rules for compliance of the different practices were rated as:

- Fully Implemented (FI): No weaknesses noted, the practice is full compliant against the CMMI assessment criteria.
- Largely Implemented (LI): One or more weaknesses noted.
- Partially Compliant (PI): More than one weakness and judged that the practice needs improvement or is inadequate.
- Not Implemented (NI): The practice is not implemented.

The assessment took 3 hours with the researcher walking through the models and the processes and the assessors assessing the models under the Specific Practices. This assessment provided us with an evaluation of the capability of TRAM and TRAP. We used the specific goals defined by CMMI to assess the capabilities of TRAP and TRAM. After each goal, we illustrate the assessment criteria as defined by CMMI and then we describe whether TRAM or TRAP is compliant to the assessment.

| Goals/ Practices | CMMI Assessment Criteria | Compliance & Benefits to Ericsson |
|---|---|---|
| ***SG 1 Manage Requirements*** | | |
| Specific Practice 1.1-1 *Obtain an Understanding of Requirements* | 1. Reach an understanding of the requirements with the requirements provider so the project participants can commit to them. | ***Compliance*****:** Largely Implemented<br>***Benefit of TRAM & TRAP to Ericsson:*** At Level M2, the models are best used communicating and gaining agreement on matters related to the process used or how to manipulate the traceability data. The assesors identified that the model elements Traceability Items, Types and Relationship were the most useful for gaining agreement with the stakeholders. They concluded that the M2 models should be used for discussions at at organisation level and with the stakeholders rather than the project instantiations models. (M0 & M1). They also pointed out that the patterns provides an even better method for gaining an agrement with the stakeholders. |
| SP 1.2-2 *Obtain Commitment to Requirements* | 1. Assess the impact of requirements on existing commitments.<br>2. Negotiate and record commitments | **Compliance: Largely Implemented.**<br>***Benefit of TRAM & TRAP to Ericsson:*** The Level M1 and M0 models offered the most benefit to Ericsson. The reason was that these process models described how to carry out impact analysis on existing requirements. |
| SP 1.3-1 *Manage Requirements Changes* | 1. Capture all requirements and requirements changes that are given to or generated by the project<br>2. Maintain the requirements change history with the rationale for the changes. | **Compliance: Fully Implemented.**<br>***Benefit of TRAM & TRAP to Ericsson:*** It was this aspect of TRAM and TRAP that provided the most benefit to Ericsson. The Requirement Manager believed that the process models could be used in all discussions related to change management and the impacts on traceability. |

| | | |
|---|---|---|
| SP 1.4-2<br><br>Maintain Bidirectional Traceability of Requirements | 1. Maintain requirements traceability to ensure that the source of lower level (derived) requirements is documented.<br>2. Maintain requirements traceability from a requirement to its derived requirements as well as to its allocation of functions, objects, people, processes, and work products<br>3. Maintain horizontal traceability from function to function and across interfaces | **Compliance: Fully Implemented**<br><br>***Benefit to Ericsson:*** In this case TRAM was the most useful set of models to achieve these practices. The models clearly illustrated how to trace between different levels of traceability. For example, in OSS-RC the models were used to show the relationship between the Input Requirements and the Detailed Requirements. |



| **GG 2 Institutionalize a Managed Process** | | |
|---|---|---|
| GP 2.1 Establish an Organizational Policy | Establish and maintain an organizational policy for planning and performing the requirements management process. | **Compliance: Fully Implemented**<br><br>**Benefit to Ericsson:** The assessors agreed that the solution framework added a very valid benefit at an organizational level for managing the process. They should be used for gaining agreement between the different parties involved, providing a good framework for discussions and it was easily maintainable. They identified that the only problem with the solution was that it required a person with the right modelling, process and traceability background to drive the effort and in the hands of the wrong personnel that the process would fail. Furthermore, they suggested that the framework should be supported with clear, concise presentation material before it could be used in Ericsson. |

| | | |
|---|---|---|
| GP 2.3 Provide Resources | Provide adequate resources for performing the requirements management process, developing the work products, and providing the services of the process. | **Compliance: Partially Implemented.**<br><br>**Benefit to Ericsson:** The whole framework as it currently stands did not fully address resource allocation. They agreed that resource numbers and names could be added to the different process models but they believed that this area could be improved especially around estimating resource allocation. In future work resource allocation should be addressed in the models. |
| GP 2.5 Train People | Train the people performing or supporting the requirements management process as needed. | ***Compliance: Not Implemented.***<br>***Benefit to Ericsson:*** While the models and the process were not used for any training sessions, they were used in training type settings, with presentation foils and supporting background information as per normal course structures. The assessors agreed that this was one of the primary advantages of the entire framework. They noted that the layered approach would provide logical course modules and that they provided an effective method for exercises and group discussions on matters related to traceability. |
| GP 2.6 Manage Configurations | Place designated work products of the requirements management process under appropriate levels of configuration management | ***Compliance: Fully Implemented.***<br><br>**Benefit to Ericsson:** Both assessors believed that the configuration aspect of the models addressed configuration management sufficiently, especially its relationship with traceability items. |
| GP 2.8 Monitor and Control the Process | Monitor and control the requirements management process against the plan for performing the process and take appropriate corrective action. | **Compliance: Fully Implemented**<br><br>**Benefit to Ericsson:** The models created and the underlying processes were easy to monitor and control. They suggested an approach for attaching more notes to the models that contain the corrective actions but this is a common modelling complaint. |

| GG 3 Institutionalize a Defined Process | | |
|---|---|---|
| GP 3.1 Establish a Defined Process | Establish and maintain the description of a defined requirements management process | **Compliance: Fully Implemented**<br><br>**Benefit to Ericsson:** This was one of the strengths of the framework for Ericsson. The framework was a description of the entire processes, from an organisational to project level. |

**Table 11-14: CMMI Assessment Results**

In conclusion, while we were dissatisfied that we had to use two assessment frameworks, it did however, have its own advantages. Firstly, both frameworks describe traceability and this gave us a better overall understanding of traceability. This furthered our knowledge of traceability from two separate perspectives on traceability, one from an international community supported by the United Nations, the other a North American perspective, based on principles defined originally by the US Department of Defence.[24] Secondly, using the two frameworks at two different stages not only give us good indicators for areas of improvements it ensured that we were doubly sure that in fact the models and the processes were assessed completely. In most instances different assessors have the skills required to use the two different frameworks, which was the situation in this study, therefore we received two different assessor's perspectives, which helped validate the overall usefulness of the Traceability Framework.

The question of which standard should be used for traceability? Once again the answer would be "it depends". CMMI is more widely used, supported with more literature and case studies; therefore it is the obvious choice. CMMI is easier to come to understand, and the SEI do an excellent job of providing case studies and conference papers that support different approaches. With regard to the definition of traceability, we believe that the ISO 15504, recommends more traceability practices across the entire development lifecycle. ISO 15504, has an academic feel to it, while CMMI has a more practical dimension to it. As suggested earlier on in Section 11.4.5, the integration of these two assessment frameworks on all matters related to traceability would provide a very powerful standard for the research communities to assess different aspects of traceability.



### 11.6.2 Key Benefits to Ericsson

While we mentioned the benefits to Ericsson in Table 11-14 above, the four main benefits to Ericsson as described by the OSS-RC Requirement Manager were as follows:

- *Gaining Agreement:* Despite the wealth of knowledge in Ericsson, gaining agreement on matters relating to traceability is difficult. A large number of participants in the different

[24] The fact that CMMI had its origins in the US DoD, caused some ethical issues for the researcher to overcome. However, this does not take from the assessment framework, the researcher would however prefer to use standards that are created by international organisations rather than ones that have their origins and funding in military projects.

workshops agreed that both the TRAM and TRAP were very effective approaches to gaining agreement on aspects of traceability.

- *Promotes Better Communication:* Throughout the design and testing of the components many of those participating in the process commented that this approach promotes a framework for discussion, promoting better communication.

- *Supports Reuse:* The models and processes can be updated and reused in future projects. The OSS-RC Requirement Manager described that the models provided an efficient way of describing the different aspects of the Requirement Management Plan. For example, the traceability items, the relationships and the link types.

- *Training:* The different levels of abstractions and the layers provide a good framework for delivering courses related to requirement engineering and traceability. One course developer stated that the models would also provide good visualisations during the course exercises. For example at the end of the course, giving the model elements to the attendees and getting them to design a model equivalent to the TRAM or the TRAP.

## 11.7 LAB & FIELD TRIALS (PATTERNS)

While evaluating patterns is not an easy task, especially when the patterns that you are proposing are novel with no reference literature on how to evaluate these new patterns. Before discussing the evaluation technique that we applied let us first reflect on Alexander's criteria for evaluating patterns. In his criteria, *empowering users*, Alexander intended pattern languages to be a means of sharing design knowledge with users of buildings, as part of *a participatory process*. In this study the TRAM and TRAP evolved from the inputs of real users, mainly from the OSS-RC telecom domain but also from modelling experts. One of the initial discoveries of the pattern was during one such modelling review meeting in South Africa. Within Ericsson patterns emerged during discussions and white board sessions. While the researcher in many cases documented the patterns there were a number of situations when the requirement manager or configuration manager assisted with the creation of the pattern. Furthermore, when the template was distributed to a number of participants in the survey, they created patterns from their experiences with traceability. For example, Software Futures who specialised in the delivery of requirement engineering solutions, created a number of patterns. In some cases the patterns were simplistic with the biggest problem identified by the users being the difficulty they encountered creating UML models.

Secondly, Alexander's intention was that a pattern language should allow users to *generate complete designs*. He explicitly relates pattern languages to Chomsky's notion of generative grammar. In this study some of the patterns that we created, for example, Define Traceability Item, came before the actual modelling sessions and hence did assist with the generation of the design. Furthermore, the OSS-RC Requirement Management Plan which describes the complete implementation of traceability in OSS-RC was replaced by the researcher and the Requirement Manager using traceability patterns. Therefore the traceability patterns could generate the complete traceability implementation.

Finally, Alexander talks of his aim to achieve 'the Quality without a Name'. (Alexander et al., 1977, Alexander, 1979)His stated aim is that people experiencing buildings developed using pattern languages should recognise this quality. Their lives should be enhanced by the experience. While this is a little abstract and difficult to measure, we believe that by using patterns the quality of the practice of traceability should improve.

Due to the time constraints and the sheer magnitude of this research project we did not get an opportunity to carry out long controlled experiments to investigate if the patterns did enhance the quality of the traceability practices but our results did drastically improve the success of this research project by assisting us with a formalised approach for data gathering, communication of all matters related to our empirical studies, the presentation of our project and for simplifying complex traceability concepts. On a number of occasions we did use the patterns for presentations to different team members and the informal feedback was very positive. The aspects of the patterns that we found most difficult to evaluate were based around the reuse measurements. However, researchers before have described that by using patterns you do in fact promote reuse which we hope applies to the traceability patterns. To overcome this problem we developed the TRAPT tool, which we demonstrated at a number of participatory workshops. However, we did not test this in a controlled industrial context and further evaluations are recommended.

### 11.7.1 Lab & Field Trials Patterns

Two lab sessions were carried out in the Fall of 2005, where a presentation on Traceability Patterns were given lasting approximately 1 hour for each. Thereafter, each student completed an exercise to create a traceability pattern. The primary objective of this exercise was to evaluate how the patterns improved their understandability and communicativeness of matters related to the patterns.

Students performed the following tasks, in order, for the listed components:

- *Create*

Given a scenario, for example, two requirements, with attributes and relationships to each other, each student was asked to create a pattern using the pre-defined templates.

- *Communicate*

Each student was asked to present the pattern to the rest of the group.

Students then were asked to fill out a questionnaire to evaluate the experience of using patterns. 10 students completed the survey.

Table 11-15, *Lab Trials with Traceability Patterns*, provides a summary of the responses for the background information section of the questionnaire. [25]

| Question | Response |
|---|---|
| Department: Computer Science | |
| Program level and year | Undergrad – 2nd year : 2 students<br>Honours- 4 students<br>Masters – 1st year: 2 students<br>Masters – 2nd year: 2 students<br>PhD – 1st year : 1 student |

[25] Some discrepancies occurred in the early documentation of the results in this experiment. For clarification, the number of students involved in this experiment is 10 not 12.

| | |
|---|---|
| | Overseas Student Masters: 1 student |
| Do you understand the concepts of traceability? | Yes: 1 No: 9 |
| Have you ever described traceability practices to anyone? | Yes: 1 No: 9 |
| Have you an understanding of patterns in any form? | Yes: 8 No:2 |
| Did you successfully create a traceability pattern? | Yes: 9 No: 1 |
| Did you communicate the pattern to the other attendants? | Yes: 8 No: 2 |
| In your opinion did the pattern approach help you to problem solve on the traceability exercises? | Yes: 9 No: 1 |
| Do you think patterns promote easier communication on matters related to traceability? | Yes: 10 No: 0 |

**Table 11-15: Lab Trials with Traceability Patterns**



In 2005, the concepts of traceability patterns were in its infancy. We did not test, the patterns from the perspective of the model TRAM and TRAP patterns. Considering the small number of participants the preliminary results were encouraging. Lab testing on a concept like traceability where so few students have any practical experience with traceability is difficult. However, it does reflect a worst case scenario.

In a similar experiment at Ericsson, we carried out the same test with six software engineers from design and test disciplines. Using the same technique as that described above we achieved similar results as those from the lab. However, at this stage 2007, we decided that further tests were required. The tests and results are shown below:

| Evaluation Criteria | Description of Experiment | Results |
|---|---|---|
| Learnability? | How quickly and easily did the user learn to use the pattern template and create a new pattern. | After 30 minutes of instruction from the researcher the resources began creating patterns. These patterns were of mixed quality, and the biggest problems encountered was that the engineers did not know how to create the sketches. This was a common complaint when creating the traceability patterns. |
| Communicativeness | We evaluated how the patterns helped communication on matters related to traceability. | We asked the Requirement Engineer to describe an aspect of the OSS-RC Requirement Engineering Plan in conversational form. Then we gave him a pattern to communicate another concept. All participants agreed that the |

| | | |
|---|---|---|
| | | patterns promoted better communication. |
| Understandability | To test if the patterns improved the understanding of those involved in the experiment on matters related to traceability. | We began the experiment by describing a complex aspect of traceability, by using plain English. We asked the participants to grade their understanding of the concept. Then we used a traceability pattern to describe the exact same concept, once again asking them to evaluate their understanding. The results were impressive. Every participant rated their understanding of the issue as greatly improved.<br><br>All six people involved in the experiment agreed that the pattern approach helped them understand the concepts quicker. They found the consequences the most difficult part to understand. |
| Structuredness | A simple definition of traceability was that patterns define structures between traceability items. | We walked through the TRAM and the TRAP and asked the participants to grade their understanding of the structure. We then divided the different models into a number of different patterns and asked then to grade their understanding of the structure. They all agreed that patterns simplified or componentised complex structures related to the TRAM and TRAP. |

**Table 11-16 Pattern Validation**



While both these tests are clearly not sufficient, they do however, give us preliminary results that patterns are an effective way of describing traceability. The reason for lack of further testing was due to the time constraints that we faced with this project. Over the duration of this project we did carry out other experiments with users, for example during lab trial sessions, these experiments were carried out too early into the evolution of the patterns and therefore not suitable for discussion.

However, the patterns that we present in Chapter 9, all emerged from human experiences and in essence this is the main benefit of patterns. Capturing experiences and formulating that experience in a formalised approach. Moreover, there is an abundance of literature on patterns and their characteristics and we can only assume that considering we followed a similar approach that should we carry out further, larger scale tests that they would produce results similar to the results from other pattern efforts.

We strongly suggest that future research should be carried out on testing these patterns, we are confident that we have achieved our primary objective of providing the

research community with a novel approach for describing aspects of traceability. It is our hope that future researchers will take this concept and develop it further for future generations of traceability research.

## 11.8 CONCLUSIONS ON VALIDATION

*"On the basis of data gathered from industrial sources and previous research efforts, traceability practices are in need of a structured, systematic and disciplined rule-based modelling approach to overcome the problems being encountered in the field"*

*And that "the package of the TRAM and TRAP models, plus the patterns provide a flexible basic package, easily adaptable to a wide range of users, with potential to overcome many of the problems in the field."*

Building a Traceability Framework is not a simple process. This chapter has attempted to test all the components that encompass the Traceability Framework.

The results from the lab trials demonstrate that:

1. The TRAM and TRAP components can be created as self-contained configurable entities.

2. The TRAP and TRAP conform to the underlying standards.

3. It is possible to model traceability concepts and traceability process elements using a layered approach, with each layer providing benefit to different users.

4. The ISO 15504 assessment framework is a viable framework for assessing aspects of traceability.

A field trial was conducted and performance measurements were taken. These supported the assertions that:

1. The Traceability Framework is relatively simple and understandable.

2. The TRAM and TRAP components are useable, manageable, understandable, extensible and reusable.

3. That traceability patterns add benefit to practitioners, by promoting better communication, quicker learnability and understandability and that they can be utilised to describe the emerging structures in traceability models.

4. That CMMI is a viable assessment framework for assessing certain aspects of the Traceability Framework.

These experiments have shown that there is promise for the Traceability Framework and similar approaches to replace the traditional non-model based, non-pattern based approaches. The results from this work vindicate the model based approach and improved the efficacy and efficiency of traceability. Also, by investigating particular validation techniques, this work has demonstrated possibilities for future research into traceability testing.

It is hoped that the results of this work will change the way people describe traceability. The evaluations and feedback received from users and colleagues has strengthened the case for building traceability models and utilising traceability patterns.

Building upon a foundation of extensibility, it then will be possible for traceability researchers and practitioners to work on providing more applications to users, thus bridging the wide gap between current research and real world scenarios, and ultimately improving traceability practices in the future.

# Chapter 12 SUMMARY & CONCLUSIONS & FUTURE WORK

## 12.1 INTRODUCTION

This chapter provides a brief overview of the research, the results and findings of the work, the conclusions that can be drawn, and directions for future work. It is the concluding chapter of the thesis, but it is more than a brief statement of conclusions. Its purpose is to provide an executive summary of the empirical study on the state of the art of traceability in small, medium and large organisations and report the findings from the design, test and validation of the Traceability Framework.

Each section endeavors to follow a pattern: a brief summary of the work in one or more chapters; a summary of findings; a brief discussion of important insights; a summary of the outcomes carried forward to later work, or implications for further work leading to conclusions.



## 12.2 FINDINGS FROM STATE OF THE ART REVIEW

### 12.2.1 Background & Summary of Work

As stated at the start of this thesis, the author came into this research project with over six years experience across thirty-five countries in building complex telecommunication systems with Ericsson, which is widely accepted as one of the worlds top telecom systems development companies. Throughout the early to mid nineties Ericsson utilised its own procedural languages, tools and waterfall processes. Its document centric approach was well supported by tried and tested processes. Each document had strict document numbers and traceability was implicit between the chain of artefacts produced in the development environment. However, due to changes in international network standards (GSM, W-CDMA and UMTS) and changes in the market needs, major new system developments were underway. In 1997, Ericsson's joined with Rational to introduce new best practices in Requirement Engineering in the new projects, and new use-case driven processes were introduced and new requirement tools were brought into use.

It was into this environment, in 1998, that the author got his first taste for the complexities of requirement engineering. He gained first hand experience of the many challenges that projects and individual engineers faced in ensuring the development of quality products. Numerous difficulties arose namely; multiple sites, diverse development environments, requirement duplication, unproven processes, poor change management and incomplete tools. All these factors had to be overcome to deliver projects in short lead times with tough competition in a volatile market. The author's fascination with traceability grew from the chaos all around him. He constantly questioned the problems that were arising, looking for better, simpler ways to provide a solution. It was with this background that the author decided to undertake a research project to investigate better ways of carrying out traceability, thereby attempting to provide a solution to practitioners and academics alike.

### 12.2.2 Findings

During the literature review in 2003, the author was struck by a number of key problems in the recent publications. Firstly, there was a noticeable lack of empirical data in the past decade on the state of the art of traceability in small, medium and large organisations. During the nineties, researchers like Gotel and Finkelstein and Ramesh did provide survey information describing the problems faced by organisations. Of particular interest to this study was Lindvall and Sandhall's case study in the early nineties of traceability on object oriented projects; a new paradigm at that time. Furthermore, in 1995, the Chaos Report highlighted the importance of requirement engineering and traceability which showed that poor requirement engineering was a root cause for many software development problems. Yet, in 2003, there was still not a single report telling the story of traceability practice and problems over a number of years. We believe that this fact alone merited a research project in its own right.

Secondly, much of the research addressed micro level traceability matters, like traceability between model elements, which provided solid contributions, however few solutions addressed traceability at a macro level, for example traceability across the entire product development family. Many papers, proposed new techniques but in some cases the solution lacked empirical validation. One could argue that perhaps this was also due to the fact that few researchers had access to industrial data that spanned the entire development lifecycle. While new communities were emerging and new workshops were taking place, in particular the ACM's Traceability in Emerging Forms of Software Engineering (TEFSE), there was still no single voice or authority that collaboratively brought together, captured the real-world problems or specified the direction of future research. Therefore at this stage we already knew that we could make a major contribution to the traceability communities, simply by telling the story from the field.

The basic principles that we deduced from the literature are that traceability is a practice that promotes the development of better quality products that meet the needs of the paying customer. Traceability is the discipline of getting an entire organisation to work together to build a quality product. It encompasses all practices from sales and marketing right through the product development lifecycle to maintenance and support. Traceability is about relationships between all artefacts that impact the system in any way. Traceability is also about controlling change, it supports impact analysis, it mandates communication between all personnel who can impact the quality of the system. Generally, organisations that claim good end to end traceability practices have high levels of process maturity and quality standards, but even then implementing traceability is no simple task because it involves many far reaching problems.

On completion of the literature review, we were sure that a major contribution to the research community was to undertake a case study that that would build on the authors experiences and tell a story about traceability in large corporations developing complex enterprise systems and the many challenges that were faced and the steps that were taken to address these problems.

A number of pivotal papers provided valuable direction to the evolution of a traceability solution. Patricio Letelier, at TEFSE 2002, described how to build a traceability framework using UML and a metamodel approach. This contribution directed us towards the OMG and the emerging Model Driven Architecture that they proposed. From our experience we knew that UML was a modeling language that was widely supported by Ericsson and more importantly understood by many in the traceability community. Zisman

and Spoudakis described the problems faced when implementing traceability in complex product families. Therefore, we knew that the research community were aware of the problems that the author would face in Ericsson who had a large portfolio of products. Steve Riddle and Paul Arkley introduced a position paper in 2003, describing the empirical work that they intended carrying out over the coming years. They also highlighted the lack of empirical data on traceability as a critical problem and produced some revealing papers in the following years on traceability at prestigious conferences. A solution was forming and initial experimental work began.

In the years that followed there were a number of key publications, which in many ways did not influence our research direction rather they reconfirmed that we were in fact going in the right direction. In 2005, Martin Gills presented a Survey of Traceability Models in IT projects. Similarly to our efforts, he utilised questionnaires and interviews to survey 6 IT companies: to learn the attitudes of project members towards the traceability; to capture the most typical traceability relationships between traceability items; to evaluate the amplitude of differences between traceability practices and to compare his results with similar research efforts.

A pivotal moment in the story of traceability in the 21st Century, as a result of the TEFSE workshops, was the formation of the Centre of Excellence for Traceability in 2006. Its primary objective is to create a community of researchers and experts in the field of traceability with a goal of improving traceability practices and techniques. An output from this initiative and workshop was the release of the Centre's Problem Statement and *Grand Challenge Report*. However, because this report was only released in 2006, it did not influence our work except once again to confirm, that the problems that we identified and the solution we developed did in fact reflect the challenges and needs of the traceability community.

In 2005, another dedicated traceability workshop appeared in Europe at the OMG's European Conference on Model Driven Architectures (ECMDA), however, in this instance they had a different focus; traceability in model driven architectures. This forum discussed many different aspects of traceability, in particular, change management of models and impact analysis of model modifications. The papers at this workshop provided us with some valuable contribution on different aspects of models and traceability. Furthermore we presented a paper at this workshop and gained some valuable insights from the peer review.

While striving to be impartial in our collection and analysis of results, our initial hypothesis after the literature review was that the lack of observable scientific discipline and engineering empirical data which did not reflect or address all the problems that were being faced in the field. Therefore we set about by rectifying this situation by using the experiences and contacts gained in Ericsson to document the state of the art, the challenges faced and provide an ethnographic perspective on what changes were introduced and the impact these changes had on the success or failure of traceability in the Ericsson's domain

At the start of 2004, we had started discussions with Ericsson's on the viability of a collaborative case study. We set about designing the methodology. During the method design we discovered a weakness in our approach. We only had data from a single source, and this data only gave us an insight of traceability in large organisations, who by Ramesh's definition were already "high-end" users. What about the smaller organisations? What was the state of the art and the challenges that they faced? To answer these questions we set about incorporating an industrial survey. The fact that this project had moved to Cape Town, it seemed the logical site to carry out a survey. Our connection to the Cape

Town SPIN (Software Process Improvement Network), with affiliations to the SEI, provided us with a perfect environment for gathering data from a variety of different companies. As traceability traverses the entire development organisation and as this network brought together software professionals from many of the software disciplines we set about gathering data. Once again we were concerned by the fact that this only gave us data from one country, therefore it was decided to utilise our professional connections in Ireland to broaden our perspective of traceability practices.

With little expertise in executing a survey, we evaluated earlier efforts. In 1994, at the 1st Conference on Requirement Traceability, Gotel and Finkelstein presented an empirical traceability study of 100 samples of data. They identified two problems with traceability; the lack of a *common definition of traceability* and the *conflicts in the understanding by the practitioners* of the traceability problem. What was most interesting about their method was they used grounded theory approaches to propose two new concepts in traceability namely; *pre-requirement specification* and *post- requirement specification*. They concluded by stating that many of the problems they identified were attributed to inadequate pre-requirement specification. In another survey in 1998, Ramesh investigating traceability across 26 software development organisations divided traceability users into two separate groups: low-end users and high-end users, which as the name implies it refers to the attitude they have towards traceability and the maturity of their practices. These two surveys, not only provided us with valuable classifications for our project, they also described the methods that they used and the approaches they followed to gather this data.

