A Framework for the Analysis and Evaluation of Software Development Methodologies Based on Formal, Intrinsic and Pragmatic Criteria

Dominic Damon Riordan

Thesis presented to the University of Cape Town Department of Information Systems, January 2007, in partial fulfilment of the requirements for the course Masters in Information Systems, INF500W.

Abstract

At present, there are a substantial number of software development methodologies in use. This research proposes a framework, based on Van Belle's (2003) framework for the analysis of Enterprise Models, to analyse, evaluate and compare software development methodologies using a set of criteria based on three dimensions (formal, intrinsic and pragmatic). The formal dimension is concerned with structural aspects of methodology process models (formal) such as methodology process model size, complexity and modularity. The intrinsic view is concerned with the inherent meaning of methodologies as described in their reference material and is composed of criteria such as methodology support for project management and verification and validation techniques. Finally, the pragmatic view is concerned with the contextual aspects of methodologies, or aspects which cannot be assessed purely on the information contained within the reference material, such as methodology situationality and flexibility. A thorough review of the relevant literature is provided to describe the formation of the proposed framework. The theory and context of methodologies is also discussed, along with the reasons for and against their use and the main methodological paradigms which form the basis of most modern software development methodologies. The proposed framework is validated using a number of popular and well-known software development methodologies, specifically: Structured Systems Analysis and Design Methodology (SSADM), Object-Oriented Analysis and Design (OOAD), the Microsoft Solutions Framework (MSF), the Rational Unified Process (RUP), Extreme Programming (XP) and Feature-Driven Development (FDD). The results of the analysis supported the proposed framework with the majority of the criteria being both applicable to the domain and the measures for the criteria being satisfactory and valid.

Declaration

I declare that this research is my own unaided work. It is submitted in partial fulfilment of the degree of Master of Commerce in the University of Cape Town. It has not been submitted before for any other degree or examination in any other university.

Signed

Dominic Damon Riordan

On this the day of , 2007.

Acknowledgements

Thanks to my supervisor for this project, Prof JP van Belle, for his advice, encouragement and constructive criticism.

Thanks also to Zoe, Rory, Jonathan and Tristam Riordan for their love, advice and encouragement.

Thanks are also due to Elizabeth Kelly, for her love and support (and proofreading), and my adoptive Cape Town family (Greg and Sash Streatfield and Rolf Werndle) for putting up with me.

Finally, thanks to my employers, BSG (Africa), and co-workers for their support and encouragement.

Table of Contents

Index of Tables

Index of Figures

Chapter 1: Introduction

Historical Perspective

Software development methodologies became part of the software development process in the 1960s and 1970s as software developers became increasingly aware of the growth of software development as a discipline and realised that a level of regulations and guidelines would be useful (Kiely and Fitzgerald, 2003, p. 188). However, it is only in the last 20 years or so that significant interest has been expressed in software development methodologies (Probert, 2001, pp. 437-443).

The result of this interest is that there are currently a substantial number of methodologies available for use. Before 1990, there were few methodologies available, but this ballooned to an estimated 1000 name-brand methodologies available by 1994 (Alhir, 1998, p. 10; Jayaratna, 1994, p. 44). Of course, some methodologies enjoy substantially wider use than others but offer very little in the way of standardisation, which is clearly evidenced by the difference in syntactic descriptions – they are alternately defined as methodologies (Jayaratna, 1994, p. 35), methods (Goodland and Slater, 1995), software engineering methods (Sommerville, 2004, p. 11), software development processes (Booch, 1995 in Krutchen, 1999, p. 15) and software engineering processes (Krutchen, 1999), all of which refer to the same type of entity referred to as software development methodology, or simply methodology, in this research.

Berki and Georgiadou (1998, p. 1) recognise the importance of selecting the right software development methodology by stating that the choice of an appropriate methodology has become one of the most critical issues in software development, as the quality of a software system "largely depends on the process model for its construction". Booch (1994, p. 23), on the other hand, argues that there is "no magic, no 'silver bullet' that can unfailingly lead the software engineer down the path from requirements to the implementation of a complex software system" and states that there is no "'best' method", and no best way to decompose a complex system. This sentiment is echoed by Korac-Boisvert and Kouzmin (1995, in Kiely and Fitzgerald, 2003, p. 190) who opine that not all Information Systems (IS) development projects are amenable to conventional, or structured methodologies. To choose a methodology for a software development project is, according to Berki and Georgiadou (1998, p. 1), a "very difficult task" as a result of the differences in existing frameworks, which vary in areas such as "philosophy, models [and] products".

Previous research based on explicitly comparing methodologies has, almost without fail, focused on the main methodological paradigm in use at that particular point in time. For example, much of the earlier work, such as Burns and Dennis (1985) and Mannino (1987), focuses on the structured methodologies, while research performed in the early- to mid-1990s, such as Hong *et al.*, 1993; and The Object Agency, 1995, tends to focus on the Object-Oriented paradigm. Later research, for example that performed by Abrahamsson *et al.* (2003), focuses on the Agile paradigm, which has grown in use in the last eight or ten years. One aspect of methodological evaluation and comparison

that seems to be universally absent from the previous work is that of comparing and evaluating methodologies across different paradigms.

While both Burns and Dennis (1985) and Mannino (1987) compared the prototyping methodology to structured methodologies, the prototyping methodology does not, *per se*, belong to a specific paradigm. It is considered a forerunner to Boehm's Spiral approach and thus the iterative and Agile methodologies and it is now considered as a technique and used by a number of methodologies (Abrahamsson *et al.,* 2003, p. 246, Boggs, 2004, p. 37). As a result, one of the motivations of this research is to compare more than one methodology from each of the major paradigms (Systems Development Life Cycle, Object Oriented and Agile) against one another.

A second motivation for this research stems from the approach with which much of the previous research approached the analysis, comparison and evaluation of methodologies. Most of the previous work concentrates on intrinsic or semantic criteria, for example a methodology's level of coverage of the software development life cycle (Van Den Bosch *et al.*, 1982; Mannino, 1987; Abrahamsson *et al.*, 2003). While some research, such as that performed by Hong *et al.* (1993), took formal, or syntactic, criteria into account through building meta-models of methodologies and comparing that data, and some research, such as Kelly (1987) and TOA (1995), took pragmatic criteria, like supporting resources into account, no research has taken criteria from all three viewpoints into account.

As a result, no single approach for the comparison and evaluation of methodologies, with the exception of TOA (1995), which was intended only for the evaluation and comparison of Object-Oriented methodologies, took formal, intrinsic and pragmatic criteria into account. As such, and in order to attempt to take criteria encompassing these evaluation views into account, the framework proposed by Van Belle (2003) for the analysis, evaluation and comparison of enterprise models has been adapted for use in this research.

A number of detailed and respected frameworks for the analysis, evaluation and comparison of methodologies have been proposed, most notably the NIMSAD (Normative Information Model-based Systems Analysis and Design) framework (Jayaratna, 1994) and that proposed by Avison and Fitzgerald (1995). The NIMSAD framework takes a somewhat different approach, aiming to understand the methodology context, methodology users and the methodology itself, while the framework proposed by Avison and Fitzgerald (1995) is based on aspects such as methodological philosophy, model, techniques and tools and practice. Interestingly, McLeod and Roeleveld (2002) use the NIMSAD methodology to measure methodological effectiveness, as well as the Chart of Concepts (Castellani, 1998) and Method Points method (McLeod, 1997) to measure methodology complexity, and Krogstie *et al.*'s (2000) quality assessment framework to measure modeling capability.

Other popular frameworks include that proposed by Singh and Kotze (2003), whose approach focuses on the human component of the methodologies evaluated and compared, and Sorenson (1995), which focuses on the perceived strengths and weaknesses of methodologies.

The framework proposed by Moody (2001) uses the Technology Acceptance Model (TAM) to investigate how variables such as actual and perceived effectiveness and efficiency affect use of a methodology.

This research aims to provide a framework for the evaluation of a number of different software development methodologies, which are disparate in terms of approach and philosophy. A secondary aim of this research is the validation of the framework through the analysis and evaluation of a set of software development methodologies.