With little knowledge of the challenges that smaller companies faced we set about designing the questionnaires and interview scripts. From our experiences in Ericsson and based on the findings from the literature review, we knew that incomplete processes and tooling problems needed to be addressed by the survey. We were also interested to discover what the factors that influence traceability and how the attitudes of different roles in the development lifecycle had towards the importance of traceability. At this point we also had some preliminary ideas of using traceability patterns as describing different aspects of traceability. Therefore, in the survey we needed to investigate if following a pattern approach would add benefit to our project, by formalising the data gathering, analysis and presentation of our findings.

Before commencing the case study, it was paramount to decide what application domain we should use. Due to the contacts already made and the location of the design house we chose the Operation Support System (OSS) domain for the case study. We already knew that the developments of telecom systems are intrinsically difficult with complexities in the underlying technologies, standards and development environments that were perfect for research into traceability. Furthermore, we knew the history of requirement engineering in the OSS domain and that they were “high-end” users. The fact that the author already knew the technologies at hand and could begin the project with a knowledge base that many researchers in the same context, but without the same background knowledge would find difficult. We were confident that the OSS domain would provide an abundance of information that would make a valuable contribution to the research world.

Before starting the case study, we faced a number of key critical issues. How do we address confidentiality issues? What role would the researcher play, ethnographer or active participant? What were the objectives of the case study? How would the data help us with the development of the solution? By answering these questions we had set careful boundaries on the scope of the project. While the familiarity of the domain was a major

advantage it also posed its own specific problems. For example, the researcher had to redefine his role as ethnographer rather than active participant. However, as in life there is always a trade off. There were times that his expertise was requested in certain problem solving activities and in order to maintain good working relationship he agreed. This lead to a loss of time at certain points but on final analysis this time was repaid in good will by the management and the participants in the study.

It has been established that traceability is a practice that brings together all the disciplines in the entire software product development lifecycle. However, in essence this was as much a study on human interaction and the effects of managerial decisions on humans as it did on the impact of technologies on traceability. Large organisations must overcome the challenges of not only the management and understanding of vast amounts of information, but also the problems of design and development geographically dispersed development, cultural difficulties, complex organizational structures as well as dealing with changes in the market needs in order to develop quality products. With this background we will now review the findings of the Case Study and Survey.

## 12.3 FINDINGS OF CASE STUDY & SURVEY

The survey included a mix of companies of different sizes, from micro-sized ones of up to 9 people, to small ones of up to 50, and medium sized ones of up to 250. A small number of larger companies over 250 people were also included. Participants in the survey were drawn from all professions and disciplines in the development chain, from technical sales, and requirements engineers to product/project managers, to designers, developers and testers.

The survey was conducted in 2005. In all 83 persons participated in the survey from 19 organisations. On analysis of the results we discovered that only 18% of these smaller companies had a dedicated requirement management and traceability tool, however, 43% did have some form of informal approach, for example spreadsheets for managing requirements and traceability items. As we were investigating the use of process models, we needed to know the current state of the art of process modelling. While 87% of organisations had a development process only 9% used process modelling techniques. With regard to requirement engineering, 52% of the respondents believed that it was sufficiently defined in the organisation and a shocking 18% admitted that traceability was in fact sufficiently described in their processes. Requirement Management Plans in general were not produced. With a background in course delivery and mentoring we were interested in the influences that education had on the attitudes. This was difficult to capture by analysing questionnaires, therefore we used the interviews to extrapolate the opinions of the different software engineering roles. Although 79% of computer scientists did receive some form of requirement engineering education, the majority of those interviewed did not feel that this discipline was sufficiently covered in the course they underwent in university. This was particularly prevalent among the most experienced interviewees. Suggestions for improvements were made and better education was one of the key challenges that needed to be overcome in any solution that we would build. As expected the overall attitude of the respondents towards traceability varied greatly throughout the product development lifecycle. While the project managers, requirement managers and systems engineers, all regarded traceability as an important practice, what was most worrying was that only 30% of Senior Designers, 31% of Designers, 37% of Testers and 33% of System Testers, believed that traceability is in fact an important practice. Once again, this highlighted to us the need for a solution that was usable by all roles in the development lifecycle. Questions

like: how can we create a solution that promotes better communication? How can we make the resources not directly involved with the management and control of the project feel more involved in the overall process?

Finally, in the survey, we investigated the factors that influence traceability. We expected that the main factors would be tool and process related, however we were surprised to find that many of those involved stated that time, resource and budget restrictions were in fact the main reasons given for not practicing traceability. The findings of the main factors that influence traceability are best summarised as:

- *Lack of budget, time and resources* was the greatest influence.
- *Insufficient tooling* was given as the next greatest factor holding back progress.
- *Poor process definition*, particularly for requirements engineering and change control.
- *Lack of training* was believed to be of comparable influence.
- *Lack of commitment* from the management on matters related to traceability.

Therefore our solution must address these findings as design considerations. In conclusion, this survey not only provided the researcher with some interesting statistics for comparison with the data being collected in Ericsson, it also provided us with insights or considerations for our solution framework. Any solution that we propose must be scalable for small and large organisations.

On further analysis of the survey, in particular the interview scripts a number of common trends began to appear. For example, poor communication between the management team and the developers or testers, lack of training being undertaken, incomplete processes, lack of tools, inconsistencies in the terminology and concepts. While there were differences in the scale of the problems or the magnitude of the information impacting traceability, it was interesting to discover that many of the same problems emerged from the two samples. So, were the problems the same as those discovered in the case study in 2004? What was unexpected was the many similarities:

- *Lack of resources* was considered a significant problem by nearly 2 out of 3 at Ericsson, specifically in the requirement engineering discipline.
- *Tooling problems* were the dominant problem at Ericsson.
- *Poor processes* were a significant problem at Ericsson too.
- *Lack of training* was regarded as the second most serious problem.

Other factors also influenced Ericsson, which had their grounding more in the magnitude or complexities of developing telecom systems that smaller organisations do not face, included; *complex product structures, complex numbering systems*, *poor communication*, *lack of clarity in the definition of roles* and *responsibilities for traceability* and *negative attitudes* to traceability

### 12.3.1 Insights from Empirical Study

On further analysis of the case study data in 2004 it became clear that the OSS domain suffered other major obstacles that many smaller organisations would not face. From investigation during the Case Study, it became clear that in the OSS project at Ericsson, there were some further special factors at work.

Firstly, the OSS product family was exceptionally complex. In 2003, the OSS development programme was organised as three separate products namely; RANOS, CN-OSS and GSM-OSS, each product having separate market units, product development units and many different projects and sub-projects. The sheer scale of the overall OSS products

was mind blowing, with sixteen development sites scattered between several European nations, speaking different languages, and with different development practices and local standards and traditions. Another aggravating factor was the economic recession that took place in the IT market in 2001, which led to a reduction in staff numbers from approximately 120,000 employees in 2001 to 60,000 at the end of 2002. The effects of this reduction in staff numbers had a crippling effect on the organisation, not only on repressed IT budgets but it also resulted in staff shortages which inevitably effected requirement engineering and hence traceability. Furthermore, the repressed budgets and staff shortages had serious impacts on the delivery of process and requirement related training. Tight licencing agreements with third party suppliers, restricting the delivery of training except by certified instructors, whom in many cases did not fully understand the domain specific problems led to less course that did not address the needs of the user community.

However, a number of large changes between 2004 and 2007 had major impacts on the successful practice of traceability. The better economic climate led to better resource allocation, the investment in a propriety tool called MAR's, and better training all contributed to better practices and attitudes towards traceability.

In 2004, there was only one requirement manager assigned to requirement engineering responsibilities. His role was primarily one of requirement management than of requirement support. He defined the Requirement Management Plan, he collaborated with the process team to ensure that the OSS-RC process met the needs of the users, he interfaced with the stakeholders, the product team and the main project team on matters related to the management and control of the requirements. He did not have sufficient time for any administration functions, little time for ensuring the integrity of the requirements in the repositories and no time supporting individual team members on traceability. By 2007, there were four resources involved in requirement engineering functions. The three extra resources played a major support role. They supported the users, they carried out desk-to-desk calls and they provided mentoring to those who required it. The impact and influence such a strategy had on the attitudes and confidence of the users was staggering. This was reflected by the number of extra licences used by the OSS-RC, from 20 in 2004, to 250 in 2007. The management decision to invest in resource allocation definitely paid of.

The next major change, and possibly the one that had the biggest impact was the introduction of the propriety tool, MAR's, which supported good cross-product and cross project functionality providing a single environment for the entire product development unit. Interestingly, the stakeholder's requirements and the product requirements were stored in Focal Point, but good synchronous functionality provided easy movement of requirements between tools. Change management and impact analysis activities were easily integrated using this tool. In addition, the increase in resource allocation also provided support to the every day user.

In addition, Ericsson adopted a "three-to-one" product strategy to integrate RANOS, CN-OSS and GSM-OSS into one product, OSS-RC. This reduced the number of design houses from sixteen to three, which greatly simplified the organisational structure. Instead of separate market units, product units, product management team and a multitude of projects, the result was that one single organisation now managed all aspects of the product. That led to better communication between the success critical resources and solved many problems, for instance, the duplication of requirements, the complexities in document or product numbering and the unification of processes.

One of the major differences between Ericsson's, the large high-end aspect of this study and the smaller organisations was the existence of a traceability corporate strategy. While this strategy did not mandate a tool it did provide the rules on many matters related to traceability, including the criteria for dealing with third party suppliers, the relationship between the hardware and the software, the shipping and product numbers. This corporate strategy was issued to all products being developed not just the OSS aspect of the domain, and in our opinion it was the existence of such a directive that distinguishes between high-end and low-end users. It became clear that the interrelationship between corporate decisions, product strategies and organisational or institutional factors and environmental influences all played a major role in the development of successful traceability solutions. Therefore we concur with Ramesh's earlier hypothesis that organisational factors, institutional and environmental factors to all impact traceability. However, the human factor involved in traceability is still the number one criteria for success. The engineers, must receive the correct training and must have confidence in the tools and processes before traceability has any chance to succeed.

## 12.4 FINDINGS ON SOLUTIONS

Using the evidence from the literature and mapping techniques from the problem or design considerations gathered from both the survey and the case study we set about designing, implementing and testing the solution framework which comprises of three separate but interrelated components namely; a semantic model, a process model and the intuitive but novel traceability patterns. The solution needed to address the problems of poor communication, poor training situations, reusability of practices and knowledge, inconsistent terminology and the simplification of complex concepts.

Following an iterative approach, the testing activity commenced early and not only provided us with essential validation results they also gave us insights into future directions with the design. For example, the decision was made to split the semantic and process model as feedback due to the complexities of the models against one of objectives that our solution was to simplify traceability concepts that could be understood by all participants in the development lifecycle. The TRAceability Process (TRAP) primary objective was to define the main process elements that assisted any organisation to create a traceability process namely; phases, lifecycles, roles, responsibilities, guidance and traceability activities. From our previous experience, the literature review and the initial results from the empirical studies we evaluated that UML provided the best language for creating reusable, extensible, maintainable and easily understood models. While SPEM, had not been previously used in traceability research, we evaluated that it provided the best metamodel based approach for creating generic processes adaptable by any organisation. SPEM and the OMG's Model Driven Architecture both recommend a layered approach so we set about creating models that provided abstractions that offered a solution or different perspective depending on the layer. At the topmost layered or meta-layer, the models defined the concepts and their relationships in a platform and domain independent manner. At the lower levels we instantiated the metamodels from an organisational (M1) or project (M0) perspective. Because UML is not adapted for traceability, it was evaluated that UML profiles gave us the flexibility to extend UML without losing conformance characteristics while still allowing us to describe all concepts related to traceability. Similarly, with the TRAceability Model (TRAM) we designed the semantic models using UML, the OMG's MDA layered approach focusing on traceability semantics, the traceability data and the manipulation of this data. Once again the lower levels described real-world enactments.

In the design of our methodology we kept an open mind for the emerge of new theory or grounded theory. The traceability patterns were just that, new theory that emerged from the empirical study and the modelling efforts. Very early into the model designs we observed recurring structures emerging from the models. By giving these patterns a name, and formalising the descriptions of the recurring problems and by using a template, we had a new approach for describing traceability problems and the solutions should one exist. While the traceability patterns emerged during the modelling stages of the semantic and process models they also provided a unique approach for representing empirical findings, offering an effective means of communicating traceability best practices or recurring traceability problems to a diverse group of people involved in anyway with traceability. We propose an iterative methodology for using the traceability patterns that consists of five main stages, *information gathering, analysis, design, implementation and evaluation.* Following these stages traceability user's at any level of competency can describe experience-based knowledge that is formalised and easily communicated, maintained and simplifies even the most complex traceability issue. In addition the patterns could be used to promote cross-discipline communication, something that is relevant to traceability because of the involvement of software engineers from several domains. We presented a generic template for the formal specification of patterns and supported our work with a number of examples from the telecoms domain.

Finally, we evaluated the proposed traceability framework using both laboratory test methods and field trials and experiments. Validation of the different layers provided us with one of the challenges of using the MDA's layered approach. Creating models at different layers of abstraction is effectively a new perspective on modelling with little supporting validation or empirical evidence that supports the testing process of each layer. Questions like, who should validate the different abstractions, when should it be done and how do the results of one layer effect another all needed to be addressed throughout this study. Furthermore, while models of traceability had been created before the novelty of our approach, the testing problem was an essential obstacle to overcome. In the laboratory trials we carried out deskchecks, participatory controlled lab trials including cognitive walkthroughs with industrial participation to evaluate the high-level metamodels or profiles. In 2005, to gain an insight into future directions with the TRAP, in fact to make a "go", "no-go" decision on whether to continue with developments on the process, we utilised the ISO 15504 to compare the maturity of the TRAP against the capabilities of a commercially available process, the Rational Unified Process. In the field trials we once again assessed the models and the patterns except this time in the Ericsson's domain and broadening the test scope to include all three layers M2, M1, M0. Once again we carried out cognitive walkthroughs and got feedback from participatory workshops. Furthermore we utilised the CMMI assessment framework which assisted us gain understanding of the traceability practices as defined by a widely accepted standard and for judging the maturity of the TRAM and TRAP. Utilising CMMI assessors from with Ericsson we evaluated the maturity of our solution. The main contribution of using both ISO 15504 and CMMI was that both the CMMI and ISO 15504 assessment framework provide a suitable approach for assessing traceability models and processes.



## 12.5 CONCLUSIONS ON SOLUTION

When modelling new concepts and applying new techniques, the most basic primary objective must be; is this a viable approach that produces scientific results that can be recreated in a similar context. In this study not only do we apply new layered approaches, new and complex languages based on new standards and specifications, but we

also apply it to the traceability domain which largely is unchartered territory. The choices to use a modelling approach were firmly grounded on the best available standards and yet to prove that this approach is the best solution is very difficult to achieve. To prove that it adds benefits in a context was a simpler task. What we did demonstrate is that it is possible to model traceability at different levels of abstraction from different perspectives using different types of models; from semantic models to process models. We also proved that it is possible to use the OMG's modelling standards, their MDA approach and the SPEM metamodel, and this is a main achievement of this study. Furthermore, we made some very practical observations along the way. Separating process models and semantic models reduced the complexity considerably and increased the understandability of the models. Suppressing the attributes and operations in certain contexts also proved to be the best approach. Of course, when evaluating models with a modelling expert this is not the case, but when evaluating the models with a project team including; process team members, project management, requirement management, design and test members it is better to display only the attributes or operations that add meaning to the model, or by their existence they simplify the model.

One could argue that we merely created a pictorial representation of a complex domain at different levels of abstractions but in its essence this where the success lies. We proved that it was possible. Using the layers proved to be the right approach, especially in large organisations who develop enterprise applications with complex product families. Each layer provides a use or benefit from a different perspective. They helped with gaining agreement on matters related to traceability, they stimulated conversations and promoted knowledge sharing and while we did not prove directly that the solutions are reusable, based on the abundance of literature on creating models we are confident that in fact these solutions are reusable. For smaller organisations who face time and resource constraints, a layered approach may be an overly complex approach not worthy of the results that is obtainable. They should however, adapt this work, perhaps reusing the metamodels (or Profiles) that we provide and go straight to the project instantiation layer, or M0. In essence, the M1 layer represents the difference between the large or small organisations. Smaller organisations do not have to deal with many of the challenges faced by the larger organisations, for example, complex product structures or organisational issues. In its essence the M1 layer is the organisational layer.

Another aspect of our modelling effort is the use of UML Profiles. UML comes with a standard set of modelling elements and understandably it does not support many of the elements that are needed to describe traceability. The UML Profile, extension mechanism, gave us the flexibility to use the UML constructs, rules and constraints and the freedom to define our own elements to capture the different traceability elements that we needed in this context. Overall, we would recommend the use of UML Profiles in future traceability modelling research.

Using a modelling approach, we had to overcome many challenges along the way, the major one of gaining agreement amongst a variety of users and in the process sharing information or knowledge on the topic of traceability. However, getting access to the expertise to validate our models that satisfied conformance to standards criteria on one hand and assessing the usability of the models on the other hand proved a major obstacle to overcome. The best solution was to triangulate the expertise, having modelling experts and practitioners on the same review. This proved to be the best approach, in that by gaining agreement from both helped us overcome the dichotomy between simplicity and usability versus conformance and correctness.

Models should provide a solution to a problem in a context. As this is a project about traceability it is appropriate to illustrate that the solutions provided, do indeed map to the problems identified. In Figure 11-1 below*, Problems Traced to Solutions*, we illustrate that the problems that were identified are traced to the solutions that we proposed in the Traceability Framework. We kept a clear focus on simplicity and usability while testing the modelling standards and specifications on their promises of solving a problem. In this approach we succeeded. Modelling traceability is certainly the right approach for future projects. However, the researcher must understand the context and the competence of the organisation that they are designed for. We fear that in far too many cases the models are only designed without a vision of the objectives or users and in many cases are only understandable by a select few. Therefore, before any decision for modelling of traceability, we strongly recommend that they follow the process outlined in this study, and ensure that whatever the modelling situation that the modeller fully understands the problems they are trying to overcome, the objectives of their work and the outcome that they expect.

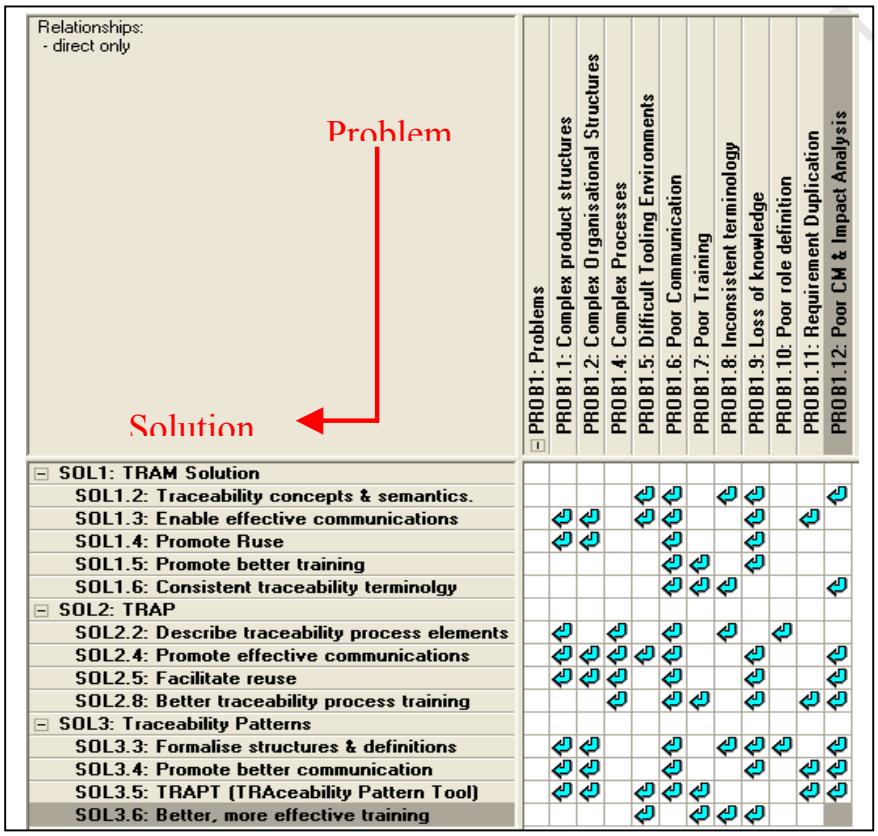



**Figure 12-1 Problems Traced to Solution**

Assessing and evaluating the models and the process was the next challenge we faced. The first step was lab experiments, for instance desk checks and participatory cognitive walkthroughs, to test and validate the meta-models. The next intuitive step was to test the models in the field, with participants who had input into their design testing for conformance, usability, understandability and so on. But we needed an assessment framework to test and measure the capability or maturity of the traceability practices that we propose. And this raised a number of questions. We were proposing traceability practices, at a process and semantic level, how then would we test these practices? In general, traceability measurements and benchmarks is a fuzzy area. Only in the last couple

of years did a traceability community even emerge therefore there are few standards that define traceability measurements across the entire development lifecycle.

An unforeseen challenge arose when it came to evaluate the framework in Ericsson. They were using CMMI. In the beginning it caused problems with learning a new and complex standard but in the end the process involved in the evaluation proved to be to our benefit. Once again we learned how to evaluate the capabilities and maturity of our work using assessors from Ericsson, and we gained a better insight into further definitions of traceability that needed to be incorporated into our solution. So what is our contribution by applying these standards? Of course the results obtained but we also addressed an issue of traceability that needs further investigations in future research projects. A number of researchers have called for the need for a standard for benchmarking or measuring aspects of traceability. Then why not take these two standards and reuse the aspects that are useful to this cause. The result would be a Traceability Standard that could be used in any traceability situation. Unquestionably, this is an area for future research. In 2005, Ericsson, assessed using CMMI their requirement engineering process to be at Level two. In early 2008, another assessment increased this maturity level to Level three. While we are not claiming that this was directly because of the discussions or improvements as a result of our work. However, one of the main reasons for this increase was due to the fact that they created process models, not dissimilar to the TRAM and TRAP models that we created. It does highlight though that following a modelling approach does improve the overall practice of traceability.

The final aspect of our solution was the traceability patterns. This was an intuitive component that simply emerged, perhaps because of the grounded theory method that we followed, perhaps because we clearly understood the problems that traceability practitioners faced and this was an obvious solution. While patterns have been used in almost every facet of software design and development, they did provide a number of interesting results. They solved many of the communication issues, the loss of traceability issues due to staff attrition, the reusability of successful practices, they formalised an approach for consistent terminology and they provided a solid basis for course delivery on requirement engineering and traceability. The patterns that we identified emerged from the model structures we were creating, and help simplify or breakdown the models into easy to understand components. But also they addressed some of the human challenges that practitioners face with traceability. In this study they provided us with a formalised approach for gathering information on traceability. We distributed a pattern template during the empirical study and we observed that the personnel would document their experiences once they were given the tool to do so. In earlier work we developed the TRAPT (TRAceability Pattern Tool) for creating, categorising and storing the traceability patterns in an easy to use, searchable tool. Validating these patterns was more an experiment in human behaviour than a technical study. We carried out lab experiments with students and experiments in Ericsson which provided us with preliminary results. However, the main proof we have that these patterns do provide a benefit to the traceability community, lies in the fact that most of the examples we describe were captured as real-world experiences in the field. We do not suggest that these patterns are the silver bullet that solve all the traceability problems, what we do suggest is that in the context of this study they provided us with a very valuable new approach for problem solving and describing traceability. We would strongly recommend in future research projects that the approach we followed and the patterns that we created be applied in further application domains.

In conclusion, this research designed and tested a Traceability Framework consisting of three components; a semantic TRAceability Model (TRAM), a TRAceability Process (TRAP) model, and Traceability Patterns. Overall we would strongly recommend based on the findings of this study that indeed these three components integrated into the same solution space does in fact mitigate some of the problems identified in this study and more importantly it provides a framework for future research. The final question: could this become a product in the market place? In its current format, probably not, but by carefully building good interfaces, better traceability modelling capabilities and by improving the pattern development interfaces, this has some viable possibilities. However, further testing and real life applications would be required and further research by the traceability community could make this a product worthy of the market place.

## 12.6 FUTURE WORK

The Traceability Framework proposed in this project is a first attempt at integrating traceability process, semantics and traceability patterns into one solution space. As already described in earlier sections of this chapter there is an abundance of research possibilities and further work that can be carried out from our contributions in this study.

In the short term it is essential that this framework be applied to a number of different applications domains using similar processes. Only then will we truly understand the potential of this framework. At the meta-level this would require an evaluation of the metamodels and UML profiles proposed by a group of experienced traceability researchers. The models at the lower levels (M1 and M0), which are suitable as a reference should not be used by researchers or practitioners unless they are adapted further for the specific domain. Therefore, the single most important future direction must be further applications of the framework.

It is felt that in the long term it may be possible to develop a *grand unified traceability framework* which as well as combining the components described in this study would also incorporate solutions to the challenges reported in the Grand Challenge Report. While we have addressed some of the challenges that the traceability community face, for instance, transferring traceability knowledge, training and certification, link semantics and human factors, there are further challenges that could be incorporated into the framework. For example, measurements and metrics for resource allocation and cost benefit analysis. While we did not address either of these challenges we believe that our approach could be used for generating and maintaining traces that takes into consideration factors such as time, effort and system quality. Or metrics that would calculate the cost benefit of applying a certain technique or value add of certain patterns.

The lack of empirical traceability data is a very worrying problem for the traceability community. This report highlights a number of factors that we believe need further investigations in the field. What is the impact of complex product families in domains other than the telecom's domain? What is the effect of traceability corporate strategies on the successful practice of traceability? What is the impact of executive decisions on organisational structures on implementing traceability? Should organisations develop their own in-house traceability tools? What is the recommended number of requirement engineers and managers to the rest of the development organisations? This research raises as many questions as it does provide answer. It is of the utmost importance that further studies be undertaken. When we started this study we were shocked by the lack of recommended empirical methods was at our disposal to collect traceability data in the

field. Further reports on the successes or failures of traceability needs to take place. However, we do recommend the approach that we followed. Traceability is a complex business and the best way of getting to the core of the problems is to mix qualitative and quantitative methods together, to apply ethnographic and participatory techniques and in our case to include grounded theory in the defined method. To analyse the results, data triangulation proved to be the best approach. These suggestions need further validation.

While the survey that we carried out provided us with a lot of good information on the state of the art of traceability in small and medium sized organisations, as the results of our survey clearly show further investigations are needed. More evidence on the factors that influence traceability, further investigations on the problems that they face and more data on the relationship between education or training and the influences that this has on the attitudes towards the importance of traceability and the impact of better training programs from small organisations needs to be undertaken. As this paper goes to publication there is a lot of talk that we are entering into another recession. Questions like the impacts this has on smaller companies, the effects of repressed IT budgets and the impact this has on all software engineering practices and in particular traceability needs further investigations. While we did not investigate the impact of outsourcing strategies on traceability is very prevalent today. Software development is now very much a global village and we need to understand what this means to the practice of traceability.

We introduce traceability patterns, we propose a methodology for the creation and use of the patterns and we carried out some basic testing. The traceability community would need to embrace some or all of these pattern, produce more examples and carry out further tests before we could propose this as a best practice to industry. The opportunities that arise from these traceability patterns are endless. In particular we believe that while patterns as a pedagogical tool for communicating best practices add great benefit, the fact that traceability is the law of adding constraints then further investigations into constraint languages should be undertaken. The traceability patterns that we propose are formalised approaches for grouping rules and decisions in relation to traceability. Because we use UML Profiles to extend the UML vocabulary and hence describe the patterns it is not required to use a constraint language to describe the constraints. However, by using the Object Constraint Language (OCL), provided by IBM, which is a precise text language that provides constraints, could have far reaching consequences on formalising the patterns. In fact OCL, could be applied to the TRAM and TRAP providing a formalised approach for representing the rules and constraints on the models.

# REFERENCES


AIZENBUD-RESHEF, N., PAIGE, R., RUBIN, J., SHAHAM-GAFNI, Y. & KOLOVOS, D. (2005) Operational Semantics for Traceability. *ECMDA-Traceability Workshop, Nuremberg*, 7-14.

ALEXANDER, C. (1979) *The timeless way of building*, Oxford University Press.

ALEXANDER, C., ISHIKAWA, S. & SILVERSTEIN, M. (1977) *A pattern language: towns, buildings, construction*, Oxford University Press.

ALEXANDER, I. (2002) Towards automatic traceability in industrial practice. *Proc. of the 1st Int. Workshop on Traceability*. In conjunction with the 17th IEEE International Conference on Automated Software Engineering, Edinburgh, U.K, IEEE Computer Society Press.

AMBLER, S. (1999) Trace your design. *Software Development,* 7**,** 48-55.

APPLETON, B. (1997) Patterns and Software: Essential Concepts and Terminology. *Object Magazine Online,* 3**,** 20-25.

APPLETON, B. (1998) Patterns and Software: Essential Concepts and Terminology. *White Paper*.

APPLETON, B. (2000) Patterns and Software: Essential Concepts and Terminology.

ARC-ADVISORY-GROUP (2007) ARC Advisory Group Official Website.

ARKLEY, P., MANSON, P. & RIDDLE, S. (2002) Position Paper: Enabling Traceability. IN IEEE (Ed.) *1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*. In conjunction with the 17th IEEE International Conference on Automated Software Engineering, Edinburgh, U.K, IEEE Computer Society Press.

ARKLEY, P. & RIDDLE, S. (2005) Overcoming the Traceability Benefit Problem. *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, 385-389.

ARKLEY, P., RIDDLE, S. & BROOKES, T. (2006) Tailoring Traceability Information to Business Needs. *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)-Volume 00*, 234-239.

ASSUMPTIONS, C. O. (1998) Experiences in the Adoption of Requirements Engineering Technologies. *Crosstalk*.

BABBIE, E. (2001) *The Practice of Social Research,* Wadsworth, Thomson Learning Inc.

BENCOMO, A. (2005) Extending the RUP, Part 1: Process Modeling. IN IBM (Ed.).

BENDRAOU, R., GERVAIS, M. P. & BLANC, X. (2005) UML 4 SPM: A UML 2. 0-based metamodel for software process modelling. *ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems, UML*, 2-7.

BORCHERS, J. O., FINCHER, S., GRIFFITHS, R., PEMBERTON, L. & SIEMON, E. (2001) Usability pattern language: Creating a community. *AI & Society,* 15**,** 377-385.

BUDINSKY, F. J., FINNIE, M. A., VLISSIDES, J. M. & YU, P. S. (1996) Automatic Code Generation from Design Patterns. *IBM Systems Journal,* 35**,** 151-171.

BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P. & STAL, M. (1996) *Pattern-oriented software architecture: a system of patterns*, John Wiley & Sons, Inc. New York, NY, USA.

CARIMO, R. A. (2006) Evaluation of UML Profile for Quality of Service from the User Perspective. *School of Engineering.* SE – 372 25 Ronneby, Blekinge Institute of Technology, Sweden.

CHARLES F. GOLDFARB, STEVEN R. NEWCOMB, W. ELIOT KIMBER & NEWCOMB, P. J. (1997) Information processing -- Hypermedia/Time-based Structuring Language (HyTime) - 2d edition (ISO/IEC 10744:1997). IN ISO/IEC (Ed.), ISO/IEC.

CLELAND-HUANG, J. & SCHMELZER, D. (2003) Dynamically Tracing Non-Functional Requirements through Design Pattern Invariants. *Workshop on Traceability in Emerging Forms of Software Engineering*. In conjunction with IEEE International Conference on Automated Software Engineering, Montreal, Canada, IEEE Computer Society Press.

CMMI (2006) CMMI for Development. Pittsburgh, PA, Software Engineering Institute.

COLIN ATKINSON, BRIAN HENDERSON-SELLERS & KÜHNE, T. (2001) To Meta or Not to Meta—That Is the Question. *Application Development Trends Articles*

COPLIEN, J. O. (1996) *Software Patterns*, SIGS Books & Multimedia.

COPLIEN, J. O. (1997) Idioms and patterns as architectural literature. *Software, IEEE,* 14**,** 36-42.

COPLIEN, J. O. & SCHMIDT, D. C. (1995) *Pattern languages of program design*, ACM Press/Addison-Wesley Publishing Co. New York, NY, USA.

DENZIN, N. (1984) *The research act.*, Englewood Cliffs, NJ: Prentice Hall.

DICTIONARY, O. E. (2006) Oxford English Dictionary. *Forum for Modern Language Studies*.

DÖMGES, R. & POHL, K. (1998) Adapting traceability environments to project-specific needs. *Communications of the ACM,* 41**,** 54-62.

DOMGES, R., & POHL, K. (2009) "Adapting Traceability Environments to Project Specific Needs." Communications of the ACM 41.12 (2009): 55- 62

DOWSON, M. (1993) Software process themes and issues. *Software Process, 1993.'Continuous Software Process Improvement'., Second International Conference on the***,** 54-62.

EASTERBROOK, S., SINGER, J., STOREY, M. A. & DAMIAN, D. (2007) Selecting Empirical Methods for Software Engineering Research.

ECKLUND JR, E. F., DELCAMBRE, L. M. L. & FREILING, M. J. (1996) Change cases: use cases that identify future requirements. *Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications***,** 342-358.

EDWARDS, M. & HOWELL, S. (1991) A methodology for system requirements specificationand traceability for large real-time complex systems. Technical report, U.S. NavalSurface Warfare Center Dahlgren Division.

EGYED, A. & GRUNBACHER, P. (2002) Automating requirements traceability: Beyond the record & replay paradigm. *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on.* In conjunction with the 17th IEEE International Conference on Automated Software Engineering, Edinburgh, U.K, IEEE Computer Society Press.

ESPINOZA, A., ALARCON, P. P. & GARBAJOSA, J. (2006) Analyzing and Systematizing Current Traceability Schemas. *30th Annual IEEE/NASA Software Engineering Workshop*

EUROPEAN-COMMISSION (2002) REGULATION (EC) No 178/2002 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 28 January 2002 laying down the general principles and requirements of food law, establishing the European Food Safety Authorityand laying down procedures in matters of food safety. IN THE EUROPEAN PARLIAMENT AND THE COUNCIL OF THE EUROPEAN UNION (Ed.), Official Journal of the European Communities.

FEAGIN, J., ORUM, A., & SJOBERG, G. (1991) A case for case study. Chapel Hill, NC, University of North Carolina Press.

FOLMER, E., WELIE, M. & BOSCH, J. (2006) Bridging patterns: An approach to bridge gaps between SE and HCI. *Information and Software Technology,* 48**,** 69-89.

FORFÁS (2005) International Trade & Investment Report, 2004. IN ENTERPRISE, T. A. E. (Ed.), THE NATIONAL POLICY AND ADVISORY BOARD FOR ENTERPRISE, TRADE, SCIENCE, TECHNOLOGY AND INNOVATION.

FOUCAULT, M. (1984) *'Nietzsche, Genealogy, History',* London, Penguin.

GARCIA, F., RUIZ, F. & PIATTINI, M. (2003) Metamodeling and measurement for the software process improvement. *Computer Systems and Applications, 2003. Book of Abstracts. ACS/IEEE International Conference on*.

GILLS, M. (2005) Survey of Traceability Models in IT projects. *ECMDA Traceability Workshop, Nuremberg, Germany*.

GLASER, B. G. (1967) *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine de Gruyter.

GOTEL, O. & FINKELSTEIN, A. (1994a) Modelling the Contribution Structure Underlying Requirements. *Proc. of the 1st Int. Conf. on Requirements Engineering (ICRE).* IEEE Computer Society

GOTEL, O. C. Z. & FINKELSTEIN, A. C. W. (1994b) An Analysis of the Requirements Traceability Problem. *Proceedings of the First International Conference on Requirements Engineering (ICRE'94).* Colorado Springs, Colorado, USA, IEEE Computer Society Press.

HAYES, J., CLELAND-HUANG, J. & DEKHTYAR, A. (2006) Grand challenges in traceability. Fairmont, West Virginia. , Center of Excellence for Traceability.

HAYES, J., DEKHTYAR, A., International Journal of Software Engineering and Knowledge Engineering (IJSEKE), Volume: 15, Issue: 5 (2005) pp. 751-781

HAZELINE, U., FRÉDÉRIC FRANÇOIS, A. & TAYLOR, R. N. (2007) Development processes and tools: An end-to-end industrial software traceability tool *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering ESEC-FSE '07* ACM Press

HERRMANN, T., HOFFMANN, M., JAHNKE, I., KIENLE, A., KUNAU, G., LOSER, K. U. & MENOLD, N. (2003) Concepts for usable patterns of groupware applications. *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work***,** 349-358.

HU, Z. & VOLLMAR, G. (2001) Towards XML Metamodel Patterns for XML Data Modeling. *Proceedings of the 12th International Workshop on Database and Expert Systems Applications table of contents***,** 71-74.

HUMPHREY, W. (1989) *Managing the Software Process,* Reading, Mass.

ICT, I. (2006) Leadership in Information and Communications Technology. IN ICT (Ed.) Dublin.

IEEE (1998) IEEE Guide for Information Technology - SystemDefinition - Concept of Operations (ConOps) Document. . New York, IEEE

ISO/IEC (1998 ) ISO/IEC TR 15504: 1998, Information technology - Software process assessment,Parts 1 - 9.

ISO/IEC (2003) ISO/IEC 13250:2003 Information technology -- SGML applications -- Topic maps IN INTERNATIONAL STANDARDS FOR BUSINESS, G. A. S. (Ed.), ISO.

ISO/IEC (2006) ISO/IEC 13250-2:2006 Information technology -- Topic Maps -- Part 2: Data model. IN INTERNATIONAL STANDARDS FOR BUSINESS, G. A. S. (Ed.).

ISO/IEC (2007a) How are ISO standards developed?

ISO/IEC (2007b) ISO/IEC 13250-3:2007 Information technology -- Topic Maps -- Part 3: XML syntax. IN INTERNATIONAL STANDARDS FOR BUSINESS, G. A. S. (Ed.), ISO ( International Organization for Standardization) and IEC (the International Electrotechnical Commission).

JACOBS, S. & EUROLAB, E. (1999) Introducing measurable quality requirements: a case study. *Requirements Engineering, 1999. Proceedings. IEEE International Symposium on.* Limerick, Ireland.

JARKE, M. (1998) Requirements tracing. *Communications of the ACM,* 41**,** 32-36.

KELLEHER, J. (2005a) A Method for Modelling a Mature Process. *Software Process Improvement · 12th European Conference, EuroSPI 2005.* Budapest, Hungary, Springer.

KELLEHER, J. (2005b) A reusable traceability framework using patterns. *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering***,** 50-55.

KELLEHER, J. (2006a) Traceability Patterns. *4th ECMDA Traceability Workshop* Bilbao, Spain, OMG.

KELLEHER, J. (2006b) Utilizing use case classes for requirement and traceability modeling. *Proceedings of the 17th IASTED international conference on Modelling and simulation***,** 617-625.

KELLEHER, J., HOLLINGS, D. & SHARPEY-SCHAFER, K. (2005) Representing Software Traceability using UML and XTM with an investigation into Traceability Patterns. University of Cape Town.

KELLEHER, J., POTT, A. & BERMAN, D. (2004) TraPT: A Traceability Pattern Tool. University of Cape Town.

KHOLKAR, D., KRISHNA, G. M., SHROTRI, U. & VENKATESH, R. (2005) Visual specification and analysis of use cases. *Proceedings of the 2005 ACM symposium on Software visualization***,** 77-85.

KRUCHTEN, P. (2003) *The Rational Unified Process: An Introduction*, Addison-Wesley Professional.

LEA, D. (1998) Christopher Alexander: An Introduction for Object-Oriented Designers. *The Patterns Handbook: Techniques, Strategies, and Applications*.

LETELIER, P. (2002) A Framework for Requirements Traceability in UML-based Projects. *1st International Workshop on Traceability in Emerging Forms of Software Engineering.* In conjunction with the 17th IEEE International Conference on Automated Software Engineering, Edinburgh, U.K, IEEE Computer Society Press.

LIMÓN, A. E. & GARBAJOSA, J. (2005) The Need for a Unifying Traceability Scheme. *Proc. Traceability Workshop, European Conference in Model Driven Architecture (EC-MDA)***,** 47–55.

LINDVALL, M. (1994) *A study of traceability in object-oriented systems development*, Dept. of Computer and Information Science, Linköping University.

LINDVALL, M. (1997) An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Systems Evolution. PhD thesis.

LINDVALL, M. & SANDAHL, K. (1996) Practical Implications of Traceability. *Software Practice and Experience,* 26**,** 1161-1180.

LINDVALL, M., SANDAHL, K., CARLSHAMRE, P., REGNELL, B. & NATT OCH DAG, J. (2001) An Industrial Survey of Requirements Interdependencies in Software Product Release Planning. *Fifth International Symposium on Requirements Engineering***,** 27-31.

Mason, P. Saeed, A. Arkely, P. Riddle, S (2003) Meta-modelling approach to traceability for avionics: a framework for managing the engineering of computer based aerospace systems, ISBN: 0-7695-1917-2, 233- 246

MA, J. & WANG, Y. (2006) A Quantitive Context Model of Software Process Patterns and Its Application Method. *Proceedings of the Sixth International Conference on Quality Software*, 243-250.

MAK, J. K. H., CHOY, C. S. T. & LUN, D. P. K. (2004) Precise modeling of design patterns in UML. *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, 252-261.

MARTIN, P. Y. & TURNER, B. A. (1986) Grounded Theory and Organizational Research. *The Journal of Applied Behavioral Science,* 22, 141.

MELTZOFF, J. (1998) Critical Thinking About Research: Psychology and Related Fields.

. *Washington DC: American Psychological Association.*

MINGES, M., SIMKHADA, P. (2003) The Evolution to 3G Mobile-Status Report. . International Trade Union.

MODELPLEX.ORG (2007) MODELPLEX Project Home Page.

MYERS, M. D. (1997) Qualitative Research in Information Systems. *Management Information Systems Quarterly*

NOOR, N. M. M., PAPAMICHAIL, K. N. & WARBOYS, B. (2003) Process Modeling for Online Communications in Tendering Processes. *Proceedings of the 29th EUROMICRO Conference'New Waves in System Architecture'. IEEE Computer Society*, 17-24.

OMG (2005) Software Process Engineering Metamodel Specification.

OMG (2007) UML Infrastructure Specification, v2.1.1.

OSTERWEIL, L. (1987) Software processes are software too. *Proceedings of the 9th international conference on Software Engineering*, 2-13.

PAULK, M. C., WEBER, C. V., CURTIS, B. & CHRISSIS, M. B. (1995) *The capability maturity model: guidelines for improving the software process*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

PEPPER, S. & MOORE, G. (2001) XML Topic Maps (XTM) 1.0. *TopicMaps. org Specification*.

PÉREZ-MIÑANA, E., TREW, T. & KRAUSE, P. (2002) Issues on the Composability of Requirements Specifications for a Product Family. IN PRESS, I. C. S. (Ed.) In conjunction with the 17th IEEE International Conference on Automated Software Engineering, Edinburgh, U.K, IEEE Computer Society Press.

POHL, K. (1996) PRO-ART: Enabling Requirements Pre-Traceability. *Proceedings of ICRE,* 96.

PYECHA, J. (1988) A case study of the application of noncategorical special education in two states. Chapel Hill, NC: Research Triangle Institute.

RAMESH, B. (1998) Factors influencing requirements traceability practice. *Communications of the ACM,* 41, 37-44.

RAMESH, B. & JARKE, M. (2001) Toward reference models for requirements traceability. *Software Engineering, IEEE Transactions on,* 27, 58-93.

RATIONAL, I. (2007) Requisite Pro Product Description.

ROBINSON, H., SEGAL, J. & SHARP, H. (2007) Ethnographically-informed empirical studies of software practice. *Information and Software Technology,* 49, 540-551.

ROLLAND, C. (1998) A Comprehensive View of Process Engineering. . IN PERNICI, C. T. E. (Ed.) *Proceedings of the 10th International Conference CAiSE'98*. Pisa, Italy, Springer.

SANDAHL, K., BLOMKVIST, O., KARLSSON, J., KRYSANDER, C., LINDVALL, M. & OHLSSON, N. (1998) An Extended Replication of an Experiment for Assessing Methods for Software Requirements Inspections. *Empirical Software Engineering,* 3**,** 327-354.

SCHWARZ, H., (2009) "Towards a Comprehensive Traceability Approach in the Context of Software Maintenance," csmr, pp.339-342, 2009 European Conference on Software Maintenance and Reengineering.

SEEDORF, S., NORDHEINER, K., KRUG, S., (2009) "STraS: A Framework for Semantic Traceability in Enterprise-wide SOA Life-cycle" 13th Enterprise Distributed Object Computing Conference.

SEFIKA, M., SANE, A. & CAMPBELL, R. H. (1996) Monitoring compliance of a software system with its high-level design models. *Proceedings ICSE,* 18**,** 387–396.

SEI (2007a) Concept of Operations for the CMMI. Carnegie Mellon University, Pittsburgh, Software Engineering Institute.

SEI (2007b) A Framework for Software Product Line Practice. Carnegie Mellon University, Pittsburgh, Software Engineering Institute.

SHERBA, S. A., ANDERSON, K. M. & FAISAL, M. (2003) A Framework for Mapping Traceability Relationships. *Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering.* IEEE Computer Society Press.

SHROTRI, U., BHADURI, P. & VENKATESH, R. (2003) Model checking visual specification of requirements. *Software Engineering and Formal Methods, 2003. Proceedings. First International Conference on***,** 202-209.

SIMPSON, J. A. & WEINER, E. S. C. (1989) *The Oxford English Dictionary*, Clarendon Press Oxford.

SMITH, A. (1776) *AN INQUIRY INTO THE NATURE AND CAUSES OF THE WEALTH OF NATIONS,* Scotland.

SMMDA (2006) International Workshop on Semantic Modelling and Model-Driven Architecture

*European Conference on Model Driven Arctitecture.* Bilbao, Spain.

SOMMERVILLE, I. & SAWYER, P. (1997) *Requirements Engineering: A Good Practice Guide*, John Wiley & Sons, Inc. New York, NY, USA.

SPANOUDAKIS, G., AVILLA GARCEZ, A. & ZISMAN, A. (2003) Revising Rules to Capture Requirements Traceability Relations: A Machine Learning Approach. *Proceedings of the 15th International Conference in Software Engineering and Knowledge Engineering (SEKE 2003), San Francisco, USA***,** 570–577.

SPANOUDAKIS, G., ZISMAN, A., PÉREZ-MIÑANA, E. & KRAUSE, P. (2004) Rule-based generation of requirements traceability relations. *The Journal of Systems & Software,* 72**,** 105-127.

SPENCE, I. & PROBASCO, L. (1998) Traceability Strategies for Managing Requirements with Use Cases. *White Paper, Rational Software Corporation*.

STABLEIN, R. (2006) *Data in Organization Studies*

London: Sage.

STAKE, R. (1995) *The art of case research.* , Thousand Oaks, CA: Sage Publications.

STANDISH_GROUP (1995) The Chaos Report. . IN GROUP, T. S. (Ed.).

STANDISH_GROUP (2002) What are your requirements? (based on 2002 CHAOS REPORT). IN STANDISH GROUP, T. (Ed.) West Yarmouth,MA, The Standish Group International

STANDISH_GROUP (2006) The Chaos Report

STANDISH_GROUP (2003) CHAOS Reports.

STRAUSS, A. L. C., J. (1998) *Basics of Qualitative research techniques and Procedures for Developing Grounded Theory,,* USA, sage.

TELEDYNEBROWNENGINEERING (Ed.) (2007) *XTie-RT Product Description*.

TELELOGIC (2007) DOORS Product Description.

TELLIS, W. (1997) Introduction to case study. *The Qualitative Report,* 3**,** 1-11.

TNI-SOFTWARE (2007) Rectify Product Overview.

TROCHIM, W. & LAND, D. (1982) Designing designs for research. *The Researcher,* 1**,** 1-6.

VAN WELIE, M. & TRÆTTEBERG, H. (2000) Interaction Patterns in User Interfaces. *PLoP 2000 conference*.

VANHOOFF, B. & BERBERS, Y. (2005) Supporting Modular Transformation Units with Precise Transformation Traceability Metadata. *Proc. Traceability Workshop, European Conference in Model Driven Architecture (EC-MDA)*.

VITECH (2007) Core Product Suite.

WALDERHAUG, S., JOHANSEN, U., STAV, E. & AAGEDAL, J. (2006) Towards a Generic Solution for Traceability in MDD. *ECMDA Traceability Workshop.* Bilbao, Spain, OMG.

WALKER, A. (1997) ModelMaker-User Manual Version 3. Cherwell Scientific Publishing Limited, Oxford.

WEBSITE, I. (2001) Internet homepage of the International Council on Systems Engineering at http://www. incose. org. INCOSE.

YIN, R. (1989) *Case study research—design and methods.,* Newbury Park (CA), Sage Publications.

YIN, R. K. (1993) *Applications of Case Study Research. , ,* Newbury Park,, CA: Sage.

YIN, R. K. (1994) *Case Study Research: Design and Methods.* , Thousand Oaks CA: Sage.

ZAVE, P. (1983) Operational specification languages. *Proceedings of the 1983 annual conference on Computers: Extending the human resource***,** 214-222.

ZHANG, Y. M. J. C. S. (2004) A process component model for enterprise business knowledge reuse. *IEEE International Conference on Services Computing.* IEEE.

ZISMAN, A., SPANOUDAKIS, G. & CYSNEIROS, G. (2003a) A Traceability Approach for i* and UML Models. *Proceedings of 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems-ICSE*.

ZISMAN, A., SPANOUDAKIS, G., PEREZ-MINANA, E. & KRAUSE, P. (2002) Towards a Traceability Approach for Product Families Requirements. *Proceedings of 3rd ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications, Orlando, USA, May.* In conjunction with the 17th IEEE International Conference on Automated Software Engineering, Edinburgh, U.K, IEEE Computer Society Press.

ZISMAN, A., SPANOUDAKIS, G., PEREZ-MIÑANA, E. & KRAUSE, P. (2003b) Tracing Software Requirements Artefacts. *Proceedings of the 2003 International Conference on Software Engineering Research and Practice, SERP,* 3.

# APPENDIX I TRACEABILITY TOOLS SURVEY

In this appendix we review some of the traceability tools that are available to industrial and academic users. A traceability tool is a repository for holding traceability items, their attributes and their dependencies. The tools typically track the following information per traceability item; traceability item tag (e.g. <MRS> which is the tag identifier for each traceability item that is contained in the Main Requirement Specification), traceability item type (e.g. functional, usability), name, description, owner of item, the item status, issues and options, the decisions and reference to dependent items.

In general these tools have the functionality to configure and change the traceability items, to carry out impact analysis, to baseline items, to store attributes, to provide metrics, to record the status of items, provide filters, query or search functionality and provide a user friendly interface that displays the data to all parties. More sophisticated tools will integrate with the development tools in other development environment. [26] For example there is a link between the traceability item stored in your traceability tool and the design elements and test elements stored in other tools. You make a change to an item in any of these tools and this change is reflected across all tools that references it.

Integration between traceability items and other development tools was a major factor in Ericsson's in the early 2000s. One of the problems they encountered was how do the testers import their test cases from the test suite into the traceability tool. Sychronisation between the traceability items was one of the most important factors that Ericsson's use in their tool selection criteria.