The origin and composition of this framework will be discussed in Chapters 2 to 4, and the actioning of the framework and evaluation of the methodologies will be provided in chapters 5 to 7. Substantial previous work has been done in the field of methodology comparison and evaluation, as evidenced by the volume of work presented in the literature survey. The body of work surveyed in the literature survey encompasses some of the research performed in this field in the last twenty years, and this is shown in the subject matter of this research, as it covers the major methodological paradigms.

Purpose and Scope

The purpose of this research is threefold. Firstly, there is an attempt to determine and consolidate the criteria considered most relevant for the analysis and evaluation of software development methodologies, suggested and defined in a wide range of research. This includes research into methodologies, methodology evaluation, method engineering and enterprise model evaluation, amongst others, and the intention is to extract the most relevant and valuable criteria. The second purpose of this research is to provide a framework for use in the analysis and evaluation of methodologies. The intention of this framework is to use techniques and tools that have already been used in the analysis and evaluation of methodologies, such as metamodeling, the NIMSAD framework (Jayaratna, 1994) and the Zachman Framework for Information Systems Architecture (Zachman, 1987), but to organise them based on the criteria defined above using Van Belle's framework for the evaluation of enterprise models (2003) as the structuring mechanism.

The final purpose of this research is to validate the suggested framework by actioning it using a number of well-known methodologies currently in use and easily available, via their reference material. Instead of using proprietary or obscure methodologies, the intention is to provide the validation of the framework in as transparent a manner as possible. The scope of this research is limited to methodologies with easily and publicly available reference material. Derived or adapted methodologies where no reference materials are available are probably beyond the scope of an intrinsic analysis.

Scientific Contribution

The chief scientific contribution of this research is the provision of a significantly more detailed and comprehensive framework for the analysis and evaluation of methodologies than is currently available. This will be achieved through the synthesis of many traditionally-used methodology evaluation techniques, which often tend to be used individually.

As such, the proposed framework provides three different views of the methodologies being evaluated, specifically the:

- *Formal*, which is derived from mapping the processes that underpin the methodologies being analysed to a metamodel, and evaluating the results.
- *Intrinsic*, which is derived from the content of the methodology reference material.
- *Pragmatic*, which is derived from contextual and situational aspects, beyond the scope of the reference material.

Given that these different views are derived using a number of criteria, each of which use specific methods of analysis, the proposed framework offers a more powerful method of evaluation than current frameworks. It is hoped that a second contribution of this research is a framework that is both extensible to future users and actionable in a non-academic context. An attempt has been made to ensure that the framework is as modular as possible to ensure that future users could use anything from an individual criteria, to a single view (such as the *Intrinsic* view) derived from the framework.

Structure of this Research

The remainder of this research is structured as follows:

- *Chapter Two* details the theory and discipline of software development methodologies, clarifies the syntax that will be used throughout this research, discusses the reasons for and against the use of methodologies and describes some of the main methodological paradigms and methodologies currently in use.
- *Chapter Three* contains the literature survey and provides a summary of relevant literature, mainly relating to the discipline of methodology evaluation and comparison.
- *Chapter Four* describes the research methodology and the structuring method and framework to be used for the methodology analysis and evaluation.
- Chapters Five to Seven detail how the framework was operationalised and validated, in terms of the formal, intrinsic and pragmatic views respectively.
- *Chapter Eight* presents the conclusions of the research, a short discussion of the philosophical basis of the research and some suggestions for future research.

The research concludes with a comprehensive list of references, and a number of appendices detailing aspects of the research that were considered too bulky or detailed to be included in the text of the research.

Chapter 2: The Theory and Discipline of Software Development Methodologies

This chapter provides definitions of aspects of software development methodologies to ensure that a common syntax will be used throughout this research. Thereafter, the theory and attributes of methodologies will be described, the reasons for and against the use of methodologies will be discussed, and brief overview of the important methodological paradigms that are being used will be provided. The intention of this chapter is to 'set the scene' for the research as the criteria used in the proposed framework focuses on the evaluation of their attributes, and methodologies form the basis of the data used in the actioning of the framework.

Definitions

Firstly, a number of terms should be defined in the context of software development projects, namely **paradigm**, **method**, **process**, **methodology** and **notation**.

Alhir (1998, p. 35) defines a **paradigm** as "an organised, self-contained collection of related components that form the foundation of languages and methods and define a set of concepts forming ideas or notations associated with subjects". An example of a paradigm, in the context of this research, is the object-oriented paradigm, which underpins a large number of methodologies such as OOAD and the RUP.

Jacobson (1992, p. 3) states that a **method** provides the explicit step-by-step procedures to be followed in applying the necessary concepts and techniques, while Alhir (1998, p. 17) defines a **method** as a description of how to conduct problem solving efforts by specifying an overall problem-solving approach and its components, as well as specifying how problems and solutions are viewed in relation to a problem-solving approach. Jacobson (1992, p. 3) adds to this definition, stating that a **method** provides the explicit step-by-step procedures to be followed in applying the necessary concepts and techniques.

A **process** is, according to Alhir (1998, p. 17), the realisation of a method by using the method's problem-solving approach to solve a problem and realise a solution. Jacobson (1992, p. 3) states that a **process** allows the **method** to be scaled, so that it can be applied to projects with many interacting activities and parties. Methodology processes are discussed in more detail in Chapter Five.

On the other hand, a **methodology** is defined as a taxonomy, or well-organised, collection of related methods (Alhir. 1998, p. 17), while Sommerville (2004, p. 11) believes that a **methodology** is a structured approach with the aim of facilitating "the production of high-quality software in a cost-effective way". Jayaratna (1994, p. 37) defines a **methodology** as an explicit way of structuring one's thinking and actions, which contains models and reflects particular perspectives of reality based on a set of philosophical paradigms. Incidentally, Jayaratna (1994, p. 35) argues that, in the field of information systems at least, the term 'methodology' means the same as the term 'method'. Nance and Arthur (1988, p. 221) define a **methodology** as a collection of complementary methods and a set

of rules for applying them. An example of a software development methodology is the Rational Unified Process. Finally, Booch (1994, p. 23) defines a **notation** as a language for expressing a model. An example of a notation is the Unified Modeling Language.

Systems development is defined by Jayaratna (1994, p. 35) as a discipline concerned with the development of information processing systems and the definition of the general set of activities that need to be performed in order to produce these systems, but without agreement, even at a general level, as to what these activities should be. This differs from both software development/engineering[1], which is concerned only with the activities of software construction and quality assurance and does not take organisational issues into account, and methodologies, whose steps and activities are explicitly stated and ordered (Jayaratna, 1994, p. 34).

The Theory of Methodologies

The importance of a software development methodology lies in its ability to solve larger, "real world" problems and that while small problems can be solved through the development of small programs or models in an *ad hoc* manner with little discipline and few supportive techniques, larger problems cannot. As a result, methodologies provide the necessary structure and supportive techniques to solve these problems by fulfilling two roles; providing a conceptual contribution to understanding the development task, be it software or model oriented, and by being a practical design guide and serving as a "blueprint for development environment tools" (Nance and Arthur, 1998, pp. 220, 221).

A methodology should consist of phases, which in turn consist of sub-phases, each of which guide the system's developers in their choice of the techniques that might be appropriate at each stage of the project and also help them plan, manage, control and evaluate information systems development projects. A methodology should also have objectives, including (Avison and Fitzgerald, 1995, pp. 11-14):

- Specifying the system requirements accurately.
- Enforcing systematic system development, thereby enabling project monitoring and control.
- Providing a system at acceptable cost, within a reasonable time frame.
- Minimising the impact of changes to requirements during the development process.
- Documenting the system and allowing for future maintenance.
- Providing a system that ultimately adds value to the various stakeholders.

A software development methodology should organise and structure the tasks comprising the effort to achieve global objectives in a software development project. It should also include methods and techniques for accomplishing individual tasks within the framework of the global objectives, as well as prescribe an order in which certain classes of decisions are made, and the ways that make those decisions lead to the global objectives (Nance and Arthur, 1988, p. 221).

1 Terms normally used interchangeably to refer to software construction (Jayaratna, 1994, p. 34).

The Context of and Need for Methodologies

Avison and Fitzgerald (1995, pp. 8, 9) state that information systems have existed for many years, but that it is only in the recent past that they have involved computers and applications. Correspondingly, the need for methodologies to manage the software development process has risen sharply. These early computer applications were mostly implemented *ad hoc*, or without an explicit methodology and the result of this *ad hoc* implementation of software was a lack of applicability of the software to the requirements of the users. As managers demanded more appropriate systems for their outlay, three main changes were made, specifically (Avison and Fitzgerald, 1995, pp. 8–10):

- A growing appreciation of the part of the software development that concerned analysis and design, and thus the role of the systems analyst.
- The realisation that, as organisations were growing in size and complexity, more integrated software solutions were required to meet their needs.
- An appreciation of the desirability of accepted methodologies for the development of systems.

As a result of these changes, structured methodologies such as SSADM were developed and used (Avison and Fitzgerald, 1995, p. 20). Methodologies are adopted for use for a variety of reasons according to Alhir (1998, pp.9-10), Fitzgerald (1996, p. 5) and Jayaratna (1994, p. 227). From a management and organisational perspective, methodologies both facilitate project management, minimising risk and uncertainty, and provide a purposeful framework for the application of techniques and resources, allowing skill specialisation and the division of labour, which in turn provides economic benefits. The adoption of methodologies also allows for the standardisation of the development process, which can subsequently facilitate interchangeability among project team members and increase productivity and quality. Methodologies can also reduce the complexity of the process of software development in that they provide:

- A reductionist subdivision of the complex process, on a 'divide and conquer' principle, in the number and dimensions of a set of elements that the methodology extracts from the 'action world' for modeling and manipulation.
- A structural framework for the acquisition and systematisation of knowledge and a structure which provides a variety- and complexity-reducing mechanism, thus providing an epistemological benefit to the users of the framework.

More practical reasons for the adoption of methodologies for software development are that they allow organisations to be eligible for certifications such as ISO and Software Capability Evaluation (SCE), both of which are viewed as highly prestigious and a competitive advantage. The use of standardised methodologies also allows organisations to be eligible for government contract work, where the use of specific methodologies (such as SSADM in the UK, Ireland, Hong-Kong, Malta and Israel, and Department of Defence Std. 2167 in the USA) is often mandated.

A final reason for the adoption of a software development methodology is that they can provide "an escape from the literature bias of irrational or unstructured practitioners being the reason for problems in system development" (Fitzgerald, 1996, p. 5).

An Overview of the Important Paradigms

A number of important paradigms relating to system and software development have arisen over the last 30 or 40 years. This section explains the so-called structured paradigm, based heavily on the traditional systems development life cycle, the object-oriented (OO) paradigm, and the agile paradigm. Each of these paradigms correspond to Alhir's (1998, p. 35) definition of a paradigm in that they consist of a collection of related components that form the foundation of methods (and methodologies) and define a set of concepts forming ideas or notations associated with subjects. They have resulted in a multitude of proposed methodologies, based on their principles and concepts. This section discusses the paradigms, and gives a very brief overview of some of the more popular methodologies associated with them.

The Structured Paradigm

The structured, or traditional, paradigm was developed and adopted as a result of a lack of a methodological base for systems and software development, and became the *de facto* paradigm for systems and software development for many years (McNurlin and Sprague, 1993, p. 261). A life cycle-based approach was initially described by Canning in 1956, but it was only in 1970 that the Software Development Life Cycle was proposed by Royce, which formed the basis of a large number of widely-used methodologies such as Jackson Systems Development (JSD), the Structured Systems Analysis and Design Method (SSADM) and the Yourdon Systems Method (YSM) (Avison and Fitzgerald, 1995, p. 20, Boggs, 2004, p. 37, McNurlin and Sprague, 1993, p. 261). The structured paradigm is based on a number of sequential steps often referred to as the waterfall model because these steps are based on the premise that for a task to start, the previous one must have finished completely (Pressman, 2005, p. 79). Pressman (2005, p. 79) describes it as "a systematic, sequential approach". The steps are detailed in *Table 1*, while Enger (1981, pp. 1-24) describes the Systems Development Life Cycle (SDLC) phases in the following diagram (*Figure 1*):

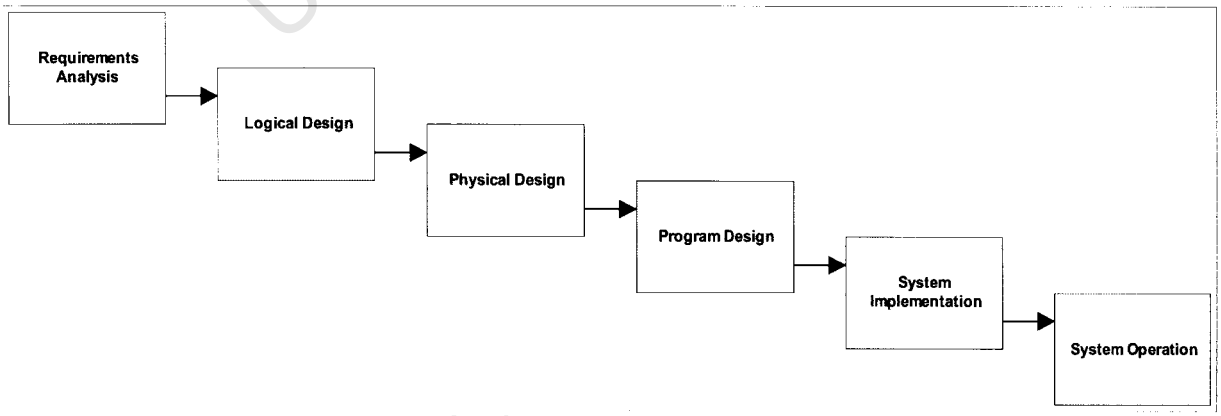

Figure 1: SDLC Phases (Enger, 1981, pp. 1-24)

The following strengths of the structured paradigm have been identified by Pressman (2005, p. 80) and Burns and Dennis (1985, p. 21):

- The structured paradigm forms the basis of many well tried and trusted methodologies.
- The use of phases ensures that there is greater control over the systems development process than there would be if a more *ad hoc* approach was used.
- The use of documentation and communication standards can ensure that specifications are complete and communicated to the stakeholders timeously.
- It has been proven to work very well on projects with a high degree of complexity and a low degree of uncertainty.
- Reviews are part of each phase and, as a result, there is a level of learning during each project.

Phase	***Description***
Requirements Analysis	This phase is concerned with evaluating user requests, conducting a feasibility study, defining the user requirements for the future system and preparing a project plan.
Logical Design	This phase is concerned with preparing the general design specifications and refining the user system requirements.
Physical Design	This phase is concerned with preparing detailed design specifications, defining subsystems and designing the future system's database structure.
Program Design	This phase is concerned with coding, unit testing and documenting the future system.
System Implementation	This phase is concerned with performing sub- and system testing, training the user personnel, establishing conversion controls and performing data conversion.
System Operation	This phase is concerned with operating, maintaining and evaluating the new system.

Table 1: The phases of the SDLC, from Enger (1981, pp. 1-24).

Both Avison and Fitzgerald (1995, pp. 30, 31), Burns and Dennis (1985, p. 20) and Pressman (2005, pp. 79, 80) point to some weaknesses in the structured paradigm:

- The structured paradigm does not handle change well. This is not effective in projects with a high degree of uncertainty, because it requires that each phase be complete before the next one begins. As a result, any changes deemed necessary in the requirements, if recognised during the construction phase, can be very difficult to implement.
- The structured paradigm often fails to meet the requirements of management in terms of information needs, as the SDLC focuses on low-level operational tasks.
- The structured paradigm often results in unambitious system design, as it focuses on replacing systems rather than re-engineering them.

- Real projects rarely follow the sequential flow that the model provides.
- A working version of the system is only available late in the project life span and, as a result, the customer will not see any benefit until the deployment phase.

The Object-Oriented Paradigm

The object-oriented (OO) paradigm is based on the use of objects for software development, a term which "emerged almost independently in various fields in the computer science, almost simultaneously, in the early 1970s to refer to notations that were different in their appearance yet mutually related" and was invented to manage the complexity of software systems in such a way that the objects represented components of a modularly decomposed system, or modular units of knowledge (Booch, 1994, p. 36). The OO paradigm was a revolutionary change in software development in the late 1980s, and it became a mainstay in application development due to its suitability for graphical user interface (GUI) and client-server systems (McNurlin and Sprague, 1993, pp. 272, 273).