Other "generic" features of the traceability software include a *centralized repository for storage of traceability items, multi-site* (different geographical locations can use same set of requirements)*, cross project functionality, adaptability* for development processes, *scalability* for large or small product or project development, multi-project compliance with *end-to-end*[27] product lifecycle traceability, end-to-end *impact analysis*, that can work on different platforms (Vista and UNIX), with *diverse, configurable*[28] *or intuitive client interfaces* (for example, Eclipse, Microsoft, Visual Studio and Windows) and *integration* with other development tools in the vendors tool-chain.[29]

To get a perspective of the best tools available commercially, in the open source community and academically was to take a selection of tools and assess them. Of course



[26] The Rational Enterprise Suite is an example of an Integrated Development Environment (IDE) which has analysis, design, implementation and test tools integrated in the same environment.
[27] End-to-end refers to the entire Product Development Lifecycle (PDLC)
[28] Ability to configure the interfaces is very beneficial to enable organisations to use suitable or similar interfaces to in-house tools
[29] In the Grand Challenges technical report they describe that a traceability tool should have the capabilities for link construction or generation, link assessment, link maintenance, and link usage; however, there is no single tool that can cover all of these tasks.

commercial tools are expensive but in many cases they offered free services to universities for research. For example, we received both IBM's Rational Requisite Pro and Telelogic Door's for free from their university programs.

| Tool Name | Vendor | Main Features | Tool Description | Tool Assessment |
|---|---|---|---|---|
| Caliber-RM | Borland | Centralized repository, adaptability, end-to-end requirements traceability, end-to-end impact analysis, diverse client set, and integration with other Borland products. | Borland CaliberRM is an enterprise software requirements management tool that facilitates collaboration, impact analysis and communication, enabling software teams to deliver on key project milestones with greater accuracy and predictability | Positive Aspects:<br>▪Easy to use<br>▪Easy to customise<br>▪Good for Requirement Versioning & Baselining<br>▪Good Audit Trail<br>▪Good Change Control<br>▪Integrates with testing tools<br>Negative Aspects:<br>▪Poor reporting<br>▪Poor metrics<br>▪Slow execution<br>Note: Only assessed demo version and played with tool in local software company |
| Caliber Define IT | Borland | Collaborative business scenario, industry-unique storyboarding, detailed analysis and specification, seamless requirements integration. | Borland Caliber Define IT is a tool for defining software requirements providing support across four key requirements definition process areas: elicitation, analysis, specification and validation. | Excellent for requirement elicitation, analysis and communication. Poor traceability management functionality. For example, only allows trace-to, trace-from relationships.<br>Note: Only saw a demo of this tool and assessed literature |
| CORE (Vitech, 2007) | Vitech | End-to-End System Traceability, Risk Management, On-Demand Document Delivery, Integrated Development Lifecycle Support | The CORE tool is more a systems engineering tool than a traceability tool. It has good functionality for behavioural modelling and data flow diagrams. Its primary functionality in the traceability domain is its ability for displaying relationships between work products. | Core 5.0 University Edition assessed. |

| | | | | |
|---|---|---|---|---|
| DOORS[30] (Telelogic, 2007) | Telelogic | Adaptable interfaces, scalable for all size products or projects, easy to use traceability matrixes, good impact analysis, integrated with other Telelogic tools in development lifecycle. It is a multi-platform tool. | Telelogic DOORS, promotes improving product quality by improving communication in requirements, better visibility of business objectives, customer needs, technical specifications with powerful capabilities for impact analysis, conformance and compliance to regulations and standards. | Full academic license provided by Telelogic South Africa |
| Requisite Pro(Rational, 2007)[31] | Rational/ IBM | Window based, interfaces with Microsoft Word documentation, easy creation of traceability items, good attribute creation, suitable for multi-site, possible web interface, flexible traceability matrixes, integrates with other Rational development tool and is based on a sound supporting database. | This tool has an easy to use interface and traceability matrixes. The creation of the traceability items and their attributes is user friendly. The database central repository is transparent to the user, while the web interface is easy to understand. | Full academic license and industrial license assessment. |
| MARS | Ericsson Products Methods & Tools (PM&T) | Central repository in Sweden, Web interface, built on Matrix One Platform, suitable for multi-site and multi-project, easy creation of traceability items, good traceability structures, good traceability matrixes, easy statistics, easy impact analysis. | Owned by Ericsson MARS is an approved Ericsson tool that is owned and controlled by the R&D PM&T organization at Ericsson. It is used by all projects in OSS-RC. | Full assessment |



[30] DOORS has four separate tools available DOORS, FASTRAK, Analyst, Net, Dashboard. In this study we assessed their main tool DOORS.

[31] The Rational corporation were bought by IBM in 2003, but they have kept the Rational name is some of their products.

| | | | | |
|---|---|---|---|---|
| Rectify (Tni-Software, 2007) | Tni-Software | You can highlight traceability items from any source (Word, PDF etc) It is possible to select the DOORS baseline you want to analyze. "Reviewer" plug-in to automate your requirements and project document reviews. Extended capabilities for management of requirement changes and comments. Improvements of the graphical view and requirement details view. | This is a unique tool in that it is a "gateway" tool between the requirement authoring in any context which helps with the deployment of your requirements management process<br><br>Reqtify has requirement capture functionality from any type of source (Word or PDF documents, Excel spreadsheets, DOORS modules, in-house developped tools, bug tracking systems) with extra functionality for impact analysis. | Full assessment |
| XTie-RT (Requirements Tracer) (TeledyneBrownEngineering, 2007) | Teledyne Brown Engineering provides [32] | Windows based, multi-user, muti-site, large complex projects, product development view-point. | XTie-RT is applicable in all aspects of systems engineering and project management, ensures product quality, enforcing the traceability of requirements throughout a product development life-cycle. XTie-RT is robust enough to use on large, complex projects. Windows based. XTie-RT is available as a multi-user system with server and client software or as a stand alone system. | Download demonstration and assessment. |



[32] The Technologies Group provides technology-based support to the U.S. Department of Defence, manufacturing solutions to space, military, environmental, and missile defense requirements. Teledyne Brown Engineering is ISO 9001 Compliant.

| | | | | |
|---|---|---|---|---|
| [33]Wonder (Hazeline et al., 2007) | Invensys | Distributed Centralization approach to artifact storage using MS SQL database. Tracing between distributed artifacts, data exchange between different tools. | The tool was intended to achieve three main goals: 1) minimize overhead in trace definition and maintenance; 2) preserve document integrity; 3) support SDLC activities. [34] | |



[33] This tool was presented at the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering. We include this tool in our survey due to its newness to the traceability tooling world.

[34] Requirements traceability is closely tied to process traceability at Wonderware.

# APPENDIX II QUESTIONNAIRE

*Please complete the answers on the lines*

1. Name _____

2. Role _____

3. Education (discipline, degree awarded)
_____

4. Did you receive requirement engineering and traceability training during your education? Yes___ No ___ Don't remember___

5. Corporation Name_____

6. Size of Company _____

7. Business Type (Financial, Telecoms)
_____

*Let us examine the processes you have in place*

8. Does your organisation have a development process? Yes___ No ___

*9.* Does your organisation have a process team? Yes___ No ___

*10.* Does your organisation keep a process model Yes___ No ___

*11.* In your opinion does the process describe requirement engineering sufficiently?
Yes___ No___

*12.* In your opinion does the process describe traceability? Yes__ No___

*13.* In your opinion does the process describe change management and control? Yes___ No___

*14.* Does the process describe roles and responsibilities Yes___ No ___

*15.* Does the process describe the traceability responsibilities? Yes No

*16.* Does the process describe the artifacts produced in the development process Yes___ No ___

*17.* Do you receive training on the process (formal/informal) Yes___ No__

*18.* Do you understand the terminology used throughout the process Yes___ No ___

*Let us examine the tool situation in your organisation*

19. Does you organisation have a specific tool for requirement management or traceability tool Yes___ No___

20. Does your organisation have an informal tool or approach for handling requirements and traceability (spreadsheets, matrixes) Yes___ No___

*Let us examine your attitudes and problems with traceability and related concepts*

21. Traceability is an important practice to successful software development? Yes___ No_____

22. In your opinion can you please number the factors that influence traceability in your organisation

*Lack of Time* _____

*Lack of training* _____

*Lack of budget* _____

*Poor processes* _____

*Insufficient tools* _____

*Poor overall processes* _____

*Lack of resources* _____

*Lack of mgt commitment* _____

*Not mandated by management* _____

*Poor supporting documentation* _____

*Poor requirement engineering practices* _____

# APPENDIX III TESFE 2005 PAPER

(Proceedings of the 3rd international workshop on Traceability in Emerging Forms of Software Engineering)

# A Reusable Traceability Framework using Patterns

Justin Kelleher
Department of Computer Science
University of Cape Town
Cape Town, South Africa
+27216505108

jkr@cs.uct.ac.za

## ABSTRACT

To accomplish reusable traceability practices a common framework must be established. In this paper we describe a traceability framework which consists of a **TRA**ceability **M**etamodel (TRAM) and a **TRA**ceability **P**rocess (TRAP). TRAM provides a language for describing the elements of a traceability process. The **TRA**ceability **P**rocess (TRAP) is a process authoring tool for publishing product lifecycle process configurations as a web site for practitioners to access. A key component of the traceability process is the introduction of *traceability patterns* which provide a standardized mechanism for the visualization and communication of reusable traceability practices.

A tool environment supporting the traceability framework is described. The **TRA**ceability **P**attern **E**nvironment (**TRAPE**d) is an environment for the structured and collaborative design of a traceability metamodel, process and patterns. Finally we represent the traceability metamodel, process and patterns using Topic Maps (ISO 13250)

**Keywords**

Traceability metamodel, traceability patterns, topic maps, ISO 13250, **Tra**ceability **P**rocess **T**ool (TraPT)

## PATTERNS IN RELATED LITERATURE

In the 1990s the software engineering world focused on the traceability domain describing the problems encountered with traceability [6, 8], the factors influencing requirement traceability [17] revealing new techniques [5], describing traceability items [19], new traceability environments [4] and traceability reference models [16]

However, little research focused on the reusability of engineering activities or the recognition of commonalities in practices within the traceability domain. In 2002 Patricio Letelier described "A Framework for Requirements Traceability in UML-based Projects" [9] We build on Letlier's traceability reference model concept by designing a traceability metamodel (TRAM) which provides a language for describing any traceability process in any domain. We used the Software Process Engineering Metamodel (SPEM) specification as an extension to the UML for software process engineering. SPEM is defined by the Object Management Group [12]

This paper is structured as follows. Chapter 2 describes the **TRA**eability **M**etamodel (TRAM), Chapter 3 describes the **TRA**eability **P**rocess (TRAP) Chapter 4 introduces conceptual traceability patterns and motivating tool support. Chapter 5 describes the representation of the metamodel, the process and the patterns using topic maps**.** Chapter 6 describes the architecture for the **TRA**ceability **P**attern **E**nvironment (**TRAPE**d) In the final chapter we review our work and discuss future research.

## TRAEABILITY METAMODEL (TRAM)

As a compromise between the requirements for a standard notation and for domain-specific modeling, the UML was designed as an extensible modeling language. We tailor UML and define traceability domain specific model elements. The Object Management Group (OMG) defines a four-layered architecture which provides a solid basis for defining a traceability framework. Figure 1 illustrates the four layered Model Driven Architecture based on the Object Management Group (OMG) standards; the Meta-Object Facility

(MOF) [10], the Unified Modeling Language (UML) and the XML Metadata Interchange (XMI) [11].

Figure 2 describes the package structure for the traceability metamodel. The MOF is an abstract language and framework for specifying, constructing and managing technology neutral metamodels. It is the foundation for defining any modeling language.

At Level M2 we define the **TRA**ceability **M**etamodel (TRAM), which is an instance of the MOF. The traceability metamodel is a precise definition of the constructs and rules needed for creating semantic model elements for the TRAeability Process (TRAP).

The TRAM_Foundation::Core package is similarly structured to the UML 2.0 Core Packages. The Extension::BasicElements package, defines the basic elements called ExternalDescriptions and Guidance. The Guidance meta-class becomes the traceability best practices and traceability techniques at the M1 (TRAP) level.

The TRAM_Extensions::Dependencies allows a Categorization dependency from a package to an individual process element in another package. It also allows an impact dependency from one WorkProduct to another WorkProduct to indicate that the modification of a WorkProduct could invalidate another.

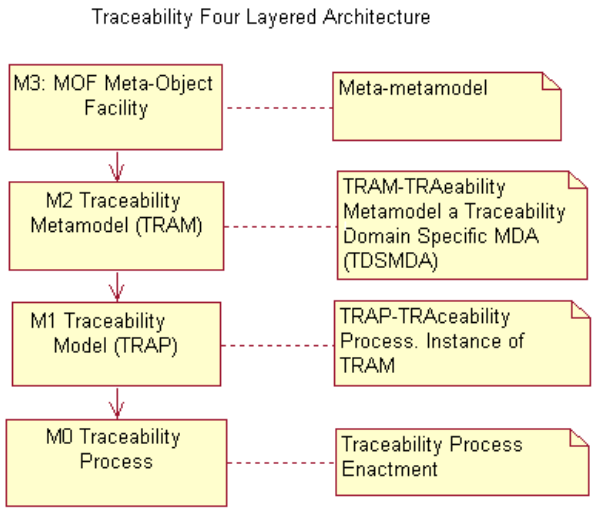



**Figure 0-1: The Traceability Process Architecture.**

Process structure is important and the Extension::ProcessStructure is composed of the elements: WorkProduct, WorkProductKind, WorkDefinition, Activity, Step, ProcessPerformer, ProcessRole. The traceability relationship between work products is essential therefore we define a work product or artifact as anything produced, consumed, or modified by the process. A WorkProductKind describes a category of work product, such as Text Document. WorkDefinition is a kind of operation that describes the work performed in the process. Activity is the main subclass of WorkDefinition. It describes a piece of work performed by one ProcessRole: the tasks, operations, and actions that are performed by a role or with which the role may assist. An Activity may consist of atomic elements called Steps. A ProcessPerformer defines a performer for a set of WorkDefinitions in a process. ProcessRole defines responsibilities over specific WorkProducts, and defines the roles that perform and assist in specific activities.

TRAP_Extension::Process Lifecycle consists of phase, lifecycle and the self explanatory iteration

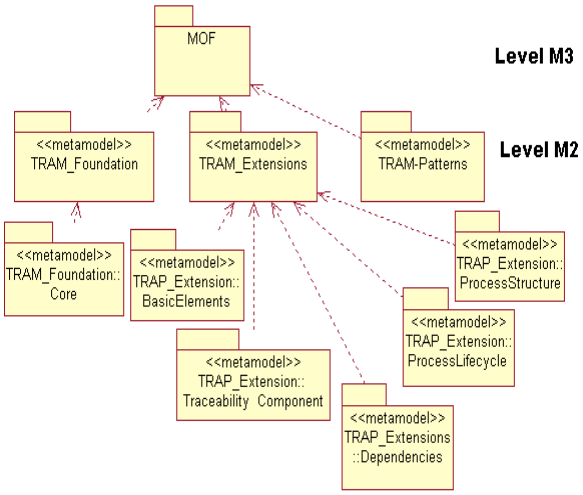



**Figure 0-2: TRAM Package Structure**

## TRACEABILITY PROCESS (TRAP)

The traceability metamodel is instantiated to become a traceability process, which can be configured for any product lifecycle scenario. TRAP is a synthesis of traceability best practices collected and developed at the University of Cape Town during a research program. It packages and incorporates the best available traceability techniques for the product lifecycle into a unified process. We divide TRAP into 4 levels: Business Unit Level, Network Level, Node Level and Subproject Level. It is structured as a sequence of high level workflows. Each workflow is broken into discrete steps supported by descriptions of traceability activities, roles and artifacts associated with each step. As the workflows and steps are organized in chronological sequence within the development life cycle, the process should ideally be read in sequence. A core aspect of any process specializing in traceability is the relationship between the work products. Therefore the backbone to TRAP is the definition of 150 traceability items and their corresponding traceability matrices.



TRAP is a process authoring tool which can be configured for any project. Once adapted the production process becomes a level M0. Figure 3 below shows an activity chart for a requirement engineer at node level.

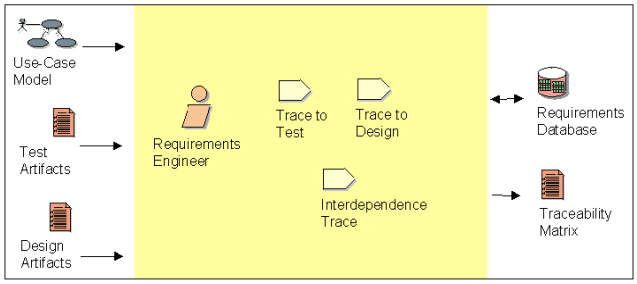



**Figure 0-3: The TRAeability Process-TRAP**

## TRACEABILITY PATTERN

In this section we describe two types of conceptual traceability patterns. TRAM is a *reusable architecture* that provides the generic structure and behavior for a family of traceability modeling elements, along with a context of metaphors that specifies their collaboration and use within the traceability domain. Traceability patterns can be employed both in the design and the documentation of the TRAM metamodel. We have identified several traceability patterns during the design of TRAM. In fact, TRAM can be viewed as the implementation of a system of traceability patterns.

It is important to recognize that meta-modeling patterns and traceability engineering patterns are different: The TRAM patterns are described using UML and hence *are executable software*, whereas traceability engineering patterns represent knowledge and experience about traceability. In this respect, the TRAM metamodel is of a physical nature, while traceability patterns are of a logical nature. TRAM is the *physical realization* of one or more metamodel traceability pattern solutions; while engineering patterns are the instructions for *how to implement* those traceability solutions. For example the TRAM_Pattern package contains the model elements for creating traceability patterns. These metamodel patterns can then be transformed to produce model pattern elements. They describe the meta-classes and meta-associations for creating traceability patterns.

Traceability patterns describe best practices; good traceability designs and captures successful work experiences [2]. In other words traceability patterns create a "shared language for communicating experience and insight" [18]. Each pattern explicitly represents experience and knowledge. Because the patterns are named, individuals can use those names to easily refer to that experience. Traceability patterns are a compact way to reference a set of decisions [11].

Therefore we describe traceability patterns at two different levels of abstraction:

1. *Generative Traceability Patterns:* Generative patterns were used to create the TRAM metamodel. Basically, this is the description of a common vocabulary for a traceability metamodel to ensure traceability is used consistently in any problem domain. Figure 4 depicts a fundamental conceptual pattern using the UML notation for a class. Multiple roles interact or collaborate by exchanging work products and triggering the execution, or enactment, of certain traceability activities. ("trace-to" & "trace-from") The overall goal of the traceability process is to trace between a set of work products. From this Role Traceability Pattern, a first step consists of "reifying" role, activity, and work product.
2. *Traceability Engineering Patterns:* These patterns help exchange traceability experience or knowledge and provide rules for generating successful traceability practices. They offer solutions to typical situations in the implementation of traceability that are frequently encountered by the product team. The traceability patterns shall be written in an instructive and intuitive way and contain guidelines which have proven successful under several comparable circumstances.

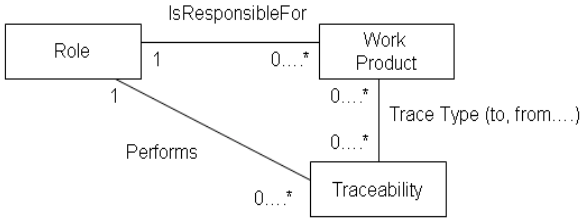



**Figure 0-4: Role Traceability Pattern**

Short examples of the traceability patterns include:

- Traceability Plan Pattern: The purpose of the Traceability Plan is to describe the activities for the product team to manage and trace the requirement work products, associated requirement types, and their respective requirement attributes.
- Traceability Strategy Pattern: This pattern describes how to set up a traceability structure. For example what is the traceability hierarchy (trace-relationship) between product requirements, model items, test requirements, design, user documentation and other traceability items?
- Product Compliance Pattern: This pattern describes how the system testers should trace from system test cases to customer requirements, to solve the product concurrence problems during acceptance test phase.

Coplien defined a pattern language as a structured collection of patterns that build on each other to transform needs and constraints into an architecture. We can now describe traceability in the product lifecycle by combining the traceability engineering patterns into a traceability pattern language [3]. The relationships between the participating patterns can be used to structure the pattern language. Another technique for conceptualizing product lifecycle traceability pattern proposes combing patterns to form hybrid patterns. Traceability hybrid patterns are identified by analyzing the key interactions between two traceability patterns. A hybrid pattern forms when one pattern is used to solve a sub-problem of another pattern [15]. We thus group or hybridize the patterns to solve traceability in the entire product lifecycle.

Traceability anti-patterns represent "lessons learned" while implementing traceability. We identified two types of traceability anti-patterns. Firstly, those patterns that describes a bad solution to a problem which resulted in a bad situation. Secondly, those patterns that describes how to get out of a bad situation and how to proceed from there to a good solution.

## TOPIC MAPS

The ISO standard ISO 13250:2003 is an international standard that can be used for the formal representation of the traceability metamodel, process and patterns as Topic Maps. These Topic Maps are a powerful, easy-to-use XTM based approach to requirements management that helps teams manage product requirements comprehensively, promotes communication and collaboration among team members, and reduces project risk [1]. Topic maps enable multiple, concurrent views of sets m*etadata* or in this case information related

to requirements and traceability. [7] The XML Topic Maps (XTM) 1.1 [13] specification defines an interchange syntax based on XML and XLink. We completed the following topic map research experiments:

- We created a traceability system with a primary navigation window using topic maps as the enabling technology. [14]
- We displayed and organized traceability items, their attributes, and their relationships with other requirements in nodes using a topic map views. We created a topic map view to display requirement attributes, such as status and priority, or to show the relationships between requirements.
- Each requirement or any traceability item was represented as a topic. We use the XTM <topic> syntax. The relationship between topics or traceability was represented by an association (<association>) e.g. “Requirement A traces to Requirement B”. Topic maps allows any kind of traceability relationship to be expressed
- The traceability items can be stored in many different locations (e.g.“http://reqpro.html is the Requisite Web location of Requirement A”) We used the XTM <occurrence> tag for this purpose.
- Topic map associations are inherently bidirectional, therefore automatic change management was possible. Robust traceability change management features visually indicated how changes affected the project, thereby giving us the ability to perform real-time impact analysis and allowing us to make informed decisions for scope management or resource allocation.
- Topic Maps include query functions for filtering and sorting the requirements and their attributes in the topic maps. We used TMQL (Topic Map Query Language) to interrogate the topic maps to extract relationships of interest. For example, we created an attribute count filter to determine how many requirements in the project have a priority with the value "High". We then combined one or more filters to produce a query. These queries provide traceability metrics concerning a project’s requirement attributes, relationships, and revisions. We retrieved information on the progress of a project with regard to priorities, workloads, and deadlines, the addition of new requirements, and changing or unstable requirements

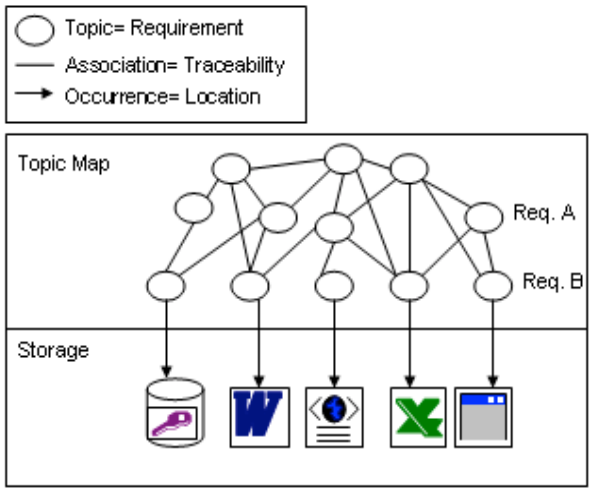



**Figure 0-5: A Topic Map Traceability System**

- We created round-trip engineering from UML to XMI to CSV to XTM and vice versa. We represented our traceability metamodel and traceability patterns in UML. We converted the

UML to XMI. Transforming the XMI into CSV we imported the CSV format into a requirement management tool (Rational RequisitePro) and reviewed the resulting traceability matrix. We finally converted the XMI to XTM visualizing the results topic maps.

- Topic maps can be merged, for example information from different sources (customer requirements versus project requirements) can be brought together into a single topic map and integrated into a meaningful whole. Merging can take place at the discretion of the user or application (at runtime), or may be indicated by the topic map's author at the time of its creation.

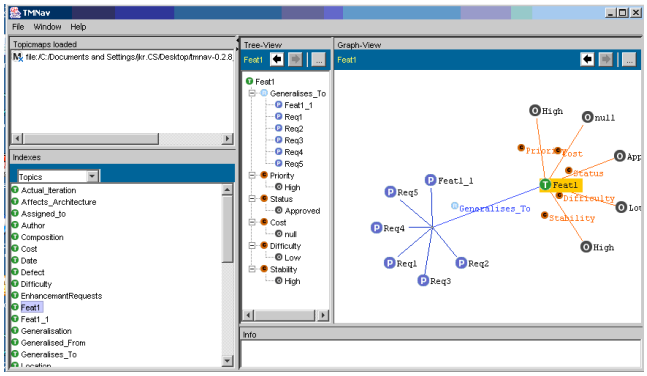



**Figure 6: Fig: Topic Map View**



## TRACEABILITY PATTERN ENVIRONMENT (TRAPeD)

We propose an integrated traceability environment framework. The Traceability Pattern Tool (TraPT) is an experimental platform integrating process, patterns and a traceability management system. The platform is the result of the integration of a number of different research projects and tools. Our proposed architecture is shown in Figure 7 comprises of six major parts:

- Module 1: The functionality of the Pattern Creator is split into pattern definition and pattern management and was designed using the MVC (Model-View-Control) pattern. Separate modules are designed with standard interfaces between them: (1) The model is used to store all the data associated with a pattern. (2) The view is used to allow the user to view the model in a variety of ways. (3) The control allows the user to manipulate the model. This is usually done using a view to give the user access. [21]

- Module 2: Catalogs and stores the patterns in a pattern encyclopedia. One of the problems encountered was that people do not understand the traceability patterns and cannot identify patterns to solve their problem. The Pattern Encyclopaedia (Fig 8) module is designed to ease the difficulties of potential pattern users. The Pattern Encyclopaedia provides literature in the form of summaries as well as academic papers and articles on topics related to patterns. The Pattern Encyclopedia module is designed to enable users to learn about patterns and traceability, a search helps identify patterns. The Pattern Encyclopaedia should be made accessible to an entire organisation.

- Module 3: Traceability Process Workbench has the TRAM process metamodel as the base from which you create your own process configurations. TRAP is the authoring and

publication tool that allows process configurations to be created for different needs and then published as a web site for practitioners to access.

- Module 4: The XTM editor for creating and editing topic maps.
- Module 5: This contains the XTM engine and Query Engine.
- Module 6: Topic map navigation window.