The OO paradigm is based on three activities: analysis, design and programming, defined as follows (Booch, 1994, pp. 38, 39):

- OO Analysis: A method of analysis that examines requirements from the perspective of the classes and objects found in the problem domain.
- OO Design: A method of design encompassing the process of OO decomposition and a notation for depicting both logical and physical, as well as static and dynamic models of the system under design.
- OO Programming: A method of implementation in which programs are organised as cooperative collections objects, each of which represents an instance of a class, and whose classes are all members of a hierarchy of classes united via inheritance relationships.

Alhir (1998, p. 10) states that the early 1990s saw a huge growth in the use of OO techniques and methods, with a jump from less than ten OO methodologies in 1989 to more than fifty in 1994, many of which were using their own notations. As a result of this growth, the Unified Modeling Language (UML) was adopted by the Object Modeling Group (OMG) in 1997 as the standard notation for OO methodologies. Both Booch (1994, p. 249) and de Champeaux, Lea and Faure (1993, p. 6) state that the Object Oriented paradigm comprises three main aspects: a notation, a process, and pragmatics. Jacobson (1992, p. 113) and Booch (1994, p. 249) describe the OO process as a model-driven process, where five different high-level phases are used:

- *Conceptualisation*, where the core requirements are gathered, resulting in the requirements model, which aims to capture the functional requirements.
- *Analysis* (described above).
- *Design* (described above).

- *Evolution*, in which the implementation is evolved. This phase results in the implementation and test models, which implement and verify the system.
- *Maintenance*, where post-delivery evolution is managed.

Pragmatic aspects, such as project management and planning, staffing, release management, reuse, quality assurance and documentation, run as parallel processes to the phases described above. The actual software development process can be shown as follows (from Jacobson, 1992, p. 33) in *Figure 2*. This process is often iterative, with the Analysis, Construction, Testing and Components (which involves the reuse of components, rather than the construction of them) phases heavily interlinked in a cyclical relationship until the system is complete (Jacobson, 1992, p. 33).

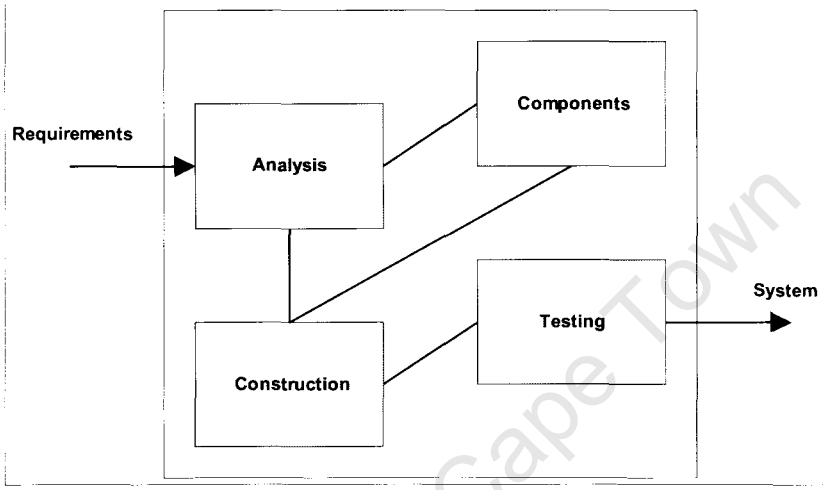

Figure 2: The OOAD process (Jacobson, 1992, p. 33)

The OO paradigm differs from the structured paradigm in that (McNurlin and Sprague, 1993, p. 273):

- It involves the development of a system based on a model and a simulation of the business. This is conceptually different from the procedural view, which involves the separation of data and processes.
- The project team and users communicate with one another using business terms, as opposed to technical terms, as these form the basis of the system objects.
- Application code and data is not separated as is the case with applications developed using the procedural paradigm.
- Data is an active aspect of the application, as opposed to the structured paradigm where data is passive, as the objects require an awareness of the data in order to perform work on themselves.
- Component reuse is encouraged and heavily used, which is not the case in the structured paradigm.

In a survey conducted by Johnson (2000, p. 69), 150 "seasoned" software developers were polled regarding their views on OO software development. The results suggested that "both OO and non-OO developers view OOSD as superior, that OO developers hold this view more strongly than non-OO developers, and that all developers view the reported disadvantages of OOSD as virtually non-

existent" (Johnson, 2000, p. 69). However, Johnson (2000, pp. 71, 72) does state that concerns have been raised about a number of aspects of OO methodologies considered disadvantages, specifically regarding its complexity, immaturity, the lack of OO database management systems, the preponderance of different OO methodologies, and the increased initial development time.

Some of the most significant methodologies based on the OO paradigm are Object-Oriented Analysis and Design (OOAD), Object Oriented Software Engineering (OOSE), and the Rational Unified Process (Alhir, 1998, p. 11). OOAD was created by Grady Booch, and OOSE by Ivor Jacobson in 1992, and is based on Objectory (also developed by Jacobson), the first OO methodology. It features many of the concepts and principles detailed above. In introducing OOSE, Jacobson (1992, p. x) states that the benefits of using OO techniques can only be gained by the consistent use of OO throughout all steps in the development process. The OOAD process and phases is shown in the figure below (*Figure 3*):

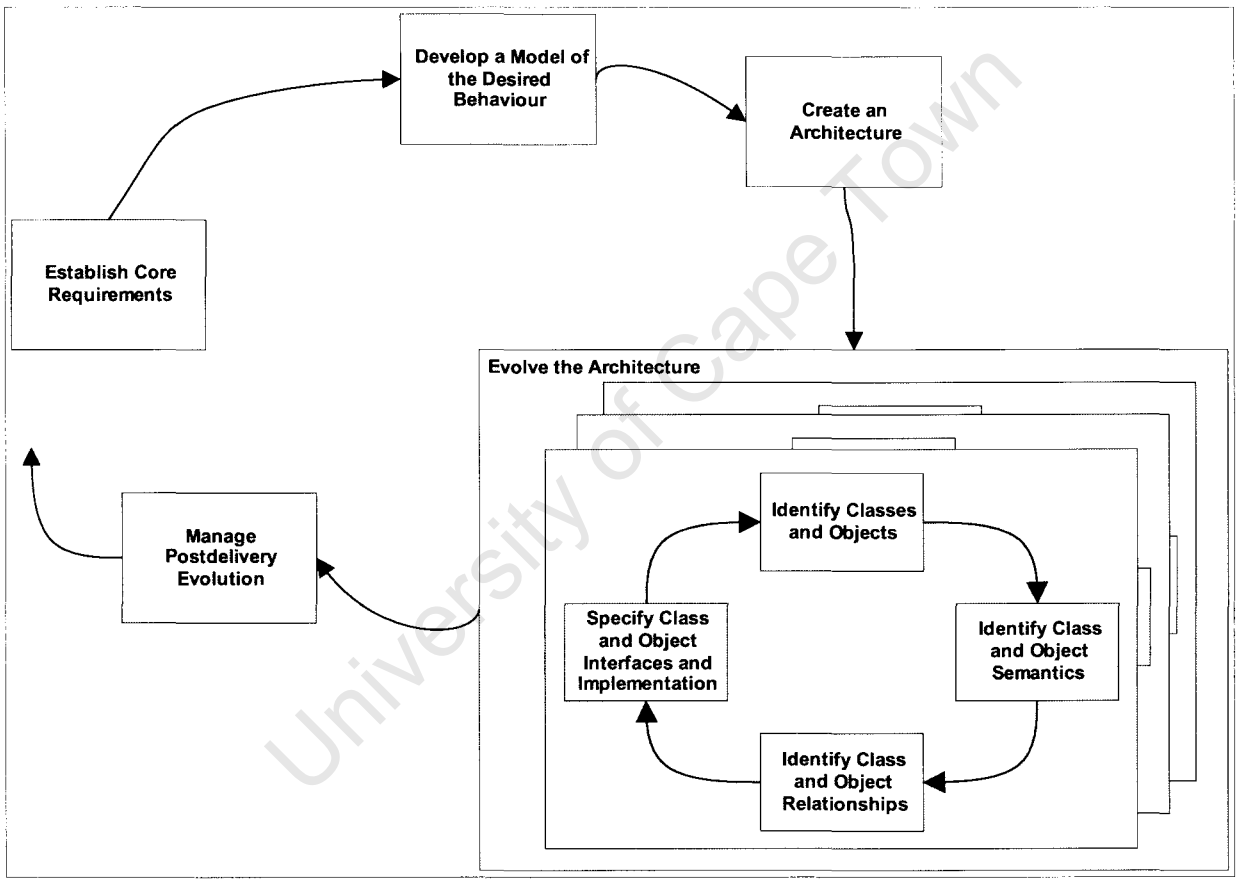

Figure 3: OOAD Process and Phases (Booch, 1994, pp. 235, 249, 259)

Jacobson was also the first to formalise the use case analysis approach, which he used substantially in OOSE (Booch, 1994, p. 158). Use case models use actors and use cases to define what exists outside the system (actors) and interact with the system, and what exists inside the system (use cases) as a behaviourally related sequence of transactions (Jacobson, 1992, p. 127). Use cases were subsequently incorporated in the Unified Modeling Language (UML) (Alhir, 1998, p. 71).