We used the TM4J which is an open-source framework for developing topic map processing applications. It provides a set of APIs that allow parsing, manipulating and exporting of topic maps. The standard topic map format used is XTM. TM4J is an open-source framework for developing topic map processing applications. It provides a set of APIs that allow parsing, manipulating and exporting of topic maps. The standard topic map format used is XTM. [20]

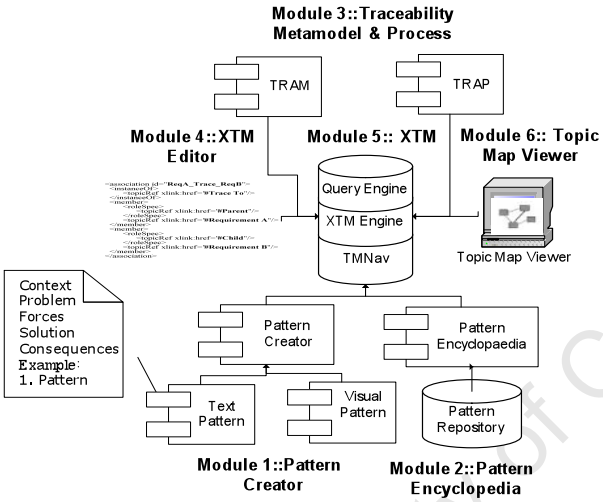



**Figure 7: Traceability Pattern (TraPT) Tool**

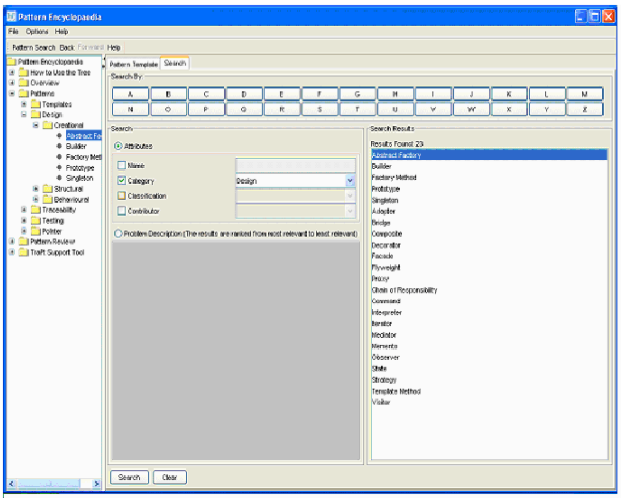


**Figure 8: Pattern Encyclopaedia**

## CONCLUSION AND FUTURE RESEARCH

In conclusion we describe a traceability process workbench comprising of traceability metamodel a language for defining traceability process models, the TRAP process authoring and publication tool, and the traceability patterns as a medium to communicate experience based traceability best practices defined in a uniform way. We explored using a traceability pattern language or traceability pattern hybridization to illustrate traceability in the product lifecycle. We described the use of the ISO 13250 standard to represent the traceability process, patterns and traceability items as topic maps. Based on our continuing experimentation we recommend topic maps as an excellent structuring language for representing traceability patterns and traceability items.

The University of Cape Town is also investigating the impact of the change to UML2.0 and the relationship between use cases and actors deleting the expression "an actor communicates with a use case" and replacing it with a new expression "an actor interacts with a subject" We are investigating the specification of the use-case class. For example, we are representing use cases as a class whose instances are scenarios. Our general approach begins by describing the use case in the TRAM metamodel. Each use case is an operation of the class *System*. All_Actors become another class. Each actor is an instance of All_Actors. A class is then created for each use case operation on System, each scenario is added as an operation of this class. An actor object starts a scenario by sending a use case message to the System, which in turn instantiates a use case object. The actor interacts with the new class objects normally. The actor can also send additional use cases messages (to System) or scenario messages (to UseCase objects). The rationale for instantiating use cases as classes is primarily to use the UML extension mechanisms of aggregations, associations, compositions, multiplicity and interfaces as traceability links. Also by instantiating use cases as classes is primarily for traceability links for testing source code. The use case classes serve as system and function test drivers.

## REFERENCES



[1] Ahmed., K. 2002. Topic Maps - A Practical Introduction with Case Studies. In Proceedings of XML Europe 2002. Barcelona, Spain, May 2002.

[2] Beck, K. Coplien, J. O., Crocker, R., Dominick, L., Meszaros, G., Paulisch, F., and Vlissides, J. (1996). Industrial Experience with Design Patterns, Proceedings of ICSE '96, Berlin, pages 103-114, March 1996.

[3] Coplien, J. O. (1997a). Idioms and Patterns as Architectural Literature, IEEE Software Special Issue on Objects, Patterns, and Architectures, 14(1), January 1997.

[4] Dömges R., Pohl. K., "Adapting Traceability Environments to Project-Specific Needs". Communications of ACM, Vol. 41, No 21, December 1998.

[5] Edwards M. and S. Howell. "A Methodology for System Requirements Specification and Traceability for Large Real-Time Complex Systems" Technical report, U.S. Naval Surface Warfare CenterDahlgren Division, Dahlgren, Va., 1991.

[6] Gotel, O. C. Z. and Finkelstein, A. C. W.: "An Analysis of the Requirements Traceability Problem," Proceedings of the First International Conference on Requirements Engineering, 1994, pp.94-101.

[7] Lacher, M.S., Decker, S. (2001). On the Integration of Topic Maps and RDF Data, from Proceedings of Extreme Markup Languages 2001. Montréal, Quebec, August, 2001.

[8] Leon, M. Intelligent Enterprise Magazine, July 17, 2000, Vol 3 Number 12

[9] Letelier P., "A Framework for Requirements Traceability in UML-based Projects" In Proc. of 1st International Workshop on Traceability in Emerging Forms of Software Engineering. In conjunction with the 17th IEEE International Conference on Automated Software Engineering, Edinburgh, U.K., September 2002 – 2002

[10] OMG, (2002) "Meta Object Facility (MOF) Specification v1.4", OMG Available from http://www.omg.org/cgibin/ apps/doc/formal/02-04-03.pdf

[11] OMG (2002) "OMG XMI Specification, v1.2", Available from, http://www.omg.org/cgi-bin/doc?formal/2002-01-01

[12] OMG, (2005) "OMG SPEM Specification v1.1" OMG Available from http://www.omg.org/

[13] Pepper, S. and Moore, G. 2001. XML Topic Maps (XTM) 1.0. TopicMaps.Org Specification. Aug. TopicMaps.org Consortium, Web site: www.topicmaps.org.

[14] Power, R. 2003. Topic Maps for Context Management, from proceedings of the 1st international symposium on Information and communication technologies. Vol. 49, Dublin, Ireland, July, 2003, pp 199-204.

[15] Ram D. J., Reddy P. J. K. and Rajasree M. S., Pattern Hybridization: Breeding New Designs Out of Pattern Interactions, ACM Software Engineering Notes, vol 29, No. 4, May 2004.

[16] Ramesh, B. & Edwards, M. (1993). Issues in the Development of a Requirements Traceability Model, Proceedings of the IEEE International Symposium on Requirements Engineering, San Diego, California, Jan. 4-6, pp. 256-259.

[17] Ramesh B., Factors Influencing Requirements Traceability Practice Communications of the ACM, 41(12), Dec. 1998. Pages: 37 – 44, ISSN:0001-0782

[18] Schauer R. and Keller. R. K. "Pattern Visualisation for Software" Comprehension, IEEE Program. Proceedings. 9th International Workshop on Comprehension, 2001. IWPC 2001 12-13 May 2001 Page(s):7 – 17

[19] Spence I., Probasco L., "Traceability Strategies for Managing Requirements with Use Cases" Rational Software White Paper, 2000

[20] Seedorf, S., Korthaus, A., and Aleksy, M. 2005. Creating a topic map query tool for mobile devices using J2ME and XML. In Proceedings of the 4th international Symposium on information and Communication Technologies (Cape Town, South Africa, January 03 - 06, 2005). ACM International Conference Proceeding Series, vol. 92. Trinity College Dublin, 111-116.

[21] G. Florijn, M. Meijers, and P. van Winsen. Tool support for object-oriented patterns. In Proceedings of the 11th European conference on Object Oriented programming, Springer LNCS 1241, pp. 472-495, 1997]

# APPENDIX IV ECMDA 2006 PAPER

(Proceedings from 4th ECMDA Traceability Workshop)

# Traceability Patterns

Justin Kelleher

University of Cape Town, Computer Science Department, Rondenbosch,
7701 Cape Town, South Africa. jkr@cs.uct.ac.za

**Abstract.** In this paper we describe two closely related subjects 1) Traceability Profile 2) Traceability Patterns. We introduce a profile that allows traceability to be represented in the UML. The patterns are a novel intuitive component that emerges from traceability activities. We describe the different types of patterns and provide a template for their creation. We provide examples of patterns that emerged during a recent survey. The purpose of our approach is to improve the understandability, reusability and communication of traceability practices.

**Keywords:** Traceability, profile, traceability patterns



## 1 Introduction

In this section we introduce the terminology used in the context of this study. Traceability is an important means to facilitate communication among the success-critical stakeholders, to ease determining the impact of changes and support their integration, to preserve knowledge and dependencies created during the design process, to assure quality, and to prevent misunderstandings [5]. Traceability is a technique used to "provide a relationship between the requirements, the design, and the final implementation of the system". [4] Traceability relationships help stakeholders understand the many associations and dependencies that exist between software artifacts created during a software development project. [11] A traceability link is a relationship between traceability items. The links are cross-references between the items connecting for example a requirement to a design element or to a source code element. Different types of traceability links can exist, traceTo, modifies, responsibleOf, validatedBy, VerifiedBy. [8]

A traceability item is defined as “Any textual, or model item, which needs to be explicitly traced from another textual, or model item, in order to keep track of the dependencies between them”. [12] A general definition of a traceability item is a “project artifact” or work product. Traceability items are usually structured in a traceability suite by a predefined traceability structure document. For example, a typical traceability structure is from high level requirements, to use cases, to design elements, implementation components and test artifacts.

Patterns describe best practices, good designs and capture successful work experiences [3]. Christopher Alexander, offered an instructive definition of patterns: “Each pattern describes a problem that occurs over and over in our environment and then describes the core of the

solution to that problem in such a way that you can use this solution a millions times over without ever doing it the same way twice [1]". In other words patterns are creating a "shared language for communicating experience and insight" Because the patterns are named; individuals can use those names to easily refer to that experience. Patterns are a compact way to reference a set of decisions and designs while suppressing the "details not relevant at a given level of abstraction" [4]. They have been used in several disciplines for capturing engineering knowledge and for providing guidelines for generating successful engineering solutions. We suggest the use of patterns as a general method for capturing and exchanging proven successful traceability practices. They generalize comparable observations of successful traceability activities from different projects into a structured format.

This paper is structured as follows. In Chapter 2 we describe a UML profile which establishes the context for introducing traceability patterns. In Chapter 3 we introduce our research method where the emergent patterns were discovered and we describe a template for describing these patterns. The different types of traceability patterns are discussed in Chapter 4 which we support with examples and observations that we identified during our survey. We conclude with a discussion of related literature and further ongoing research in this project.



**UML Profile**

UML provides an extension mechanism, the UML profile, to tailor the language to specific application areas. A profile is described in UML 2.0 as a stereotyped package that contains model elements that have been customized for a specific domain or purpose using extension mechanisms, such as stereotypes, tagged definitions and constraints. In principle, profiles merely refine the standard semantics of UML by adding further constraints and interpretations that capture domain-specific semantics. They represent an agreement within a community from which practitioners can then draw the particular model of any educational situation they want to describe.

In Figure 1 we illustrate a traceability profile. A collection of traceability patterns can be the starting point for creating a traceability profile. As a first step to creating a profile the impact of certain traceability rules are elicited. This can be done by collecting patterns that are observed or collected from previous experiences. Patterns can be utilized to describe each element defined in Figure 1.

A traceability item is any specification that can impact the system to be developed, for example a model, a diagram, a use case, a non-functional requirement, a change request, a test specification, or any other specification in the development cycle that impacts the overall system. They should be created using a pre-defined template. Item creation is a recurring activity and patterns can be used to capture each item.

Configuration management is the discipline of identifying the components of an evolving system for the purpose of controlling changes to these components. Configuration management controls the configuration of a product from product definition, development, build and maintenance. It is essential to integrate configuration management with traceability management for better product concurrence. A *ChangeSpecification* is a traceability item specifying a proposed change to another traceability item which is also under configuration control. Configuration control concerns the activity of controlling changes after a *Baseline* has been established. A Baseline is a version of a configuration established at a point in time when only controlled changes are allowed. A *Baseline*

*Specification* documents the traceability items in a configuration version. The Baseline Specification documents which item belongs to which baseline. When a Change Request is received *Impact Analysis* investigation is undertaken.

Traceability management benefits all project stakeholders, end users, project managers, developers, and testers by ensuring that they are continually kept apprised of the traceability item status and understand the impact of changing requirements specifically, to schedules, functionality, and costs. Traceability items can therefore be located in project plans, design documents, source code, correction patches, trouble reports, in emails or agreed during phone calls with stakeholders meetings. The element otherSpecification is used to capture all the other items that can have an impact on the system being developed.

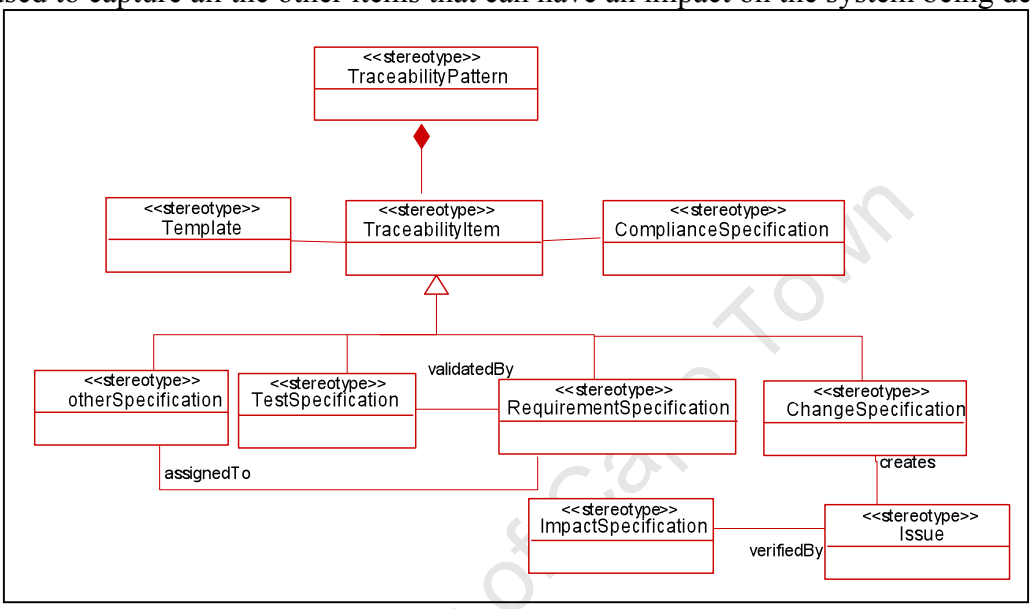



**Figure 0-1: Traceability Profile**



## Traceability Patterns

It is not uncommon for practitioners to have little or no experience in traceability. Traceability patterns aim to make traceability practices understandable to project team members in the development cycle. The following properties give a brief introduction to the Traceability Pattern approach:

- Each pattern has a template that can be applied in any traceability situation.
- They provide an abstract description of a traceability problem in a context defining a successful solution on how to solve it.
- Traceability patterns have a significant human component appealing to the aesthetic communication of core concepts in a comprehensive manner. For example, the patterns are a compact way to portray a vocabulary of traceability terminology.
- They capture experience and insight which can be reused in subsequent projects. Because the patterns are named, individuals can use those names to easily refer to that experience.

- Traceability patterns don't just describe experiences and insights, but they describe deeper traceability structures and methods for implementing traceability. For example traceability patterns can be used to describe hierarchical structures.

- A traceability pattern is a set of traceability types that can be instantiated to create traceability item object models. A pattern for a set of object models is created by identifying and defining the common types among these objects models.

- The overall traceability strategy can be described by a pattern language. For example, if we combine all patterns together we have the traceability strategy that can be easily understood, communicated and reused for later projects. The traditional requirement management plan could be replaced by a catalogue of patterns.

- Traceability rules that govern a traceability concept can be represented with a pattern with constraints encapsulated in it. For example a pattern can describe a set of constraints of how traceability items and relationships can be connected. Therefore we must provide the mechanism for constraint inheritance. A traceability pattern at the meta-level refers to a set of rules on how to create patterns. Patterns at the meta-level are of a physical nature, while traceability instance patterns are of a logical nature. Instance patterns are the instructions for how to solve traceability problems.

**Motivation for Traceability Patterns**

Patterns are usually based on experience and discovery rather than on invention. In Figure 2 we illustrate the research method that supported the detection of the traceability patterns. We began exploratory investigations with a Telecommunications Company in 2002. Our initial goal was to get a better understanding of traceability in the telecommunications application domain, the underlying technologies, the organization structure, the product structure, the project structure, the tools being used, the traceability terminology, the traceability items, the traceability process, the practices and the roles involved in the traceability domain. On analysis of the data captured we recognized that patterns or trends of traceability practices emerged.

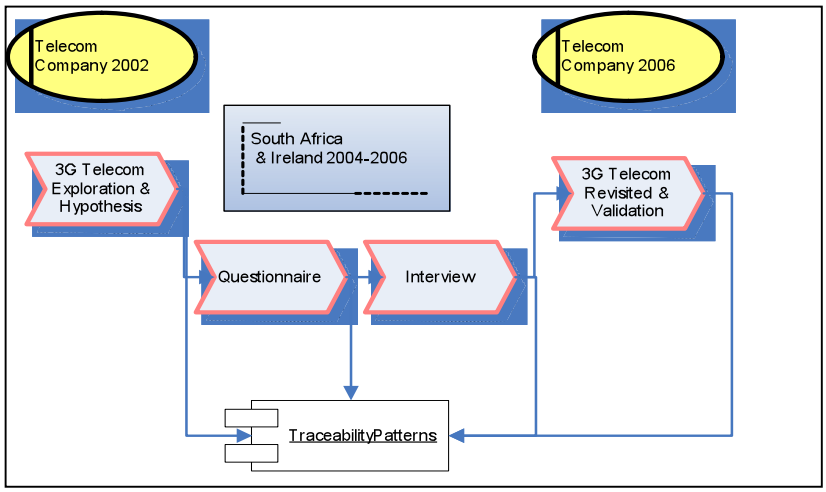



**Figure 0-2: Research Method Identifies Patterns**



## Pattern Template

The UML 2.0 infrastructure specification defines behavior as an observable effect of an operation or event, including its results. A behavioral feature is a dynamic feature of a model element, such as an operation or method. In UML 2.0, the BehavioralFeatures subpackage of the Abstractions package specifies the basic classes for modeling dynamic features of model elements. Behavioral patterns are concerned with the assignment of responsibilities between objects, or, encapsulating behavior in an object and delegating requests to it.

Each pattern must have a meaningful name, which is easily explainable and understandable. This helps with pattern classification or categorization as well as improving communication between different roles. The problem describes the traceability problem that the pattern aims to solve and the conditions that must be met before it makes sense to apply the pattern. The context describes the preconditions under which the problem and its solution recur and the patterns applicability. A description of the consequences might include an acknowledgment of the trade-offs involved in selecting a particular traceability pattern. They are used to evaluate the patterns. The implementation describes the implementation of the pattern in a traceability environment, for example how to set up requirement types and document types in Rational RequisitePro. In general, the implementation section is considered a non-normative suggestion, not an immutable rule or requirement. A constraint is a semantic condition or restriction. The rules thus specify constraints over attributes and associations which are defined in the metamodel. Most invariant is defined by an Object Constraint Language (OCL) expression together with an informal explanation of the expression, but in some cases the invariant is natural language. Patterns are defined using pattern templates called forms. A pattern is defined by specifying the values of the form fields for that particular pattern. In Figure 3 we illustrate a template for creating patterns.

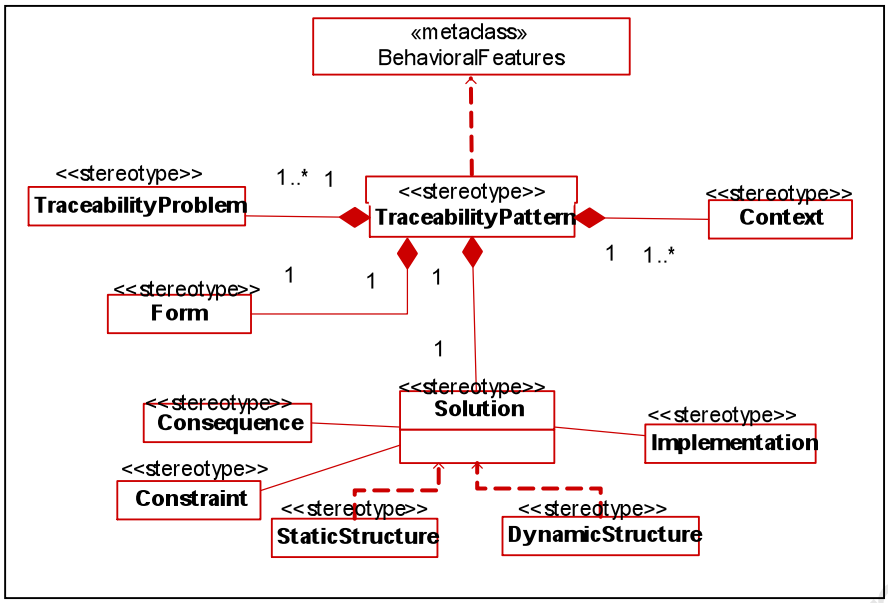



**Figure 0-3: Traceability Pattern Model**



## Types of Patterns

### Generative (or Emergent Patterns)

Alexander described this type of patterns as having force or having generative characteristics**.** Basically generative patterns are traceability actions that emerge from a problem and guide a user to generate a solution. The key word is that these patterns emerge from practical experience or from observations while implementing traceability. In Figure 4, we illustrate a profile for generative patterns.

There are a number of elements involved in the emergence of generative pattern. Traceability items are located in word documents, spreadsheets, URLs or traceability tools. We call this workspace the *TraceabilityMatrix.* A *Role* is either a physical project resource or tool based resource which performs a traceability *Activity*. For example a Role is a project manager, tester, maintenance engineer or a traceability environment. Generative patterns are derived from two or more similar observations being made in the *TraceabilityMatrix*. For example, a project manager implements a <<trace_to>> dependency relationship for each customer requirement to a use case. A generative pattern is identified on analyses of the *TraceabilityMatrix.*

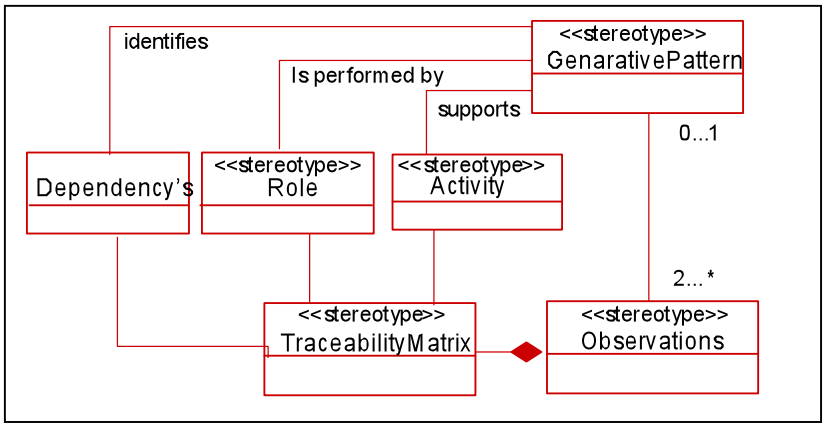



**Figure 0-4: Generative Traceability Patterns**

## Conceptual patterns:

Conceptual patterns describe and explain the key concepts and traceability principles. For example they describe the terminology and principles for traceability items, types, activities, relationships, relationship types and hierarchies. They can be used to serve as a template for creating a UML Profile, to describe a common vocabulary, to capture traceability best practices or to define new principles from white papers. While patterns are dynamic, conceptual patterns have static characteristics than generative patterns usually capturing more stable principles.

In Figure 5 we illustrate a simple conceptual pattern from Gotel & Finkelsteins pivotal paper "An Analysis of the Requirements Traceability Problem" This paper introduces the distinction between pre-requirements specification (pre-RS) traceability and post-requirements specification (post-RS) traceability. [7] In this example we use both a static diagram and a dynamic diagram to visually represent the solution.



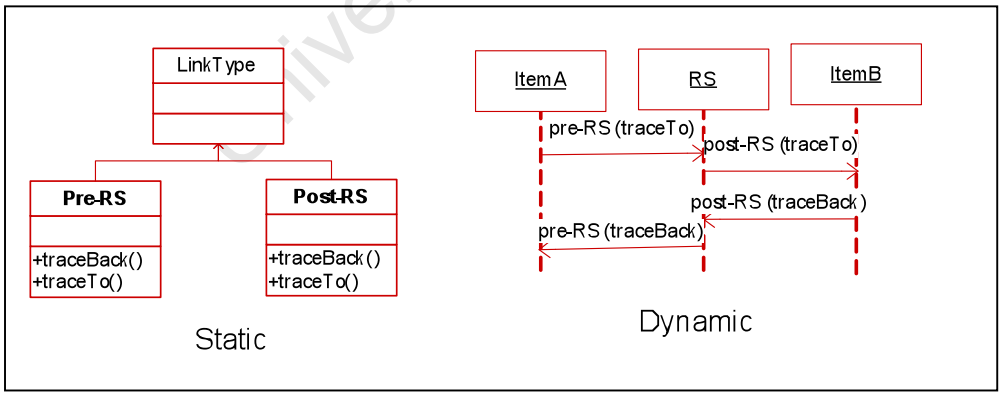



**Figure 0-5: Simple Traceability Pattern**

**Name:** Pre-Requirement Specification Traceability

**Problem:** A survey of 100 practitioners exposed that many of the problems attributed to poor requirement traceability were found to be due to the lack of pre-RS traceability.

**Consequence:**

- Perceived as an optional extra (and of low priority), so the allocation of time, staff, and resources is often insufficient. The documentation of required information is no guarantee of its traceability. That which is structured, so it is traceable in many ways, provides no guarantee it will be up to date. (for this example or one consequence is described)

- **Solution: (see Figure 3)**

- Pre-RS traceability refers to those aspects of a requirement's life prior to inclusion in the RS.
- Post-RS traceability refers to those aspects of a requirement's life that result from inclusion in the RS.

**Implementation:** In RequsitePro use the "trace to" from items created before the requirement specification and "trace from" for tracing back to the requirement specification.