The Rational Unified Process (RUP) was developed by the Rational Corporation and was based on Jacobson's Objectory approach and Jacobson, Booch and Rumbaugh's (1998) Unified Software Development Process (Krutchen, 1999, p. 33). The RUP is, according to Krutchen (1999, p. 34), an

iterative software development process and process product which covers the entire software development life cycle. It aims to provide a disciplined approach to assigning and managing tasks and responsibilities within a development organisation and which has a goal of producing, within a predictable schedule and budget, high-quality software that meets the needs of its end users.

The Agile Paradigm

Highsmith (2002, p. xxiii) defines agility as "the ability to both create and respond to change in order to profit in a turbulent business environment" and states that agile methodologies, or ecosystems, are holistic environments that include three interwoven components – a "chaordic" perspective, collaborative values and principles and a barely sufficient methodology. On the other hand, Cockburn (2001, p. xxii) defines the agile paradigm as the use of light-but-sufficient rules of project behaviour and the use of human- and communication-oriented rules, allowing the agile process to be light yet sufficient (where lightness is a means of remaining manoeuvrable).

According to Highsmith (2002, p. xvii) and Abrahamsson, Salo, Ronkainen and Warsta (2002, p. 7), the agile paradigm had two significant beginning points, the first being in 1999, when Kent Beck introduced Extreme Programming, which is acknowledged as the first true agile methodology, and the second in 2001, when representatives of Extreme Programming, Scrum, Crystal Methods, Feature Driven Development and others met and produced the *Manifesto for Agile Software Development*, aimed as an alternative to what they saw as the document-driven, overly-rigorous software development methodologies that were being used at that time.

The Agile Manifesto declares that the signatories are attempting to uncover "better ways of developing software by doing it and helping others do it" and that through this work, they have come to value (Boehm, 2006, p. 19):

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

These principles, according to Abrahamsson *et al.* (2002, pp. 11, 12) emphasise a number of things about the agile paradigm. The first principle emphasises the relationship and commonality of software developers and the human role reflected in projects, as opposed institutionalised processes and tools, and manifests itself in close team relationships and working environments. The second principle emphasises that the vital principle of developing software is to continuously deliver tested, working software at regular and short intervals, often by lessening the burden of documentation. The third principle emphasises that the relationship and cooperation between the software development team and the customers is given preference over strict contracts and that the customers provide vital input into the development process. The fourth principle stresses that the software development team should be well-informed, competent and authorised to consider possible adjustments emerging throughout the development process, as opposed to following a plan blindly.

Abrahamsson *et al.* (2002, p. 9) state that the agile paradigm offers an answer to a business community asking for lighter weight along with faster and nimbler methodologies, more in tune with the rapidly growing and volatile Internet software industry as well as the emerging mobile application environment.

The reason for this could be as a result of the following characteristics (Miller, 2001) attributed to agile methodologies:

- A modular development process.
- An iterative development process, with short development cycles and frequent releases enabling both fast verifications and corrections.
- Iteration cycles last from one to six weeks.
- A level of parsimony with regards to the development process, which results in the removal of all unnecessary activities.
- Possible emergent new risks are taken into account and adapted to.
- An incremental process approach, which allows the development of a functioning application in small steps.
- A convergent (and incremental) approach which seeks to minimise risk.
- Agile methodologies are people oriented in that they aim to favour people over processes and technology.
- A collaborative and communicative working style.

Agile methodologies were initially criticised for a lack of proof of their claims. However, the responses to empirical surveys on the perceptions of agile methodologies amongst software developers, managers and final year computer science students who have undergone work experience training, were that the adoption of agile methodologies gives positive results. However, a second aspect of the response was that agile methodologies, especially Extreme Programming (XP), require adaptation to local environments and not all agile practices are equally valuable (Misic, 2006, p. 2). Abrahamsson *et al.* (2003, pp. 247, 251) argue that there has been criticism of agile methodologies for not being complete in terms of life cycle coverage, but also present the argument of whether it is "more profitable to cover more and to be more extensive, or cover less and be more precise".

While Kumar and Welke (1992, in Abrahamsson *et al.*, 2003, p. 251) argue that completeness requires that software development methodologies be complete as opposed to partial, Abrahamsson *et al.* (2003, p. 51) state that in their analysis, completeness is an element that refers to both the vertical (i.e. level of detail) and horizontal (i.e. life cycle coverage) dimensions and that none of the agile software development methodologies that they analysed were either extensive or precise. Another aspect of criticism that has been levelled against agile methodologies is shown in the work of Abrahamsson *et al.* (2003) where only three of the agile methodologies analysed offered concrete

guidance over all phases of the life cycle that they covered, while Abrahamsson *et al.* (2003, p. 251) note that abstract principles "appear to dominate the agile methods literature and developers' minds".

A relatively large number of agile methodologies exist, as evidenced by Abrahamsson *et al.* (2002), who analysed and compared ten and mentioned more, and Noble, Marshall, Marshall and Biddle (2004, p. 217), who mention a further four over and above those mentioned by Abrahamsson *et al.* (2002). Two of the most commonly used are Extreme Programming and Feature-Driven Development. Extreme Programming is a lightweight software development methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements (Beck, 1999, p. 31).

It stresses four values, namely communication, simplicity, feedback and courage. XP aims to keep communication flowing by employing many practices that cannot be performed without communication, such as pair programming and unit testing. Simplicity refers to the fact that Extreme Programming works on the basis of finding the "simplest thing that could possibly work", while feedback is the treatment for the occupational hazard of programming that is optimism (Beck, 1999, pp.31, 32). Extreme Programming aims to facilitate communication through the use of feedback, both in terms of inter-team feedback (through pair programming and unit tests) and between the team and the customer (through having an on-site customer). The value of courage is described as being brave enough to make large changes to code, as well as being brave enough to throw code away during the effort of refactoring the code (Beck, 1999, p. 33).

Feature Driven Development (FDD) has been defined as a model-driven, short-iteration process, which begins by establishing an overall model shape, continues with a series of two-week "design by feature, build by feature" iterations. A feature is a small piece of functionality viewed as "useful in the eyes of the client" (Palmer, 2002). The process used by FDD is as follows (*Figure 4*):

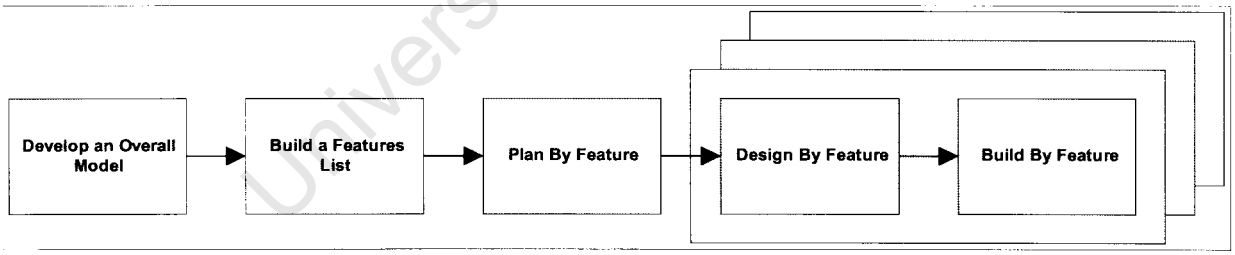

Figure 4: Feature Driven Development process (Palmer, 2002)

At the beginning of the process, the deliverables involve "more shape than content" (Palmer, 2002), in the sense that they are not very detailed, but more descriptive. The detail, and thus "content" of the specifications increases through the process, until at the sub-processes of Design By Feature and Build By Feature, the specifications are "more content than shape", and more textual than diagrammatic. Developing an Object Model sub-process involves an iterative process where the domain and development members define and refine an object model for the application. The sub-process around building a features list involves the creation of a list of functionality, or "identify[ing] all the features needed to support the requirements" (Palmer and Felsing, 2002, p. 3). Planning by features involves the prioritising and scheduling of the features and the assignation of these features

to members of the development team. The Design By Feature and Build By Feature sub-processes are the "development engine room" (Palmer, 2002). In these processes, the features are specified in detail and then built by the development team over one-to-two week iterations until the system is complete.