In Figure 6 below we illustrate another conceptual pattern. Traceability items have a unique name, identity, a revision, a specific type and attributes such as author, person responsible, origin or rationale, release number, status, priority, cost, difficulty, stability, and risk. All items should be under version control with each revision creates a separate item. It is possible to go back to a previous revision of an item and check the relationships to other items for that specific revision. Items change state and go through a life cycle. Changing the state of an item is regulated by a change control board. For example a CR may move from "new" to under "investigation", to "approved" or "rejected" to a final "closed" state. Usually only an authorized role can, for instance, promote a preliminary document to status "APPROVED". Requirements management tools generally generate several system-defined attributes, such as date created and version number. A history log records all events that have occurred to a traceability item. The recorded attributes include the action, the user, the time and a brief description of the action performed. The log provides full transparency of actions that have taken place. History contains information about each activity. The history starts when an item is created and automatically records all activities associated with that item. By analyzing the history log files patterns of activities emerge.

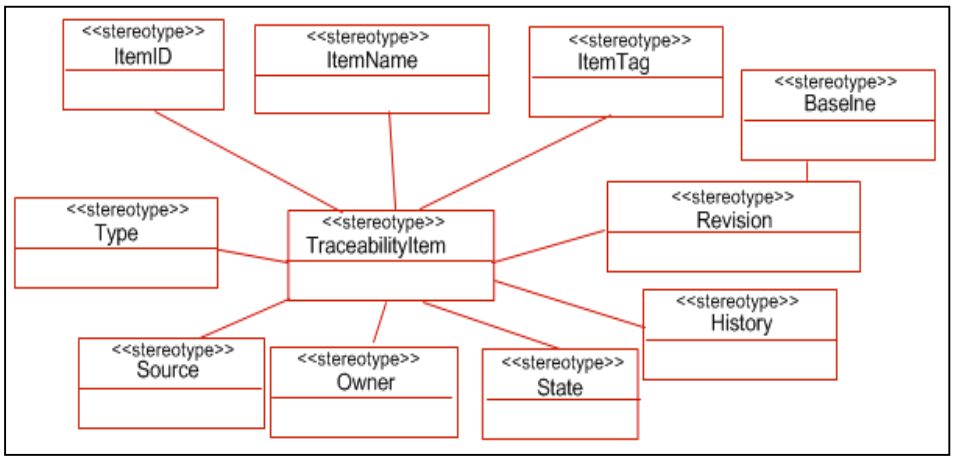



**Figure 0-6: Traceability Items**

**Observed Industrial Patterns**

During our survey a number of traceability patterns we observed.

**COTS Traceability**

*Problem:* In general COTs source code is not available to the development organization making it difficult for developers to trace from their user requirements to the functionality provided by the COTs software.

*Solution:* A pattern that explains consistency checking between each COT resource (classes, functions) and the supporting documentation. You cross check the user documentation with each resource. Each identified resource becomes a traceability item with a unique name, identifier, version and attribute values as defined in the pattern shown in Figure 6.

**Human Tracing**

*Problem:* A product can be developed over many years. A common problem recorded in our empirical study was human information was lost. The projects had lost track of who was involved in the creation of the traceability item, their contact information, department or current location. One assessment note showed that staff roles and responsibilities were poorly defined and this led to problems with the identification of the individuals involved.

*Solution:* Each traceability item must contain information on the creator or modifier including their role and responsibility.

**Faulty Data Sets**

*Problem:* The survey found that some traceability items did not have the documentation or know how needed to trace them back to their sources.

*Solution:* Use a pattern to enforce the rule that each item created must be traceable to at least one other item on creation. If this rule is applied traceability items that are not traced are not included in the baseline.

**Related Work**

Letelier's paper on building a requirement traceability framework was a major input to this paper. [9] He described the use of UML to create a common traceability framework. He establishes a UML reference metamodel for requirements traceability, representing all the software development artifacts and traceability links among them. He further used the UML extension mechanisms, for adapting UML for every projects needs. [9] Aizenbud-Reshef [2] established an approach for defining operational semantics for traceability using UML while Limon et al [10] analyzed current traceability schemes, for instance link types, in order to obtain relevant features and identify overlaps and inconsistencies among the approaches. Hu et al present a pattern- oriented approach for building a metamodel and defining the basic elements of a XML metamodel pattern. They further introduce pattern examples and define some rules on how to use these patterns. [6]

In other research we are developing a traceability process metamodel and profile. The traceability process models abstract the traceability world into sets of "entities" which explicitly capture the process, artifacts, and information flows with a great degree of detail and richness. In Figure 7 below we illustrate the process metamodel that we are working on.

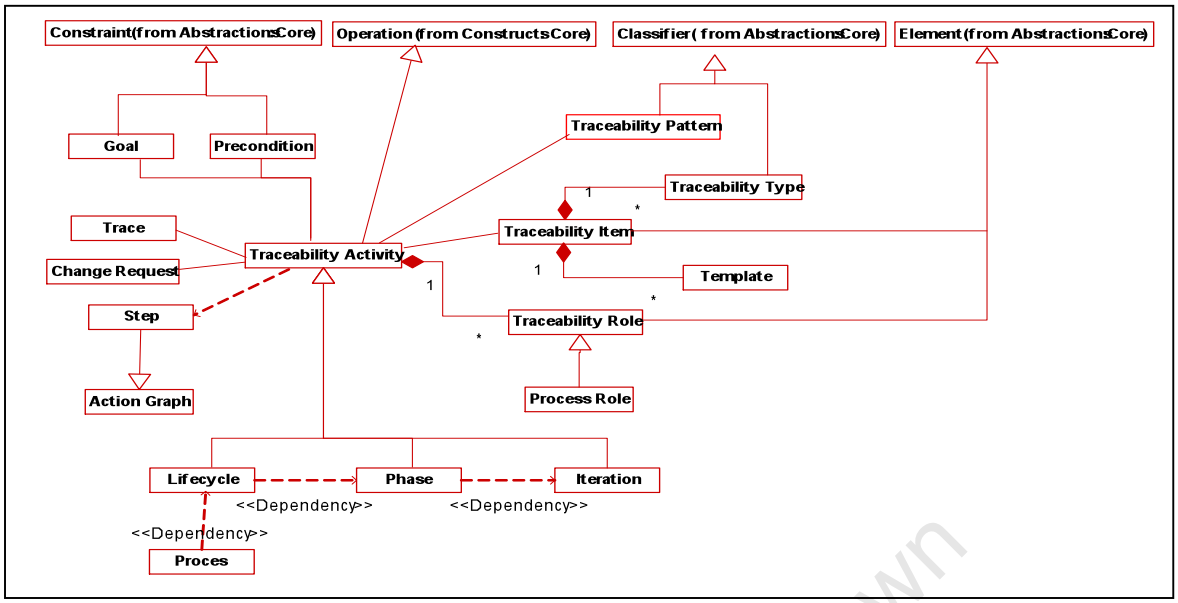



**Figure 0-7: Traceability Process Metamodel**

## Conclusions

Traceability patterns are intuitive components that are used in the creation of a traceability profile. Traceability rules that govern a traceability concept can be represented with a pattern with constraints encapsulated in it. Their objective is to create a common language to facilitate better communication, better reuse of traceability practices and to capture experiences in the development lifecycle. These patterns simplify complex problems by letting developers communicate them at a higher level of abstraction. They embody traceability knowledge and experience encapsulating a well-defined problem and its solution. The patterns help generate tested traceability rules as starting points for developers. A traceability pattern language is a set of cooperating patterns defined as abstract classes which outline their responsibilities and collaborations.

We have outlined an approach to elicit, categorize and create these patterns. The key observation of this paper is that during analysis of any traceability practices common patterns emerge. Our approach leads to better traceability practices and it encourages researchers to further develop this pattern approach with vigor and creativity.

**References**


1. Alexander C., A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977.
2. Aizenbud-Reshef1, N., Paige, A., Rubin, J., Shaham-Gafni1, Y., Kolovos, D., "Operational Semantics for Traceability", Proceedings of the 1st Traceability Workshop (ECDMA 2005), November 2005
3. Gills., M., "Survey of Traceability Models in IT projects", Proceedings of the 1st Traceability Workshop (ECDMA 2005), November 2005
4. Edwards M. and S. Howell. "A Methodology for System Requirements Specification and Traceability for Large Real-Time Complex Systems" Technical report, U.S. Naval Surface Warfare CenterDahlgren Division, Dahlgren, Va., 1991.
5. Egyed A. and P. Gruenbacher, "Automating Requirements Traceability---Beyond the Record and Replay Paradigm," Proc. 17th Int'l Conf. Automated Software Eng. (ASE), pp. 163-171, Sept. 2002.
6. Hu, Z., Vollmar, G., "Towards XML metamodel patterns for XML data modeling" 12th International Workshop on Database and Expert Systems Applications, Sept. 2001
7. Orlena C. Z. Gotel, Anthony C. W. Finkelstein. An Analysis of the Requirements Traceability

   Problem. 1994. Proceedings of the First International Conference on Requirements Engineering

   (ICRE '94), IEEE Computer Society Press, Colorado Springs, Colorado, U.S.A., April 18-22, pp. 94-101.
8. Letelier, P., "A Framework for Requirements Traceability in UML-based Projects", *Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TFFSE 2002)*, September 2002
9. Limon, A., Garbajosa, J., " The Need for a Unifying Traceability Scheme", Proceedings of the 1st Traceability Workshop (ECDMA 2005), November 2005
10. Sherba, S., Anderson, A., and Faisal, M. A Framework for Mapping Traceability Relationships in Proceedings of 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '03) (October 2003, 2003).
12. Spence I., Probasco L., "Traceability Strategies for Managing Requirements with Use Cases" Rational Software White Paper, 2000
13. Object Management Group. *Model-Driven Architecture Specification,* available at http://www.omg.org, last accessed May 2006.

# APPENDIX V EUROSPI 2005

## A Method for Modelling a Mature Process

*Justin Kelleher,*
*University of Cape Town,*
*Cape Town*
*jkr@cs.uct.ac.za*



**Abstract**

This paper describes the creation and assessment of a software traceability process. The project is part of a larger research project at the University of Cape Town. The purpose of this paper is threefold. Firstly, we describe a method to model a software process. Secondly we describe a method for assessing the capability of this process using the ISO 15504 standard. Thirdly, we compare the process capabilities of our process to that of the Rational Unified Process.

We describe the modelling of a process metamodel called **TRA**ceability **M**etamodel (**TRAM)**. TRAM provides a language for the definition of the elements of the **TRA**ceability **P**rocess (**TRAP**). The language used is based on the UML Software Process Engineering Metamodel (SPEM) specification which is defined by the Object Management Group. The goal of TRAP is to describe the implementation of software traceability across the product lifecycle. We describe the TRAP process which is a synthesis of best practices collected and developed in the course of the research programs activities which incorporates the best available requirements traceability techniques for telecommunications software projects.

We describe how we used the ISO 15504 framework in the assessment of the TRAP process. While ISO 15504 delineates a list of activities that should occur it does not to stipulate the order in which such activities should be carried out. This paper therefore proposes a process for modelling and assessing any software process.

We conclude with a comparison of the capabilities of the TRAP process to the capabilities of the Rational Unified Process.

**Keywords**

Traceability, ISO 15504, Rational Unified Process

## Introduction

The goal of software engineering is "to build a software product or to enhance an existing one" [18]. It is the disciplined approach and application of engineering, science, and mathematical principles, methods and tools to the economical production of quality software [6]. The IEEE defines a process as "a sequence of steps performed for a given purpose" [7] The Software Engineering Institute [12] states that "An essential aspect of software engineering is the discipline it requires for a group of people to work together cooperatively to solve a common problem. Defined processes set the bounds for each person's roles and responsibilities so that the collaboration is a successful and efficient one" Rational defines a process as "a set of partially ordered steps intended to reach a goal" [10]. In this paper, we introduce the **TRAP** (**TRA**ceability **P**rocess) which describes the roles and work activities for the implementation of software traceability across the product development lifecycle.

Lee Osterweil wrote in 1987: "Software processes are software, too." [15]. This notion has become accurate given that over the past two decades, visual modelling has developed as an essential discipline in software engineering.

A metamodel is a precise definition of the constructs and rules needed for creating semantic model elements at a high level of abstraction. Current literature describes a metamodel as an architectural blueprint. The process metamodel developed in this project are abstract descriptions of the implementation of a software process which has traceability as a core best practice. The Unified Modelling Language (UML) is an Object Management Group (OMG) standard used for creating a metamodel(s) of software artefacts and processes. We combine the UML and the Software Process Engineering Metamodel (SPEM) specification which is also defined by Object Management Group (OMG) to describe the TRAM and the RUP metamodel. The SPEM describes a metamodel "as a concrete software development process or a family of related software development processes" [16]. To engage in successful product development, businesses must rigorously assess their processes. The ISO 15504 standard, formly known as SPICE (Software Process Improvement and Capability dEtermination) is a "framework for the assessment of software processes" [9] and was developed by the International Organization for Standardization (ISO). The TRAP software process capability describes the range of expected results that can be achieved by following the process. We propose a capability for the TRAP and the RUP processes. We then proceed to analyse the capability of the process against the ISO target process profile with the aim of identifying which of the processes has the better capability. The basic hypothesis of this paper is: "can we model software processes using UML and SPEM, create a process from the models and assess the capability of the process using the ISO 15504 standard?" In summary the purpose of this paper is to describe an effective process modelling and assessment approach for the software traceability domain.

**Motivation For TRAceability Process (TRAP)**

Analysing the current body of literature on software traceability, Marco Leon [14] noted that traceability usage is rare, stating, “It is a very valuable but seldom used technique in today’s development processes. Traceability analysis is rarer still in the internet development industry, where it is even more essential” Scott Ambler (1999), states “It’s rare to find a software project team that can honestly claim full requirements traceability throughout a project, especially if the team uses object-orientated technology”

The IEEE has defined traceability as the identification and documentation of derivation paths (upward) and allocation or flow down paths (downward) of work products in the work product hierarchy [8] Gotel and Finkelstein have described traceability (1994) [5] as ”The ability to describe and follow the life of a requirement, in both a forward and backward direction, i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases”. Edwards and Howell (1991) [4] defined traceability as well, commenting that it is a technique used to “provide a relationship between the requirements, the design, and the final implementation of the system”. Palmer (1997) [17] has noted that “traceability gives essential assistance in understanding the relationships that exist within and across software requirements, design and implementation”. These relationships allow designers to demonstrate that the design meets the requirements as well as aiding as aiding in early recognition of those requirements not satisfied by the design. Palmer states that traceability sets out to show “how and why system development products satisfy stakeholder requirements”.

Two of the problems motivating the creation of a traceability process are, firstly there is no process in existence which focuses on software traceability across the entire product lifecycle and secondly many of the standards that mandate traceability do not provide a comprehensive guide explaining how to implement this best practice. For example, the standards governing the development of systems for the U.S. Government (e.g., MIL-STD-2167-A and MIL-STD-498 which replaces it (DoD, 1988)) require the development of requirements traceability documents, but don’t mandate how to do so. Overall, the practices and usefulness of traceability vary considerably across systems development efforts, ranging from very simplistic practices aimed simply at satisfying the mandates to very comprehensive traceability schemes used as an important tool for managing the systems development process.

TRAP is a process which can be adapted for any project. It describes the work products (traceability items or artefacts), the roles involved in creating the work products and their traceability responsibilities. The TRAP contains workflows conveying the development time for the product as a sequence, the traceability best practices and traceability guidelines, traceability patterns, and the range of traceability tools for seamless implementation of traceability in an organisation. The backbone to this process is the list of traceability items and their corresponding traceability matrixes.

## The Research Inputs

### Rational Unified Process (RUP)

The Rational Unified Process provides a process that can be customized to any software development organization's needs. A major characteristic of RUP is that it provides a disciplined approach to assigning tasks and responsibilities. This characteristic of the process proved to be very useful when we were assigning traceability tasks to the different roles. We decided to use RUP for two reasons. Firstly, it is a process framework with many traceability activities defined which is supported by an integrated tool suite. RequisitePro is a dedicated requirement management tool integrated with supporting tools and process workflows. The desired result is an easy-to-use process. Secondly, the RUP process model could be configured and adapted to satisfy our customers needs.

### Software Process Engineering Metamodel (SPEM)

SPEM is object-oriented specification which describes how to model a software process. UML is used as the notation. The SPEM is a metamodel for defining processes and their components. A tool based on SPEM is a tool for process authoring and customization.

### Architecture of integrated Information Systems (ARIS)

The ARIS Toolset integrates new and existing modelling methods for modelling processes and providing the functionality for creating and evaluating our modelled processes. The ARIS architecture is the basis for the ARIS Toolset. [20] It also serves as an orientation framework for complex development projects due to the fact that in its structuring elements it contains an implicit procedural model for the development of integrated information systems. The result is a highly complex UML metamodel, integrating the view of processes, knowledge processing, organisational structures and information systems. We evaluated a number of commercial process modelling tools but accepted the ARIS Toolset because it has the following process modelling features:

- Object Process Modelling: Represent the static, structural and data-related aspects of a process.
- Dynamic Process Modelling: Illustrate the software development lifecycle in both time and behaviour. The sequence of operations is described by mapping the sequences of events.
- Functional Process Modelling: Clarify the transient and functional aspects of the process i.e. roles mapped to responsibilities.



### Spice: ISO 15504

Several models of varying quality were studied: McCall [13], Boehm [1], FURPS , ISO 9126 [19], Dromey [3] , ISO 15504 and CMM [11] with the intention of identifying those which possess aspects deemed to be important in a Systems Quality Model. James A. McCall [13] described the problems encountered when defining software quality and the best technique for establishing a framework for the measurement of software quality.

The ISO 15504 document suite has a set of categories in which the assessors can place the data that they collect during their assessment. The result is that the assessors can give an overall determination of the organisation's capabilities, which in this project is the capability to implement traceability in the product lifecycle.

We selected ISO 15504 for the following reasons:

- It encourages self-assessment.
- It produces a set of process ratings (a process profile) rather than a pass/fail result. This is essential when comparing two processes.
- It addresses the adequacy of the management of the assessed processes;
- It takes into account the context in which the assessed processes operate;
- It is appropriate across all application domains and sizes of organization.

One of the main reasons we used the ISO 15504 was because of its international recognition and acceptance as a process standard. ISO 15504 does not conflict with social, cultural or legislative expectations and requirements. The actual standards document for ISO 15504 is divided into 9 parts [9]. The ISO 15504 framework defines the process practices for software engineering organisations as well as the measurement criteria to determine process capability. It assists the software development organisation in planning, managing, monitoring, controlling and improving the acquisition, supply, development, operation, evolution and support of software. We utilize the framework by assessing the process capability of TRAP and RUP and comparing the results.

**The Process Modelling Method**

In this section we describe the method we followed to model and assess the TRAP and the RUP processes. Figure 1 illustrates the methodology that we followed.

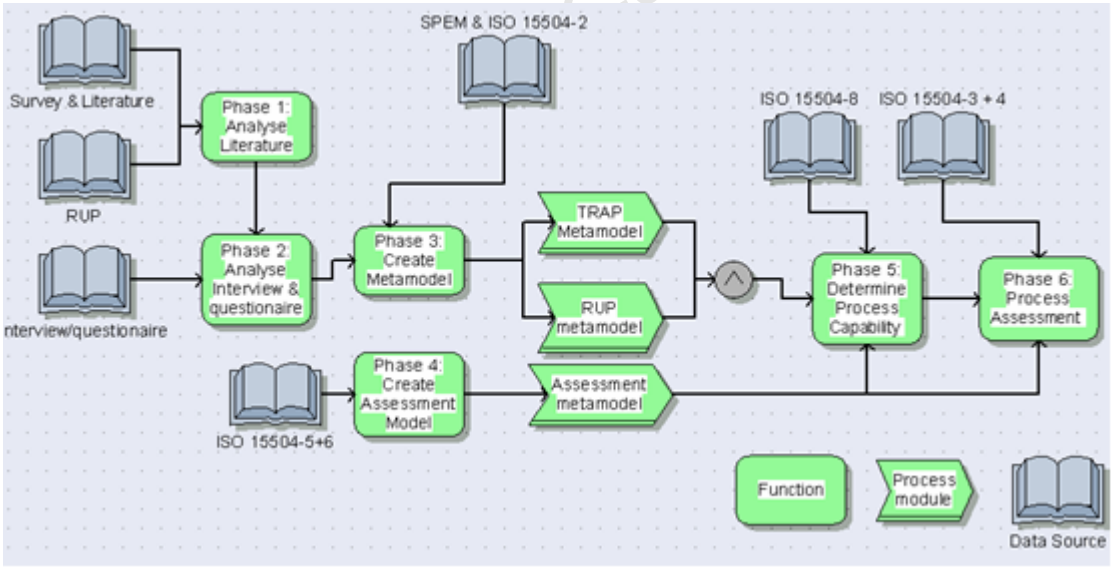



**Figure 1: The Methodology**

Table 1 describes briefly each phase, the inputs to each phase and the related outputs. During Phase 1 and 2 we created an encyclopaedia of software traceability practices. We designed and implemented a random survey by sending out questionnaires and interviewing experts in the telecommunications industry. The survey gave the process team a broader picture of the traceability practices being performed. During the interviews, the participants were asked to describe their localised traceability practices with a special focus on the relationships between the work products produced.

| Development Stage | Data Inputs | Outputs |
|---|---|---|
| Phase 1: Review Literature | ACM, IEEE, Academic research focus groups, experience documentation, ISO, Carnegie Mellon etc | Repository of reusable literature or best practices for TRAP |
| Phase 2: Interview + Questionnaires: | Review the interview and questionnaire from focus groups on process modelling and software traceability | Identify critical traceability problems and create document list for TRAP process blueprint |
| Phase 3: Create TRAP + RUP Metamodel: | The RUP + TRAP process metamodel ISO TR 15504 Software Process Engineering Metamodel (SPEM) | RUP, and TRAP metamodel (work products, activities, guidelines, work flows, best practices) |
| Phase 4 Create Assessment Model: | ISO 15504 Software Process Engineering Metamodel (SPEM) | TRAP and RUP process capability and maturity assessment report |
| Phase 5: | Questionnaire | TRAP and RUP process capability and maturity assessment report |
| Phase 6: | Review and validation of work with focus group | Approval report |

**Table 1: The Phases in the Method**

During phase 3 we model the TRAP and RUP metamodel. Figure 2 depicts the three layered abstract modelling architecture defined by SPEM. The three layers are described as:

- Level M2: The separate metamodel(s) of TRAP and RUP. The metamodel(s) al level M2 is compatible with the reference model defined in 15504-2 and the metamodel in SPEM, so that a common basis for judgment was employed.
- Level M1: The process definition. For example, RUP is defined at level M1. We adapt and configure a level M1 process for a process enactment at level M0.
- Level M0: A process enactment or instance, i.e. a process in production in a specific project.

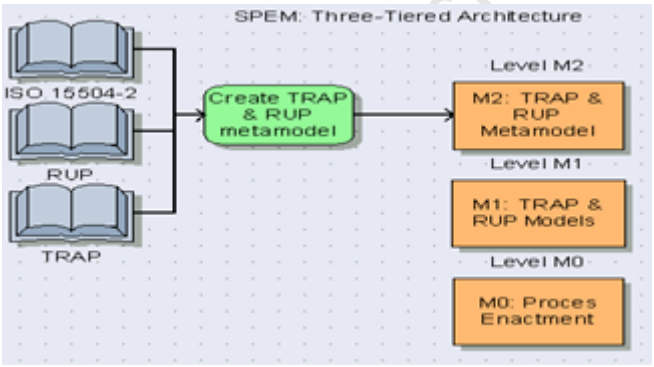



**Figure 2: SPEM 3 Layered Architecture**

## The Assessment Method

This section presents the assessment methodology we followed. Figure 3 below illustrates the TRAP and RUP assessment steps:

- Review the assessment input
- Select the process instances
- Determine the actual ratings
- Determine derived ratings

- Validate the ratings

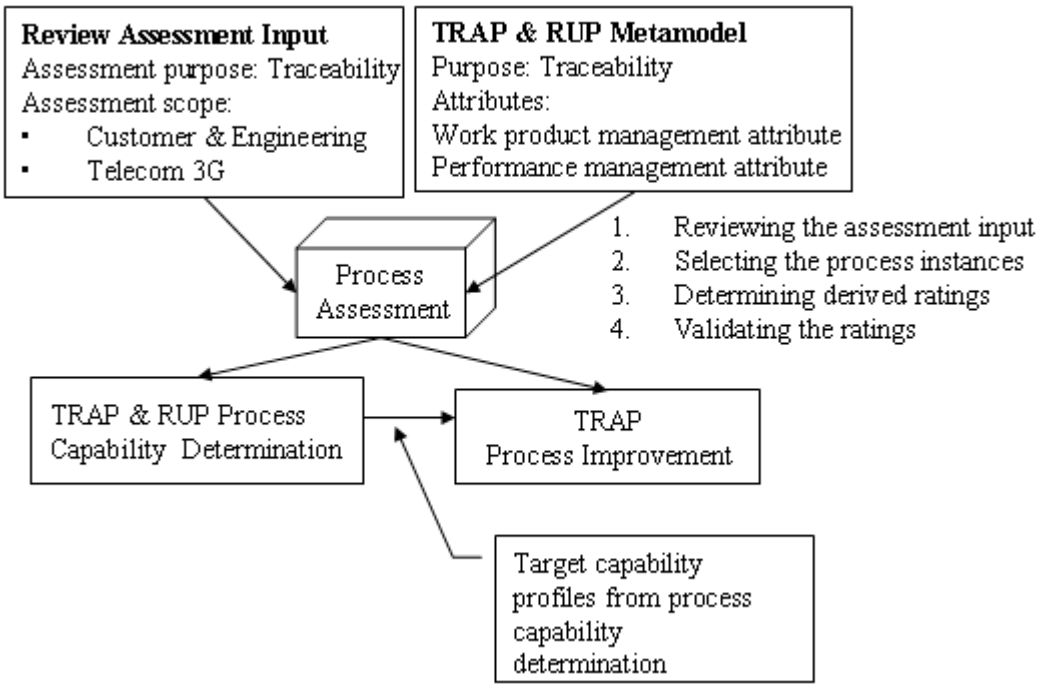



**Figure 3: The Assessment Method**



## Review the assessment input

**Purpose:** We designed and documented a unique assessment process for this research project. The main purpose was to assess the capability of the TRAP process and compare its capability to the RUP.