Conclusion

In conclusion, this chapter has discussed the most important aspects of methodologies in the context of this research. These aspects have included definitions of the most important constituent parts of methodologies, a discussion of the theory and attributes of methodologies, the context in which methodologies operate and the possible motivations for and against the adoption of a formalised methodology. Finally, a discussion of the main methodological paradigms was provided, with some insight into the chief methodologies that utilise these paradigms. The following chapter moves from describing methodologies to describing the evaluation of methodologies, in the form of the literature survey.

Chapter 3: Overview of Existing Frameworks

This chapter surveys some of the previous work performed in the field of analysing, comparing and evaluating methodologies, with a particular emphasis placed on the frameworks proposed and used in previous research. Sol (1983, in Jayaratna, 1994, p. 47) argues that there are five different ways of undertaking methodology comparisons, specifically:

- Describing an 'ideal' methodology and then comparing other methodologies against it.
- Constructing a 'generalised' measurement tool by selecting appropriate features from existing methodologies.
- Testing hypotheses about the features of a methodology based on the study of different methodologies.
- Developing a common frame of reference for viewing different methodologies.
- Developing a contingency framework to allow the appropriate methodology to be mapped onto a certain environment.

It appears that, from the literature surveyed, the most common way of evaluating methodologies involves the use of a common frame of reference for viewing methodologies. This is evidenced in the detailed frameworks proposed by Kelly (1987), Mannino (1987), Jayaratna (1994), Avison and Fitzgerald (1995), The Object Agency (TOA) (1995) and others. The level of detail in the surveyed research differs substantially; some of the work, such as Jayaratna (1994) and Avison and Fitzgerald (1995) offers detailed frameworks for evaluating methodologies, while others are much less detailed. However, work which offers new or alternative perspectives, such as Singh and Kotze (2003), who focus on the human component of the methodologies evaluated and Sorenson (1995), who focuses on the perceived strengths and weaknesses of the evaluated methodologies, is included here.

The focus of the evaluations or comparisons is also often very different. The scope of the research by Abrahamsson *et al.* (2003) is on a specific branch of the software development methodological tree, in that it is a comparison of agile software development methodologies. Kelly's research (1987) focuses on an even more specific subset of software development methodologies; that of real-time design methodologies. Burns and Dennis (1985) and Mannino (1987) compare the prototyping methodology to structured methodologies. The principles of comparing methodologies are, however, largely common and mostly methodology-neutral, as shown in this section.

A substantial amount of the research (Burns and Dennis, 1985; Mannino, 1987; Abrahamsson *et al.*, 2003; Singh and Kotze, 2003; and others) is similar in that the software development life cycle was used as a basis for the analysis of the software development methodologies, with the research analysing the different ways that the software development methodologies support the phases of the life cycle.

A large number of methods and methodologies were analysed, compared and/or evaluated. A non-exhaustive list comprises Adaptive Software Development, Agile Modeling, the Boehm Spiral Approach, the Box Structure Methodology, the Coad and Yourdan Model, COPA, the Crystal family of methodologies, that suggested by Dennis, Dynamic Systems Development Method, Extreme Programming, FAST, Feature-Driven Development, FORM, Hackos, the IBM Model, Internet-speed Development, JSD, KobrA, the Object Modeling Technique, Object-Oriented Analysis, Object-Oriented Design, PAMELA, Pragmatic Programming, (Application) Prototyping, QADA, Redish, SASD, SCR, Scrum, Structured Design for Real-Time Systems and the Waterfall approach. In total, more than 30 software development methodologies were analysed and compared in the surveyed research.

The bulk of the work surveyed is from the last 25 years, which reflects the relative youth of the field of software development methodologies. The research surveyed is organised chronologically, from oldest to most recent.

Van den Bosch et al. (1982)

Van den Bosch, Ellis, Freeman, Johnson, McClure, Robinson, Scacchi, Scheff, von Staa and Tripp (1982) define the basic dimensions for analysing the effectiveness of the implementation of a methodology. These dimensions are the software life cycle, the organisation and the methodology. They (van den Bosch *et al.*, 1984, p. 53) describe the following characteristics that need to be analysed in terms of the software life cycle:

- The phases and/or sub-phases of the life cycle, in terms of the activities, resulting products, required resources and the accepted products from previous activities.
- the products of the life cycle, in terms of specifications, designs, standards, internal and external documentation, program code, verification and validation data and results, and acceptance criteria.
- The schedules of the life cycle, in terms of both time and the sequence with which the products are to be delivered, control points and milestones, schedules of resource requirements and change control methods.

Once the life cycle has been analysed, van den Bosch *et al.* (1982, p. 54) believe that the software development methodology itself should be categorised, based on criteria such as the life cycle phases covered, included sub-methodologies, training requirements, required skills, the types of projects/applications the methodology is appropriate for, and the benefits to be achieved by using the methodology. Like Kitchenham's (1996), this framework is based on implementation-specific criteria and needs, specifically the chosen project life cycle in this case, and is thus possibly not valid in general terms as most methodological paradigms follow the same basic life cycle. However, van den Bosch *et al.* (1982) do propose categorisation criteria, such as life cycle phases covered, that should be considered valid and which have been used in subsequent research including Mannino (1987), Kelly (1987), Avison and Fitzgerald (1995) and Abrahamsson *et al.* (2002).

Burns and Dennis (1985)

Burns and Dennis (1985, p. 20) provide a framework for selecting a methodology for use in a software development project based on two dimensions; project complexity and project uncertainty. The goal of this research is the provision of a framework to allow the selection between more adaptive methodologies, such as prototyping, and more structured methodologies, based on a contingency approach. They (Burns and Dennis, 1985, p. 21) argue that there are three contingencies to determine project uncertainty, namely:

- Degree of structuredness, which describes how fixed the system definition is, and how clear the new system structure is.
- User task comprehension, which refers to the degree of understanding the users have about their current tasks.
- Developer task proficiency, which is a measure of the specific training and experience brought to the project by the developers.

In addition, they describe four contingencies to determine project complexity, namely:

- Project size, as measured in man-hours.
- Number of users, where a larger number of users generally means a more complex system.
- Volume of new information, where the greater the volume of new information generated by a system, the higher the complexity of the system.
- Complexity of new information production, which refers to the degree of system complexity needed to produce the volume of new information detailed in the contingency above.

The result of this comparison is a decision table, shown in the following illustration (*Figure 5*).

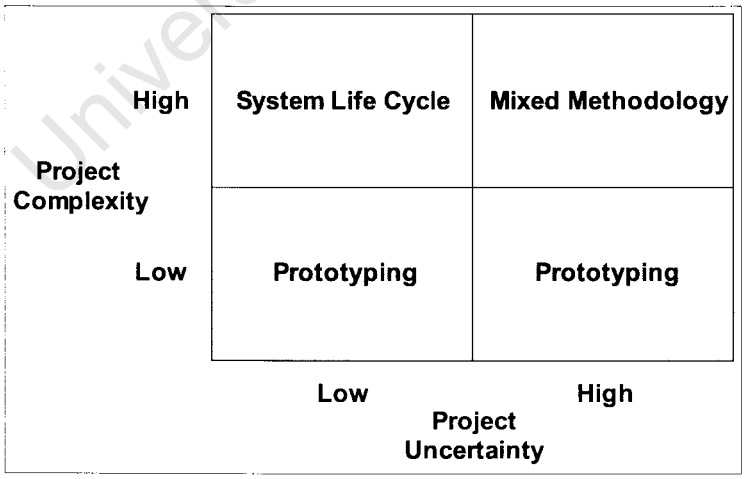

Figure 5: Decision Table (from Burns and Dennis, 1985, p. 21)

In their comparison of methodologies, Burns and Dennis (1985) compare three methodologies, the traditional Systems Development Life Cycle methodology, the Prototyping methodology and the Mixed Methodology, which incorporates aspects of both. As a result, only these three methodologies were shown in the resulting decision table, but this could conceivably be expanded to cover more

methodologies. With that said, this approach is probably better suited to compare paradigms than methodologies, as the contingency criteria are perhaps too broad to differentiate between methodologies that are similar in approach.

Mannino (1987)

Mannino (1987) provides criteria for comparing software development methodologies, and validates these criteria by comparing traditional methodologies (SASD and JSD) to methodologies that were, at the time of the research, considered to be more agile, such as application prototyping and the Box Structure Methodology. Application prototyping is considered by Abrahamsson *et al.* (2003, p. 246) to be a forerunner to the true agile methodologies such as Extreme Programming. Mannino (1987, p. 28) states that, "in order to compare [the methodologies], a standard life cycle must be defined, and each of the methodologies must be moulded to it". As a result, the methodologies were compared using a software development life cycle which comprised of five phases: Feasibility, Analysis, Design, Implementation and Maintenance (Mannino, 1987, p. 28), which are very similar to those comprising the life cycle described in *Table 1* and *Figure 1*.

The comparison involved determining how well each of the software development methodologies addressed each of the defined life cycle phases, as well as a number of secondary criteria such as completeness, consistency, graphical representation of the systems (or diagramming techniques), assumptions, system size, quality assurance, data storage, constraints and skill needed to apply the methodology (Mannino, 1987, pp. 28, 30-32). Mannino's comparison (1987) does not use a framework to organise the comparison, instead comparing the methodologies based on their activities during the SDLC, and a number of secondary criteria. While the approach may be considered valid, it would not add significantly to the current body of work to repeat this research using other methodologies. Aspects of this approach, specifically the coverage of the software development life cycle and some of the criteria used, could, however, be included in a comparison of methodologies.

Kelly (1987)

Kelly (1987) offers a framework for comparing software design methodologies for real-time systems. The design methodologies compared were Structured Design for Real-Time Systems, Object Oriented Design, PAMELA and SCR. While these methodologies are very specific, both horizontally (in terms of the focus on only the design phase of a project) and vertically (they focus, with the exception of Object Oriented Design, specifically on real-time systems), more important is the framework, or aspects thereof, used by Kelly (1987). The framework proposed by Kelly (1987) is based on criteria such as the methodology's formal foundation (the way in which the methodology views software), semantic soundness (how tightly the design specifications bind the programmer to the designers intentions), ease of change and maintainability, performance design and the degree of support for phases of the life cycle other than the design phase. Also included are criteria specifically suited towards real-time systems, such as concurrent processing and communications.

While there are many criteria within Kelly's framework that can be considered valid for general methodology analysis and evaluation, such as the formal foundation of the methodology, there are aspects that would not be considered in the analysis and comparison of more general methods such as the real-time specific criteria. Kelly's framework focuses on real-time methodologies, which are a small and specific sub-tree of methods, and the criteria included specifically for their benefit should be excluded in a comparison of more general methods. Secondly, Kelly's framework is intended only to analyse and compare the design aspects of methodologies and, as a result, does not take the full software life cycle into account, which would be necessary in evaluating the completeness of a methodology.

Connors (1992)

Connors (1992) offers a framework for comparing methodologies for the development of different types of systems, focusing on traditional information systems, decision support systems and expert systems. In describing the different ways in which these systems are traditionally developed, Connors (1992, pp. 44-46) describes the requirements and project characteristics for their development and, as a result, provides a set of scale-based criteria which can be used to differentiate between different types of software projects, as shown in *Figure 6*. Connors (1992, p. 47) also suggests an integrating model for use in the decision-making process of selecting a methodology for the development of an Management Information System (MIS), Decision Support System (DSS) or Enterprise System (ES). The model below (*Figure 7*) shows the shift in use of decision support and expert systems, from being largely on an individual level, to an organisational level (and thus using a significantly larger set of requirements, as well as the technical considerations needing to be taken into account on a larger system), and the shift of management information systems away from the organisational level toward the individual level.

One important role of these shifts, Connors (1992, p. 48) believes, is that they should affect the way in which methodologies for the development of these systems are chosen, in this way developing an organisational DSS will require a more structured methodology than the methodologies used previously when DSS's were developed on a more *ad hoc* basis for individual users.

Connors' model is, however, not sufficiently detailed, as detailed recommendations are not made, nor is a complete framework, in terms of methodology comparison and/or selection, offered. While the shifts in requirements for methodology types in his integrating model are described, in *Figure 6* (Connors, 1992, p. 47), very few recommendations are made as to which characteristics are required of the methodologies. Connors (1992, p. 48) does, however, recognise the limitations in the integrating model stating that it is a "bare concept requiring further work to determine when each kind of methodology is most appropriate, and what other variables need to be considered".

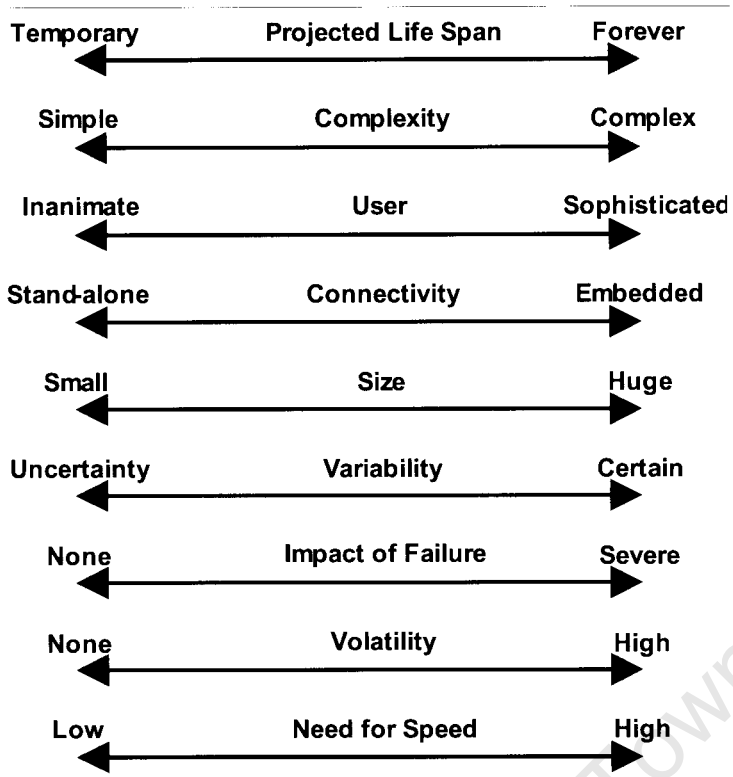

Figure 6: Project Types (from Connors, 1992, p. 47)

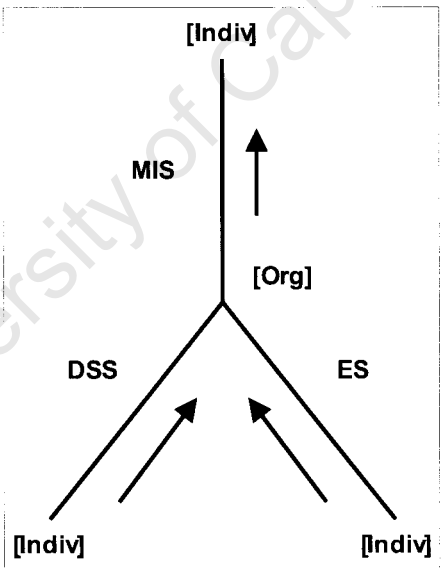

Figure 7: Integrating Model (from Connors, 1992, p. 47)

Kelley and Rogers (1992)

Kelley and Rogers (1992, p. 177) provide a framework for the evaluation of methodologies based on five elements, namely:

- Cost, measured not only in terms of the purchase price of the methodology, but also the cost of using the methodology and its associated hardware and software.
- Usability, incorporating aspects such as user friendliness, which can be broken down into practicality, convenience and helpfulness, quality, suitability, and functionality.