**Scope:** TRAP has traceability as the core engineering practices. Therefore we decided to limit the scope of the TRAM metamodel and corresponding TRAP process to the primary lifecycle process categories:

- *The Customer-Supplier* process category consists of processes that directly impact the customer, support development and transition of the software to the customer, and provide for the correct operation and use of the software product. The survey and interview results revealed that traceability to customer requirements is an important practice.
- *The Engineering* process category consists of processes that directly specify, implement or maintain the software product, its traceability to the system and its customer documentation.

A base practice is an activity that addresses the purpose of a particular process. For example tracing test cases to customer requirements is a base practice in the customer compliance process. Consistently performing the tracing practice associated with the compliance process will help to consistently achieve customer compliance. In Table 2 a coherent set of base practices is associated with each process in the process dimension. Management practices relate to the process attributes defined in the process capability dimension of the reference model. Evidence of their effective performance supports the judgement of the degree of achievement of the attribute. Management practices are the principal indicators of process capability. The set of management practices is intended to be applicable to all processes in the process dimension of the model. Evidence of the performance of the defined management practices can be derived from the practice performance characteristics.

| Process Category | Base Practice |
|---|---|
| Customer-Supplier:<br>CUS.1.1 Acquisition Preparation Process<br>CUS.1.4 Customer Acceptance Process<br>CUS.3 Requirements Elicitation Process | CUS.1.1.BP1: Prepare and negotiate contract.<br>CUS.1.1.BP4: Define acceptance criteria.<br>CUS.1.4.BP1: Evaluate the delivered product<br>CUS.1.4.BP2 : Accept the delivered product<br>CUS.3.BP1: Obtain customer requirements and requests.<br>CUS.3.BP2: Agree on requirements.<br>CUS.3.BP3: Establish customer requirements baseline.<br>CUS.3.BP4: Manage customer requirements changes. |
| ENG.1 Develop system requirements & design<br>ENG.2 Develop software requirements<br>ENG.3 Develop software design<br>ENG.4 Implement software design<br>ENG.5 Integrate and test software<br>ENG.6 Integrate and test system<br>ENG.7 Maintain system and software | ENG. 1.1 BP Specify system requirements<br>ENG. 1.2 BP Describe system architecture<br>ENG. 1.3 BP Allocate requirements<br>ENG.1.4 BP Determine release strategy<br>ENG.1.1 BP7 : Establish traceability.<br>ENG. 2.1 BP Determine software requirements<br>ENG. 2.2 BP Analyse software requirements<br>ENG. 2.3 BP Determine operating environment impact<br>ENG. 2.4 BP Evaluate requirements with customer<br>ENG.2.5 BP Update requirements for next iteration<br>ENG. 3.1 BP Develop software architectural design<br>ENG. 3.2 BP Design interfaces at top level<br>ENG. 3.3 BP Develop detailed design<br>ENG. 3.4 BP Establish traceability<br>ENG. 4.1 BP Develop software units<br>ENG. 4.2 BP Develop unit verification procedures<br>ENG. 4.3 BP Verify the software units<br>ENG. 5.2 BP: Build aggregates of software units<br>ENG. 5.5 BP Develop tests for software<br>ENG. 5.6 BP Test integrated software |

**Table 2: TRAP Process Dimension and Base Practices**

## Select the process instances

We identified the process instances for the assessment using ISO 15504-3. The TRAP software process model is an abstract representation of the way people work. Because different projects have varying levels of adherence to the model, its specific realisations on each project are called process instances. In order to provide a consistent basis for assessment, part two of the ISO 15504 document suite establishes a process model that is representative of the software process as a whole.

## Determine the actual ratings

In addition to reviewing the scales and actual results of the assessment, this section shows how the actual rating was performed. The assessment was implemented as workshop sessions. We determined the capability of TRAP and RUP against the reference model described in ISO/IEC 15504-2. Processes in the reference model are grouped according to the type of activity they address. Each process has a defined purpose describing the high-level objectives that the process should achieve. The purpose statements describe what to do, but do not prescribe how the process should achieve its objectives. Although the reference model contained in ISO 15504-2 covers a range of processes applicable to the software process, we evaluated the capabilities of TRAP and RUP only for the processes related to traceability. We determined the process capability in a systematic assessment and analysis of the TRAP and RUP software processes, carried out with the aim of identifying the strengths, weaknesses and risks associated with deploying the traceability process. The output of a process capability determination is the process capability report. It summarizes, for each key process included within the target capability statement, strengths and weaknesses expressed in terms of process attribute gaps, and the risks associated with each.

We defined the process attributes, their rating scale, and the process capabilities levels. The

process attributes were used to determine whether a process has reached a given capability. Each attribute measures a particular aspect of the process capability. The attributes are themselves measured on a percentage scale and therefore provide a more detailed insight into the specific aspects of process capability required to support process improvement and capability determination. The rating scale is a percentage scale from zero to one hundred percent that represents the extent of achievement of the attribute.

The ratings are as follows:

- **N** Not achieved: 0% to 15% - There is little or no evidence of achievement of the defined attribute in the assessed process.
- **P** Partially achieved: 16% to 50% - There is evidence of a sound systematic approach to and achievement of the defined attribute in the assessed process. Some aspects of achievement may be unpredictable.
- **L** Largely achieved: 51% to 85% - There is evidence of a sound systematic approach to and significant achievement of the defined attribute in the assessed process. Performance of the process may vary in some areas or work units.
- **F** Fully achieved: 86% to 100% - There is evidence of a complete and systematic approach to and full achievement of the defined attribute in the assessed process. No significant weaknesses exist across the defined organizational unit.

In order to rate a process one must decide what the process indicators are. An indicator is defined as an objective attribute or characteristic of a practice or work product that supports the judgement of the performance or capability of an implemented process. We defined *Traceability indicators* which confirmed that certain traceability practices were performed. The existence of base practices, work products, and work product characteristics, provide evidence of the performance of the processes associated with them. Similarly, the existence of management practices provides evidence of process capability.

Management practices relate to the process attributes defined in the process capability dimension of the reference model. Evidence of their effective performance supports the judgement of the degree of achievement of the attribute. Management practices are the principal indicators of process capability.

*For example TRAP has work product management as one of its process attributes. The indicators for this managed practice is to maintain the traceability of functional, non-functional and quality requirements, maintain work products under configuration management and baseline copies of the work product for the process correspond to the project's current development status.*

We established the TRAP and RUP ratings as follows. Firstly, adequacy ratings (F, L, P or N) were determined for all base practices and for all generic practices with respect to each base practice. Then the ratings were converted to percentages by dividing for each capability level (1 to 5) the amount of ratings on each adequacy level (F, L, P or N) with the amount of ratings within that capability level. The resulting percentages were then used to create diagrams and to derive further ratings. Figure 4 shows the determined capability results of TRAP and RUP.

## Validating the ratings

We used self assessment of the TRAP and RUP process. We are currently validating our results using an independent assessor. The TRAP process metamodel has been configured for process enactment and is currently being tested by a number of local companies in Cape Town.

## Findings of Assessment

We classified TRAP and RUP into similar categories as ISO 15504 to simplify the assessment results. The following are the TRAP processes capabilities:

**Level 1** *The software design and implementation processes*. The TRAP process performance attribute is that the process transforms the identifiable input work products to produce identifiable output work products. The traceability relationships are difficult to identify and represent due to the complexity of the design models and implemented code. The TRAP managed practices ensured that the work products are produced. We conclude that traceability is difficult to describe at the design and implementation levels.

**Level 2** *The customer acquisition and preparation process and the engineering process for the integration and testing* of the software were determined at the capability level 2. The TRAP process described the requirement (functional and non-functional) work products, how to document and control these work products, the traceability dependencies among the work products and how to control changes to the requirements.

**Level 3** *The requirement elicitation process*, the *architectural requirement process* and the *software requirements process* were determined as Level 3. These processes satisfied the work product management attribute but also the process resource attribute. TRAP described the roles involved in software traceability, their corresponding responsibilities and competencies required for performing the traceability process and the process infrastructure required for performing the traceability process was identified.

The results for the RUP capability determination were:

**Level 1** *The customer acquisition and preparation process* and the *software design and implementation processes*. The customer acquisition process is poorly defined in RUP. However, the process performance attribute that the process transforms the identifiable input work products to produce identifiable output work products was true. Like in TRAP the traceability relationships were difficult to identify and represent due to the complexity of the design models and implemented code.

**Level 2** *The customer requirements elicitation process*, the *system and software requirements process and the integration and test process* were determined to have a level 2 rating. We determined that the requirements and testing discipline are the two most mature process disciplines in RUP. The integration between the requirement management and test management environments was taken into consideration in its process rating.

Traceability is a poorly defined practice in RUP. For example, RUP describes the management of traceability dependencies in the requirements discipline but omits this practice in the business modelling discipline.

| TRAP Process Dimension | Capability Level 1 | Capability Level 2 | Capability Level 3 |
|---|---|---|---|
| Cus 1: Customer acquisition preparation process | | | |
| Cus. 2: Customer acquisition process | | | |
| Cus 3: Customer requirements elicitation process | | | |
| Eng. 1:Develop system requirements and design | | | |
| ENG.2 Develop software requirements | | | |
| ENG.3 Develop software design | | | |
| ENG.4 Implement software design | | | |
| ENG.5 Integrate and test software | | | |

| RUP Process Dimension | Capability Level 1 | Capability Level 2 | Capability Level 3 |
|---|---|---|---|
| Cus 1: Customer acquisition preparation process | | | |
| Cus. 2: Customer acquisition process | | | |
| Cus 3: Customer requirements elicitation process | | | |
| Eng. 1:Develop system requirements and design | | | |
| ENG.2 Develop software requirements | | | |
| ENG.3 Develop software design | | | |
| ENG.4 Implement software design | | | |
| ENG.5 Integrate and test software | | | |

Figure 4: TRAP Process Dimension and Base Practices

## TRAP Process Outcome

TRAP is intended to provide project managers and requirements engineers with a standardised and efficient means of handling requirements through the full product development life cycle. In particular the process is intended to encourage and support the handling of traceability in an iterative development context. TRAP incorporates the best available requirements traceability techniques for telecommunications software projects in any open systems domain. TRAP is structured as a sequence of high level workflows. Each workflow is broken into discrete steps supported by descriptions of traceability activities, roles and artefacts associated with each step. As the workflows and steps are organised in chronological sequence within the product development life cycle, the process should ideally be read in sequence. Workflow diagrams and descriptions are extensively enabled with hyperlinks to also facilitate nonlinear navigation within the process. In Figure 5 below we see an example taken from TRAP of the requirement engineering traceability activities, the input artefacts, the output artefacts and the tool environment for a local enactment.

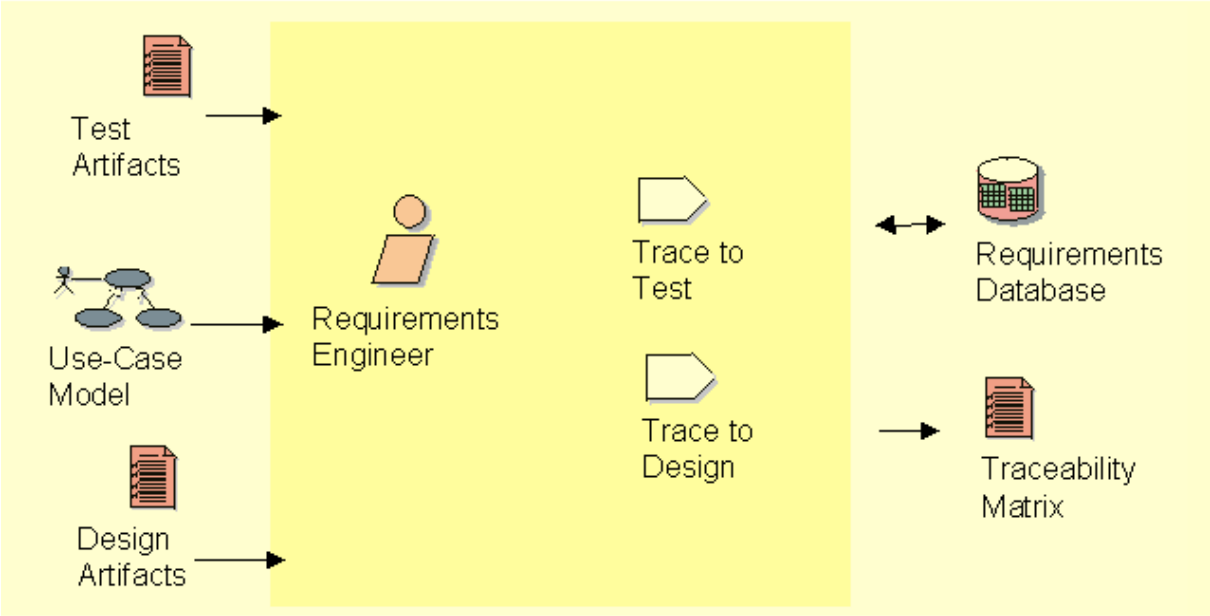



**Fig. 6. TRAP Traceability Process Enactment**

## Conclusion

This paper is part of a larger research project on requirement traceability in the product lifecycle, with the research being undertaken in the University of Cape Town and the process enactments occurring in small and medium enterprises in Cape Town. The overall goals of this project and TRAP is to encourage organizations interested in improving their processes and in particular their traceability practices to employ reliable methods for creating and assessing their process. This paper establishes a common framework for expressing the process capability ratings for a 15504-conformant assessment and it provides models and methods for building a traceability process.

To overcome the problems of process reusability we recommend that organisations begin by creating a configurable process metamodel. The Object Management Group (OMG) defines a three-layered architecture for process modelling. Our research proposes that the Object Managements Group specification SPEM is a suitable best of breed process modelling specification. The TRAM process metamodel we generated provides a language for describing the elements of the process. We supported the metamodel with a process authoring tool which publishes product lifecycle process configurations as a web site for practitioners to access.

We defined TRAP to have traceability as its core base practice. TRAP was evaluated to have a high capability level. We assessed RUP under similar conditions. While RUP has evolved into a rich family of integrated software-engineering process products we conclude that it does not support traceability consistently across the entire software development lifecycle. For example, RUP describes the activities associated with traceability in the requirements discipline but does not discuss traceability during the deployment phase. One of the outcomes of our assessment of RUP is that we deem many of its core practices asymmetrical and inconsistent. IBM have made few changes to the RUP workflows and activities over the past few years. We recommend that RUP undergo an independent ISO 15504 assessment and that IBM publish the results or make the necessary process improvements to maintain its position as a leading commercial process framework.

Overall we encountered many problems in interpreting the ISO 15504 models, due to the volume and complexity of the document suite. Our intention was to create and evaluate a simple traceability process and while ISO 15504 is a complete framework we conclude that the effort we put in using the framework is not reflected in the results we obtained . The most interesting and valid data came from the observations and interviews with industrial

experts. We therefore argue that the ISO 15504 framework needs to be streamlined and abridged for more agile approaches.

**Literature**


1. Boehm B. W., Brown J. R., Lipow M. Quantitative Evaluation of Software Quality. In Proceedings of the 2nd international conference on Software engineering San Francisco, California, United States. Pages: 592 - 605 (1976)

2. Boehm Barry W., Kevin J. Sullivan. Software economics: a roadmap. In International Conference on Software Engineering Proceedings of the Conference on The Future of Software Engineering. ACM Press New York.

3. Dromey R. Geoff. Cornering the Chimera. IEEE Software Volume 13 , Issue 1 (January 1996) Pages: 33 - 43. IEEE Computer Society Press Los Alamitos (1996).

4. Edwards M. and S. Howell. A Methodology for System Requirements Specification and Traceability for Large Real-Time Complex Systems. technical report, U.S. Naval Surface Warfare CenterDahlgren Division, Dahlgren, Va., 1991.

5. Gotel, O. & Finkelstein, A. (1994). An Analysis of the Requirements Traceability Problem. 1st International Conference on Requirements Engineering (ICRE'94), Colorado Springs, April 1994, pp. 94-101.

6. Humphrey W. S. The software engineering process: definition and scope. In Proceedings of the 4th international software process workshop on Representing and enacting the software process.(Devon, United Kingdom 1988) Pages: 82 - 83 ACMPress, New York, 1989

7. IEEE-STD-610 ANSI/IEEE Std 610.12-1990.IEEE Standard Glossary of Software Engineering Terminology. February 1991

8. IEEE Guide for Information Technology, IEEE-STD 1362-1998, Available: http://standards.ieee.org/catalog/olis/se.html

9. ISO/IEC TR 15504. International Standard for Software Process Assessment. Available: http://isospice.com/standard/tr15504.htm

10. Jacobson I., Booch G. & Rumbaugh J., Unified Software Development Process, Addison-Wesley, 1999

11. Lawlis, P.K., Flowe, R.M., and Thordahl, J.B. A correlational study of the CMM and software development performance. CrossTal/e (Sept. 1995), 21-25.

12. Linda M. Northrop. A Framework for Software Product Line Practice Version 4.2. Software Engineering Institute. Available: http://www.sei.cmu.edu/productlines/frame report/process def.htm

13. McCall James A., Joseph P. Cavano. A framework for the measurement of software quality. ACM SIGSOFT Software Engineering Notes Volume 3 ,Issue 5 (November 1978) Pages: 133 - 139. ACM Press New York(1978).

14. Marco Leon, Intelligent Enterprise Magazine, July 17, 2000, Vol 3 Number 12 http://www.intelligententerprise.com/000908/ebusiness.jhtml

15. Osterweil L. Software processes are software too. In Proceedings of the 9th international conference on Software Engineering (Monterey, California, United States 1987) Pages: 2 - 13. IEEE Computer Society Press Los Alamitos 1987

16. OMG. Software Process Engineering Metamodel (SPEM). Available: http://www.omg.org/docs/formal/05-01-06.pdf

17. Palmer J.D., Traceability. Software Requirements Eng. R.H. Thayer and M. Dorfman, eds., pp. 364-374, 1997.

18. Philippe Kruchten, The Rational Unified Process-An Introduction, 2nd ed, Addison-Wesley-Longman, Reading, MA (2000)

19. Shahid Nazir Bhatti. Why quality?: ISO 9126 software quality metrics (Functionality) support by UML suite. ACM SIGSOFT Software Engineering Notes ACM Press New York, NY

20. Zachman J.A., 'A framework for information systems architecture', in: IBM Systems Journal, 1987, vol. 26, nr. 3, pp. 276-292.

# APPENDIX VI IASTE 2006 PAPER

(Proceedings of the 17th IASTED international conference on Modelling and simulation)

# Utilizing Use Case Classes for Requirement and Traceability Modeling

Justin Kelleher
Department of Computer Science
University of Cape Town
Cape Town, South Africa
+27216505108
jkr@cs.uct.ac.za

Mikael Simonsson,
NADA
Royal Institute of Technology
Stockholm, Sweden, +46708145929,
mikael@monter.se

**ABSTRACT**

Changes to the UML 2.0 revision indicate that clarifications on the future of traditional use case needs to take place. The indications are that the use case notation can be replaced by use-case classes. Use case classes model requirement types as cohesive package consisting of requirement attributes and operations. The UML 2.0 supporting documentation does little to demystify the exploitation of the use case classes. The layered Model Driven Architecture and the underlying techniques recommended by the OMG may provide the solution.

In this paper we review the critical changes to UML 2.0 regarding requirements modeling. We demonstrate that use case classes are formal templates for describing rules on modeling requirements with instances. We present a class hierarchical structure representing the complex relationship between business, product and project requirements using UML dependencies. We adapt the UML 2.0 extension mechanism notation to depict requirement traceability links. Replacing use cases with classes and utilizing the explicit traceability links we integrate the requirement elements and the design elements into the same work space. Bridging the gap between requirements and the rest of the development process makes the test effort easier. For example, the use case classes serve as system and function test drivers**.**

We show that requirement patterns are descriptions of communicating objects and classes. The UML extension mechanisms represent the requirement traceability links. Therefore by recognizing recurring dependency relationships of we are in fact uncovering traceability patterns. We support our project with a simple requirement process framework.

Overall, we propose a new requirement modeling approach for the visualization, communication and reuse of requirements and requirement traceability using new object technologies.

**Keywords**

UML 2.0, Requirement Metamodel, Requirement Patterns, Traceability

**Context & Motivation**

The global economic slowdown has resulted in the reluctance of software companies to begin new developments. Many companies are opting for the cheaper option of updating and maintaining existing product-lines. [9] The development of families of complex software-intensive products has become a reality. [25] This new complexity combined with tighter regulations on requirement compliance and product concurrence has renewed an interest in visual requirement architectures and its symbiotic practice of requirement traceability.

The Unified Modelling Language (UML) [5] is the accepted standard for specifying different models. Like any loosely defined technology there has been a lot of speculation since its adoption by the Object Management Group (OMG). In UML use cases are informal methods for visually communicating scenarios with the stakeholders and documenting the functional requirements in the system. A use case captures the requirements as "a specific flow of events through the system, that is, an instance" [12] Requirements represented in uses cases diagrams use natural language and standard templates and must be supported by an adaptable development process which addresses

bridging the gap between the requirement and design models. A use-case realization describes how a particular use case is realized within the design model, in terms of the collaborating objects. The use-case realization provides a construct in the design model which organizes and traces artifacts related to the use case but which belong to the design model. These related artifacts consist typically of the collaboration and sequence diagrams which express the behavior of the use case in terms of collaborating objects. [16] The root problem appears to be that organizations finally understand use case engineering but are still unable to bridge the gap of tracing them to the lower level documentation. For example in Rational Rose, attaching a non-functional requirement as a supplementary specification to a functional use case is an inconsistent informal approach. Organizations do not understand that a use case diagram is a specialized class diagram.

In all versions of UML 1, the different behavioral models were independent, but dramatic changes in UML 2.0, stipulates that all the elements derive from a fundamental definition of a behavior. Use Cases are the subtle exception to this rule. UML 2.0 is divided into the UML 2.0 Infrastructure component [20] and Superstructure component [21]. The UML 2.0 Infrastructure standard defines the base classes that form the foundation for the UML 2.0 superstructure which defines six structure diagrams and three behavior diagrams. In Section 4 we review the changes to UML 2.0 and describe how the changes improve the relationship between the structural (class) and behavioral models (use-cases). Our critical hypothesis is the replacement of the use case for the use case class which broadens the scope of requirements to include all the open specification object technology.

UML has standardized the modeling languages and now the the new trend is to describe the system functionality in platform independent metamodels which can be translated for any specific domain.[24] The Model Driven Achitecture (MDA) combines multiple standards, including Unified Modeling Language (UML), the Meta-Object Facility (MOF), the XML Metadata interchange (XMI). A metamodel is a precise definition of the constructs and rules needed for creating semantic model elements at a high level of abstraction. It serves as a template for a model or in other words is an instance of a metamodel. In section 3 we suggest a requirement metamodel which we support with a requirement process framework described in section 8.

Requirements Traceability is an important means to facilitate communication [18] among the success-critical stakeholders, to ease determining the impact of changes and support their integration, to preserve knowledge and dependencies created during the design process, to assure quality, and to prevent misunderstandings.[3] Neglecting traceability or capturing insufficient and/or unstructured traces leads to a decrease in system quality, causes revisions, and thus, increases project costs and time. [8] [11] By exploiting use case classes and utilizing the UML extension mechanisms we can improve our traceability practices and further bridge the gap between analysis, design and test. In section 5 we explain by example a use case class. In Section 6 we review the UML extension mechanisms and explain how they can be applied to requirement traceability.

Related research also considers new approaches for improving requirement modeling practices. Shrotri, U et al at the 2003 IEEE Software Engineering and Formal Methods conference proposed using UML object diagrams for specifying pre- and post-conditions for use cases. [22]. At the 2005 ACM symposium on Software Visualization, Kholkar proposed to bridge the gap between the natural language used in use cases by extending the set of UML diagrams with three new diagrams that would enable rigorous specification,

analysis and simulation of requirements.[14] In 1995 Jansson described an approach that reifies a use case as a class [13].

The basic hypothesis of this paper is: "can we model requirements with use case classes, utilizing class notation and hence recognizing patterns to facilitate a new more formal approach to requirement modeling and requirement traceability"

**Requirement Traceability**

The increasing use of commercial requirements traceability environments by industry reflects that traceability is still recognized as a critical practice [7] Requirements traceability allows us to assure the continuous concordance between the stakeholder's requirements and the artifacts produced along the software development process. Zisman describes the importance of requirement traceability for analyzing the impact of new requirements or changes to existing ones. [25] Software development generates and maintains a wide range of artifacts, such as documentation, requirements, design models, and test scenarios; all of which add value to the understanding of the software system. [6]

For this paper, the term "traceability item" needs to be contextualised. The term defined by Spence and Probasco [23] as "Any textual, or model item, which needs to be explicitly traced from another textual, or model item, in order to keep track of the dependencies between them". We propose that a requirement traceability item should be a "use-case class". We define the term "requirements traceability" as the explicit tracing of requirement objects to other requirement objects, models, test requirement objects, and other traceability objects such as design and user documentation. Trace dependencies identify the relationships among the objects.

This concept was addressed by Knethen [15] who generated a conceptual traceability reference model for the embedded systems domain which improved impact analysis practices. However the primary input to this paper is Patrico Leteliers seminal paper which describes structuring a requirement traceability framework using UML. He presents a UML reference metamodel for requirements traceability, which captures all the software development artifacts and corresponding traceability links using UML. [17]



**Requirement Metamodel**

We begin the process by creating a metamodel for the requirement domain. Metamodels describe models at high levels of abstraction. A requirement metamodel is a precise definition of the constructs and rules needed for creating semantic requirement models. This domain-specific modeling (DSM) raises the level of abstraction by aligning it to a particular problem domain. A requirement model is an instance of a requirement metamodel which in turn can be used as a metamodel of another model in a recursive manner. A requirement model contains requirement elements, which traditionally were use cases. A change to a requirement any where in the hierarchy will be reflected in the model but not in metamodel. The requirement meta-model must be agreed with all stakeholders and becomes a part of the project.

Due to changes in UML 2.0 use cases are now replaced by use case classes. These classes are created by instantiating model elements from a metamodel. Requirement objects are the domain of a model and, as such, are always complete, precise, and concrete. Models of objects (such as value specifications) can be incomplete, imprecise, and abstract according

to their purpose in the model. The typical role of a requirement metamodel is to define the semantics for how model elements in a model get instantiated.

The UML 2.0 Infrastructure standard recommends a multi-layered approach; the metamodel, the model and the objects. They describe the MOF as a meta-metamodel. The Meta-Object Facility (MOF) is the OMG's adopted technology for defining metadata and representing it as CORBA objects. [19] [20]

An illustration of how these meta-layers relate to each other is shown in Figure 1. There is no rule that meta-layers are restricted to four layers. The meta-layers are usually numbered from M0 and upwards. Layer M3 MOF uses the meta-class notation to define the model elements at a meta-layer. These meta-classes are then instantiated as object instances at the requirement model layer at M2. For the purpose of this paper we define requirements in two categories. The project requirements at M0 inherits from the product requirements at M1. For example, Product A is developed in Project B and Project C. There are two releases of the product at the end of each project. Therefore, the product requirement to "Logon to the System" is the parent for Project "Logon to System" Project A and "Logon to System" Project B. A change during Project A or B will have impact in the overall requirement model.



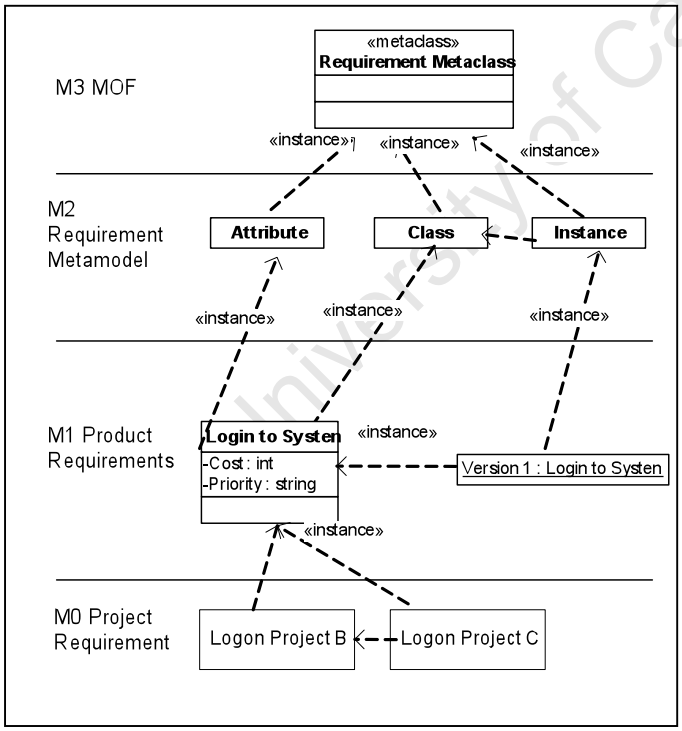



**Figure 0-1 Four Layered Requirement Metamodel**

We can further classify requirements at the M2 requirement metamodel layer into functional requirements, non- functional requirements and constraints. Non-functional requirements describe properties like a quality characteristic that are not readily captured in behavioral requirements artifacts such as use-case specifications. Constraints limit the

development such as defining the operating system or defining the programming language. In Figure 2 we have three requirement metaclasses in our metamodel.

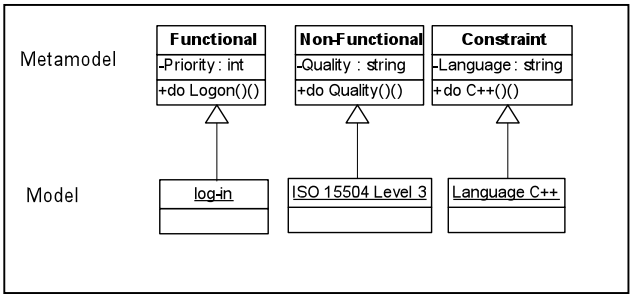



**Figure 0-2 Requirement Types as metaclasses**

A requirement model like any other model contains three major categories of elements: classifiers, events, and behaviors. The requirement model is an incarnation of the system to be built. "A classifier describes a set of objects; an object is an individual thing with a state and relationships to other objects". An event describes a set of possible occurrences; an occurrence is something that happens that has some consequence within the system. A behavior describes a set of possible executions; an execution is the performance of an algorithm according to a set of rules.

A use case also defines the interactions between external actors and the system under consideration to accomplish a goal. Actors are parties outside the system that interact with the system; an actor can be a class of users, roles users can play, or other systems. As shown in Fig 3 an actor may also be represented using a class rectangle with the stereotype «actor». [19]

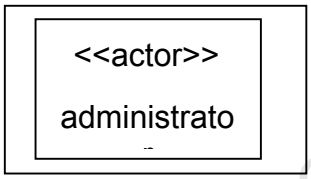



**Figure 0-3 UML 2.0 Actor Notation**

In the UML Infrastructure a classifier is defined as "A collection of instances that have something in common. A classifier can have features that characterize its instances. Classifiers include interfaces, classes, datatypes, and components". The owning of a use case by a classifier is represented with a class and the use case inside it. [19]

In the UML 2.0 Superstructure it describes that a use case can be shown using the standard class notation for classifiers. As can be seen in Fig 4 the standard recommends an ellipse icon in the upper-righthand corner of the class. If there is <includes> and <extends> the extension points may be listed in the operations compartment.

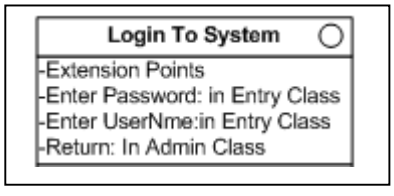



**Figure 0-4 UML 2.0 New Use Case Notation**

**Impacts of changes in UML 2.0**

In this section we review the changes to the UML 2.0 standards that impact Requirement Management and Traceability. UML continually evolves for the changing environment and technologies. For example, the UML 1.1 metamodel formally defined the "uses" and "extends" use case relationships as stereotypes of generalization. The revised UML 1.3 dropped these in favour of new "include" and "extend" relationships, which are styled instead as kinds of dependency. UML1.5 specifies that "an actor instance calls use-case operations" in the execution procedure of a use-case instance. This statement alone lead to much confusion and debate.

UML2.0 deletes the expression "an actor communicates with a use case" and replaces it with a new expression "an actor interacts with a subject". [19] It continues with "The relationship between a use case and its subject has been made explicit. Also, it is now possible for use cases to be owned by classifiers in general and not just packages". This dramatically changes our view of use cases. Sadahiro Isoda, passionately argues that the Actor-Calls-Use Case conjecture makes developers incorrectly believe that "a use case is like a system operation"

UML2.0 does not explicitly specify what properties (i.e., attributes and operations) a use-case class/object has. The behavioral aspect of the class/object must be encapsulated in the objects operations. In Fig 2 the use-case subclass inherits the attributes and operations from the superclass. Therefore the model class inherit the +do logo(), +do quality, do C++() from the metamodel. It is for this reason that we recommend that the requirement metamodel is configured at the start of every project and is kept under strict source control for the duration of the project.

In UML 1, the different behavioral models were independent, but in UML 2.0, they all derive from a fundamental definition of a behavior. This improves the relationship between the structural and the behavioral models. The nested classifiers in UML 2.0 let you designate that a behavior represented by a State Machine, Sequence Diagram or Activity Diagram is the behavior of a class or a component. In the UML Infrastructure document it states that a classifier is "A collection of instances that have something in common. A classifier can have features that characterize its instances. Classifiers include interfaces, classes, datatypes, and components". The owning of a use case by a classifier is represented with a class and the use case inside it. In UML 2.0, you can nest a set of classes inside the component that manages them, or embed a behavior (such as a state machine) inside the class or component that implements it.

We recommend that all requirements should now be described as use-case classes. These requirement classes represent an early conceptual representation for scenarios in the system, which have responsibilities and behavior. Each class describes a set of objects that share the same responsibilities, relationships, operations, attributes, and semantics. The requirement class is an abstraction for grouping requirements, for example we can define functional or non-functional requirements in classes and as metaclasses. As we will see in the next Section 6 we can employ the UML extension mechanism to now represent requirement traceability.

**Use Case Class Example.**

Requirements can be represented as use cases and use cases as classes whose instances are scenarios. The use case model must be made into a class category.

In OMG's UML 2.0 Infrastructure Specification, we can find the following definition of a UML Class: "A class describes a set of objects that share the same specifications of features (i.e. operations or attributes0, constraints, and semantics." [20] Also, a class may have relationships to other classes. An instance (i.e. object) derived from a class must have a "unique identity". A requirement is a description of what a system should do. A requirement is an entity holding a unique identity, a set of attributes and a set of traces (i.e. relationships). A requirement class is a template for creating requirements. Use Case Diagrams are a specialization of Class Diagrams such that the classifiers shown are restricted to being either Actors or Use Cases. [19] [20]

The following simple method begins by creating a use caseclass. In figure 5 we illustrate that each use case is an operation of the class *System*. We describe each actor by another class Actor. Each actor is an instance of *Actors*. A class is then created for each use case operation on *System*, each scenario is added as an operation of this class.

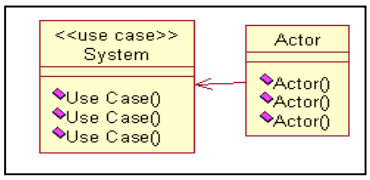



**Figure 5: Use Case Class Diagram**

In figure 6 we see that an Actor object starts a scenario by sending a message to the System, which in turn instantiates a use case object. The second message sent by the actor is the scenario name. For example, the main flow. It is sent to the instantiated use case object. Additional objects, that are an actual part of the system, can then be instantiated by the scenario. The actor interacts with the new class objects normally. The actor can also send additional use cases messages (to *System*) or scenario messages (to *UseCase* objects).

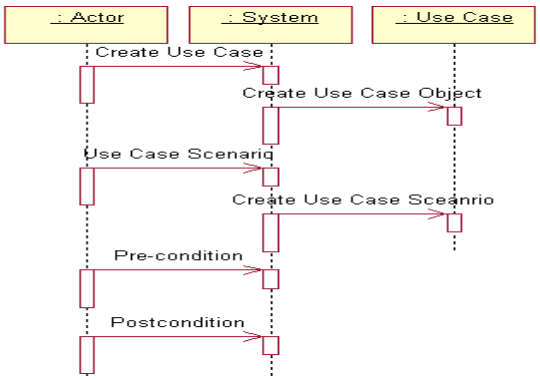



**Figure 6: Use Case Scenario Diagram**

In his way we can represent all aspects of use case design in class design. The rationale for instantiating use cases as classes is primarily for traceability links for testing source code. The use case classes serve as system and function test drivers.

**Class Associations and Traceability**

We use a UML Class to represent a requirement template and the instances to represent requirements. We illustrate a requirement template with a square containing two compartments. The top compartment is used for the *requirement template name* and the *requirement stereotype*; the requirement stereotype specifies if the requirement is *functional, non functional* or a *base requirement.* The bottom compartment is used for the requirement attributes). In figure 7 below we show the visual representation for a functional hardware requirement.

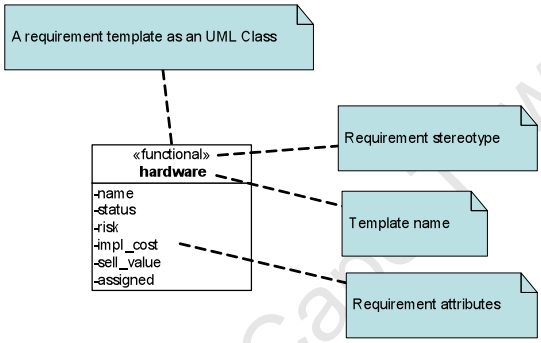



**Figure 7: A requirement template as an UML Class**

Different templates can be created for different kinds of requirements. For example, if we want to set up a template for a test case we change the template name to "test case", set the stereotype to functional and assign some appropriate attributes like: *test manager, test date, passed test* etc.

Traces between requirements are visualised by *UML Relationships*. The different UML Relationships describe the different trace types. This extends the number of traceability types. For example an *association* is a general relationship between two requirements. It means that Requirement A is associated to Requirement B; or according to the UML 2.0 specification, "A relationship that may occur between instances of classifiers……." [20] An association has a descriptive name, and is drawn with a line connecting the two requirements. On each end of an association there may be a *multiplicity* notation and a *navigability* notation. The multiplicity notation defines how many instance of Requirement *A* are associated with Requirement *B*. The navigability notation indicates that the association only can be used in one direction.

An association may be used to indicate that requirement B *uses* requirement A; requirement A is a standalone requirement and does not need to be aware of requirement B. It is up to requirement B to obey changes in requirement A. An example is given in Figure 8

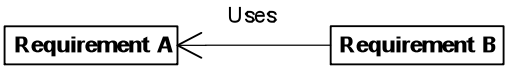



**Figure 8: Requirement Association**

Requirement aggregation is "a special form of association that specifies a whole-part relationship between the aggregate (whole) and a component part." [20] In requirement traceability aggregation is used when a requirement B is a part of a "bigger" requirement A which encloses requirement B.


**Figure 9: Requirement Aggregation**

Requirement composition is "a form of aggregation which requires that a part instance be included in at most one composite at a time, and that the composite object is responsible for the creation and destruction of the parts. Composition may be recursive." [20]. A composite relationship means the associated requirement is a part of the parent requirement (as in aggregation); in contrast to aggregation, the child requirement cannot exist alone. For example, a slider scrollbar in a window on a computer screen, or a panel of the window, cannot exist without the window being constructed first. If the parent (window) is destructed, the scrollbar and the panel are also destructed. Aggregation is visualised with the solid diamond symbol.


**Figure 10: Requirement Composition**

Generalization is "A taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier indirectly has features of the more general classifier.". A generalisation relationship means that the associated requirement inherits all features (attributes, description, etc.) from the parent requirement. Generalisation is visualised with a hollow arrow.

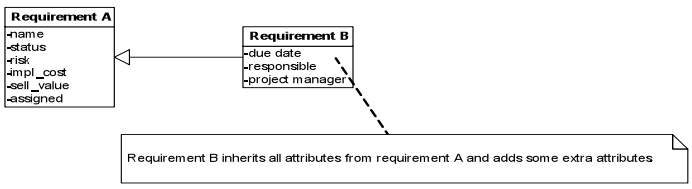



**Figure 11: Requirement Generalisation**

Generally traces (i.e. *trace to* and *trace from*) is visualised using the *UML Dependency* notation, which is defined as follows: "A relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element". A trace is modelled as a dashed line with an arrow. A stereotype keyword («trace», «usage», «import», etc.) is normally shown on the line to indicate the type of dependency. Each dependency is a traceability link. By using the stereotype facility we can have any type of new traceability links. Stereotypes of interest are:

- **Trace:** A dependency that indicates a historical or process relationship between two elements that represent the same concept without specific rules for deriving one from the other.
- **Usage Traceability:** A dependency in which one element (the client) requires the presence of another element (the supplier) for its correct functioning or implementation.
- **Import Traceability:** In the context of packages a dependency that shows the packages whose requirement classes may be referenced by another given package. For example, we have requirement classes in Feature Package A and Feature Package B.

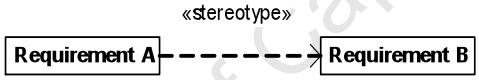



**Figure 12*:* Dependency**



**Requirement Patterns**

Much research and discussion continues about. Patterns describe best practices; good designs and capture successful work experiences [2]. Christopher Alexander, offered an instructive definition of patterns: "Each pattern describes a problem that occurs over and over in our environment and then describes the core of the solution to that problem in such a way that you can use this solution a millions times over without ever doing it the same way twice [1]." There are now pattern categories for all parts of the development process.

Traditional requirement patterns, are used to help specify user requirements. Alistair Cockburn in his book "Patterns for Effective Use Cases" describes a catalog of a few dozen use case patterns offering criteria for evaluating the quality of use cases. Each pattern describes a specific guideline or "sign of quality" that you can use to judge the caliber of a use case in a particular area. These patterns are diagnostic rather than proactive requirement recognition patterns for reuse. They are a quality reviewing tool rather than a tool for domain specific requirement recognition or traceability recognition.

We believe that the power of the requirement pattern can be found in the true design pattern definition. [10] A class acts as a template for objects. A requirement pattern is a template or

set of rules for reuse requirements and finding commonality in the representation of the traceability links. A requirement pattern describes at an abstract level the interaction between requirement classes, objects, and communication flow. Requirement patterns emerge from our representation of requirements as class. For example, the structural use case class patterns describe the basic components of use case classes, how they should be organized, and offer criteria for their use. Using the UML relationships we can also recognise traceability patterns from the relationships between the requirement classes. For example, the "Log-on-System" traces to the functional metaclass and a basic traceability pattern emerges that can be reused at the start of every project. Other traceability patterns described by means of software design constructs, the inheritance, aggregation and uses-relationship between certain requirement types.

A requirement metamodel provides the generic structure and behavior for a family of requirement modeling elements, along with a context of metaphors that specifies their collaboration and use within the requirement domain. A requirement metamodel framework is a set of cooperating requirement classes that make up a reusable design for a specific class of software. A requirement framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A requirement metamodel is a large requirement pattern. It is the combination and hybridization of a series of requirement patterns. It is a true requirement pattern language as it is implementation of a system of requirement patterns. Despite the fact that they are related in this manner, it is important to recognize that requirement frameworks and requirement patterns are two distinctly separate entities: a requirement framework is executable software, whereas requirement patterns represent knowledge and experience about requirements. In this respect, requirement frameworks are of a physical nature, while requirement patterns are of a logical nature: requirement frameworks are the physical realization of one or more requirement pattern solutions; patterns are the instructions for how to implement those solutions.



**Requirement Process Framework**

In Fig 13 we briefly illustrate the Process Framework that we are developed on this project. It describes our approach to use case class modeling, explicit traceability links and the requirement modeling and traceability pattern recognition. We are currently validating the below framework with a small project management company Coras Systems [26]. The initial results suggest that all requirements can be represented with classes. The requirement metamodel is showing good results for requirement reuse and pattern recognition.

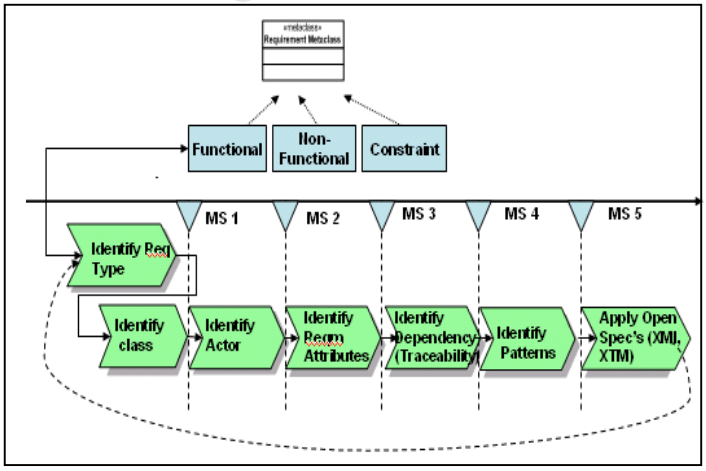



**Figure 13 Requirement Process Framework**

The main activities are: identify the requirement types, identify the actors, identify the attributes and operations, identify the traceability dependencies, recognize requirement and traceability patterns, apply open specification standards to the models, for example XMI, XML or XTM (Topic Maps) The class models can be converted into XML. One of the experiments we completed was the conversion from UML to XML to XTM (Topic Maps). The ISO standard ISO 13250:2003 is the international standard for Topic Maps. They enable multiple, concurrent views of sets m*etadata* or in this case information related to requirement classes and traceability links. We created round-trip engineering from UML to XMI to CSV to XTM and vice versa. We represented our requirement metamodel and patterns in UML. We converted the UML to XMI. Transforming the XMI into CSV we imported the CSV format into a requirement management tool (Rational RequisitePro) and reviewed the resulting traceability matrix. We finally converted the XMI to XTM visualizing the results topic maps. Fig 14 below shows a Topic Map view we created using an open source XTM tool.

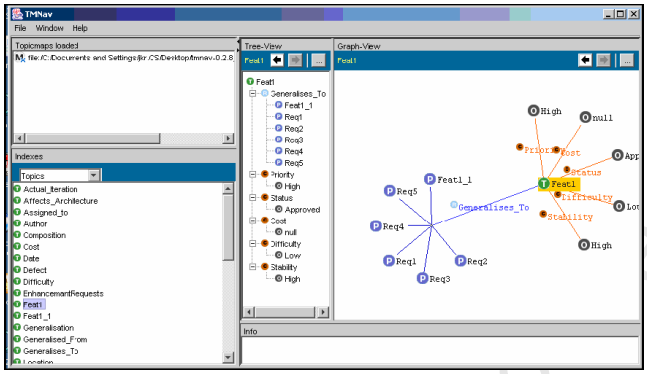



**Figure 14 Requirement Classes Represented as Topics**

The requirement process framework is an input into the metamodel layer and the requirement model layer. Changes to one metaclass or class may impact the entire requirement metamodel, model and the process framework addresses this change. We are currently reassessing the process so that it manages the User and System Requirements from the perspective of the owners and stakeholders of the system, while at the same time being sufficiently rigorous to enable the process to be used by developers. Our initial results are showing better coupling between the requirements and design model elements with easier movement of information and data between the disciplines.

## Conclusion

In this paper we review some key concepts in requirement modelling and traceability. The creation of a layered requirement metamodel was presented. The layered has a requirement meta-metamodel or MOF at its highest level of abstraction. In many situations the MOF layer can be ignored. The requirement metamodel layer is next while the product and project requirement models at the lowest two layers.

Changes to UML 2.0 support the use of use case classes. The rationale for instantiating use cases as classes is primarily for traceability links for testing source code. A requirement

metamodel provides the generic structure and behavior for a family of requirement modeling elements, along with a context of metaphors that specifies their collaboration and use within the requirement domain. Its is a collection of requirement patterns or a requirement pattern language. Requirement patterns also communicate best practices, which are experience based and which illustrate a loosely defined practice in a uniform way. Our approach offers the advantage for traceability practices, plus some operation guidance in design.

Finally we presented a requirement process framework based on an ongoing research which supports the modeling of requirements and traceability using the UML use case classes, UML extension mechanisms and the requirement patterns.

## REFERENCES


[1] Alexander C., A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977.

[2] Appleton B., "Patterns and Software: Essential Concepts and Terminology" http://www.enteract.com/-bradapp/docs, Object Magazine Online (vol.3, nro.5), May 1997.

[3]Ambler, S Tracing your design. Software DevelopmentMagazine http://www.sdmagazine.com/, P 48-54. April 1999.

[4] Beck, K. Coplien, J. O., Crocker, R., Dominick, L., Meszaros, G., Paulisch, F., and Vlissides, J. (1996). Industrial Experience with Design Patterns, Proceedings of ICSE '96, Berlin, pages 103-114, March 1996.

[5] Booch, G. Rumbaugh, J, Jacobson, I., The Unified Modeling Language user guide, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, 1999

[6]Domges Ralf and Klaus Pohl. "Adapting traceability environments to project-specific needs" Communications of the ACM,41( 12):54–62, December 1998.

[7]Edwards M. and S. Howell. "A Methodology for System Requirements Specification and Traceability for Large Real-Time Complex Systems" Technical report, U.S. Naval Surface Warfare CenterDahlgren Division, Dahlgren, Va., 1991.

[8]Egyed A., "A scenario-driven approach to traceability" Proceedings of the 23rd International Conference on Software Engineering, Pages: 123 – 132, Year of Publication: 2001

[9] Fabricius Frank, Chris Pang ,Duggan Jim. Latimer Nicole S, Gartner Group "Application Development Software Trends in Europe, 2002-2007" 5 January 2004.

[10] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., Design Patterns. Addison-Wesley, 1995.

[11] Gotel, O. C. Z. and Finkelstein, A. C. W.: "An Analysis of the Requirements Traceability Problem," Proceedings of the First International Conference on Requirements Engineering, 1994, pp.94-101.

[12] Jacobson, I. M. Ericsson, and A. Jacobson, *The Object Advantage: Business Process Reengineering With Object Technologyt,* Addison-Wesley, Reading, Massachusetts, 1995

[13] Par Jansson. Use Case Analysis with Rational Rose. Rational Software Corp, Santa Clara CA, 1995.

[14] Kholkar, Deepali G. Murali Krishna, Ulka Shrotri, R. Venkatesh "Visual specification and analysis of use cases" 2005 ACM symposium on Software Visualization Pages: 77 – 85

[15] Knethen A., "Automatic Change Support based on a Trace Model" In Proc. of 1st International Workshop on Traceability in Emerging Forms of Software Engineering. In conjunction with the 17th IEEE International Conference on Automated Software Engineering, Edinburgh, U.K., September 2002 – 2002

[16] Kruchten, P. The Rational Unified Process: An Introduction, Second Edition, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2000

[17] Letelier P., "A Framework for Requirements Traceability in UML-based Projects" In Proc. of 1st International Workshop on Traceability in Emerging Forms of Software Engineering. In conjunction with the 17th IEEE International Conference on Automated Software Engineering, Edinburgh, U.K., September 2002 – 2002

[18] Marco Leon, Intelligent Enterprise Magazine, July 17, 2000, Vol 3 Number 12

[19] OMG (2003) UML 2.0 Infrastructure Specification [Online] http://www.omg.org/cgi-bin/apps/doc?ptc/03-09-15.pdf [Accessed 30th September 2005]

[20] OMG (2003) UML 2.0 Supersctructure Specification [Online] http://www.omg.org/cgi-bin/apps/doc?ptc/03-09-15.pdf

[21] OMG, (1998), Booch, G., Rumbaugh, J., Jacobson, I., The Unified Modeling Language User Guide, Addison-Wesley, Massachusetts, 1998.

[22] Shrotri (U), Bhaduri (P) and Venkatesh (R). Model-checking visual specification of requirements. International Conference on Software Engineering and Formal Methods. Brisbane, Australia: IEEE Computer Society Press, September 2003. pp 202–209

[23]Spence I., Probasco L., "Traceability Strategies for Managing Requirements with Use Cases" Rational Software White Paper, 2000

[24] Zhang, J. "Metamodel-Driven Model Interpreter Evolution" In 2004, Conference on Object Oriented Programming Systems Languages and Applications Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications

[25] Zisman A., Spanoudakis G., Perez-Minana E., Krause P., "Towards a Traceability Approach for Product Families Requirements", Proceedings of 3rd ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications, Orlando, USA, May 2002

[26] Coras Systems www.corasystems.com/