- Power, or the ability of the methodology to increase the productivity of individuals using the methodology to do what is required to complete the project.
- Flexibility, or the extensibility, tailorability and scalability of the methodology.
- Maturity, or the methodology's experience, user base, trained personnel, progress over time and stability.

These elements are then used in an evaluation consisting of five steps (Kelley and Rogers, 1992, pp. 177, 178). First, a basis for the review of the methodology should be established by gaining an understanding of available methodologies, the context in which the methodology will operate and any essential methodology characteristics. Next, the goals for the evaluation must be established in terms of specific and bounded criteria to serve as the "compass" for the evaluation, along with the selection of a test case to provide the appropriate direction for the evaluation. Following this, the test case must be conducted using the selected methodologies, "Evaluat[ing] a broad range of framework functions, but explor[ing] issues of particular importance to the user organization at some depth" (Kelley and Rogers, 1992, p. 178).

After the evaluation has been performed, the recommendations and conclusions must be documented. The changes and enhancements to be considered for future versions of the methodology, whether the framework is adequate for the project it would be used on, and its suitability to future projects must be discussed in the document. Finally, the evaluation process should be refined to take into account the findings on issues raised, the rationale for decisions and any problems encountered during the evaluation. This framework is less applicable in the context of this research, as it requires empirical research as well as interpretation of the recommendations and conclusions.

McLeod (1992)

Rather than proposing a framework for evaluating methodologies, McLeod (1992) argues that there are a number of issues that traditional methodologies do not address correctly, and describes how methodologies might overcome these issues. These issues, which could be used as, or at least inform, methodology evaluation criteria include such aspects as the methodology's focus on people and management, methodology architecture, a methodology model and the level of integration of quality into the methodology products and tasks.

McLeod's work does not propose a framework, merely a set of key issues for inclusion, as well as arguments for their inclusion, in methodologies that do not include the issues. To date, there has not been enough research on McLeod's key issues. Further investigation is needed to test the usefulness of these criteria in the comparison of methodologies and in the decision-making process of adopting a methodology.

Rozman et al. (1992)

In the same vein as McLeod (1992), Rozman, Gyorkos and Rizman (1992) focus on a specific methodology evaluation criterion, in this case the understandability of methodologies, rather than proposing a framework for methodology evaluation. In this case, methodology understandability is used as a criterion for selecting a CASE tool to support the methodology. The relevance of this research is predicated on the method with which they performed the investigation on post-graduate students completing an examination in which they were tasked with solving a problem using methodologies that they had studied (Rozman *et al.,* 1992, p. 44). Once they had completed their examination, they were given a questionnaire which related to their perceived understanding of methodologies, and which methodologies they planned on using in the future.

The results received were illuminating: of the two methodologies utilised, Jackson Structured Development (JSD) and Structured Analysis Structured Design (SASD), the students considered JSD substantially more difficult to understand and viewed SASD as more effective and easier to understand (Rozman *et al.*, 1992, p. 45), which corresponded to their preference in terms of using SASD on future projects.

Hong et al. (1993)

Hong, van den Goor and Brinkkemper (1993) present a formal approach to the comparison and evaluation of Object Oriented (OO) analysis and design methodologies, based on modeling methodology processes and concepts. They (Hong *et al.*, 1993, p. 689) believe that, in order to be accurate and objective, OO methodologies should be compared on a uniform, formal and unbiased basis. This approach is based on two steps. The first step involves the construction of a formal representation of two meta-models of the methodologies, representing the analysis and design steps, and the concepts and techniques provided by each methodology (Hong *et al.*, 1993, p. 689). The second step involves the comparison of the methodologies based on the constructed meta-models. Hong *et al.* (1993, p. 690) motivate their approach by stating that "various analysis and design methodologies can be viewed as specific modeling processes that apply a specific set of techniques", thus supporting the process of metamodeling in the analysis of the methodologies.

In their research, Hong *et al.* (1993, p. 691) constructed two meta-models per methodology compared, corresponding to their view that a methodology contains two aspects, processes and concepts. The first meta-model, the meta-process model, describes the activities and steps of the analysis and design phases, including the inputs and outputs of each step. The second meta-model, the meta-data model, describes the concepts and associations between the concepts, as applied in the various diagrammatic and textual techniques of the methodology (Hong *et al.*, 1993, p. 691).

The comparison of methodologies (Hong *et al.*, 1993, p. 693) was based on three concepts: the processes used by the methodologies, the concepts used in the methodologies, and the techniques provided by the methodologies. In terms of the comparison of processes, the methodologies were compared against a 'supermethodology'. This is defined by Hong *et al.* (1993, p. 694) as the smallest

common denominator of all activities depicted in the meta-process models of the compared methodologies, to determine whether they did less, the same or more per activity than the supermethodology. For evaluation purposes, the main concepts relating to the OO paradigm were compared, such as Class, Object, and the types of relationship, as well as more detailed criteria such as concurrency mechanisms (Hong *et al.*, 1993, p. 694). The criteria for the comparison of techniques is based on the techniques used to capture the concepts (Hong *et al.*, 1993, p. 694). For example, classes can be captured using an Object Model OO Analysis Diagram using the OOAD methodology, or using an Object Diagram using the OOAD methodology (Hong *et al.*, 1993, p. 695).

However, Hong *et al.* submit that the research is limited in some ways (1993, p. 697). Firstly, no comparison was made of the guidelines and rules provided by each methodology. Secondly, because they were limited by the Entity Relationship model, several concepts of some of OO methodologies are particularly difficult to represent in meta-models. The result of this is that the accuracy of the comparison results may be negatively affected. Finally, Hong *et al.* (1993, p. 697) note that they did not compare how a methodology guides the user to design a better software system and to make the most use of OO technology such as reusability.

Jayaratna (NIMSAD) (1994)

Jayaratna (1994, p. 45) proposes the NIMSAD framework for methodology evaluation, with the stated aims of understanding the area of problem solving, helping to evaluate methodologies and their structure, steps, form and nature; and helping to draw conclusions about the methodologies evaluated. The NIMSAD framework has four elements, namely: the problem situation, or methodology context; the intended problem solver, or methodology user; the problem-solving process, or methodology; and the evaluation of the above three (Jayaratna, 1994, p. 49). The framework is shown in *Figure 8.*

In terms of the problem situation element of the framework, Jayaratna (1994, p. 59) argues that organisations need to be considered as purposeful systems, or systems which have to be designed or formed to achieve their purposes. As a result, intended problem solvers need to acquire a deep understanding of the organisation if they are to become effective problem solvers. The NIMSAD framework thus uses a general model to show the essential elements of a problem situation and their formal and informal interconnections and relationships (Jayaratna, 1994, pp. 61-63).The design and development and, ultimately, the success of effective and efficient information systems is largely dependent on the intended problem solver, or methodology user, the second element of the NIMSAD framework (Jayaratna, 1994, p. 63). Jayaratna (1994, p. 64) suggests the use of a "mental construct" model to evaluate this element, using criteria such as perceptual process, values, motives and prejudices, reasoning ability and skill and knowledge sets. The aim of the second framework element is to identify how a methodology helps its users to make sense of situations, manage relationships with others, take action and identify and solve problems (Jayaratna, 1994, p. 70).

The third element of the NIMSAD framework is the problem-solving process. This phase has three sub-phases (problem formulation, solution design and design implementation), which in turn have a number of associated stages. These stages include, amongst others, gaining an understanding of the 'situation of concern', performing the diagnosis, deriving notational systems, performing the conceptual and physical designs of the system and implementing the conceptual and physical designs (Jayaratna, 1994, pp. 74, 75). The final and most important element of the NIMSAD framework is, according to Jayaratna (1994, p. 108), the evaluation. Jayaratna (1994, p. 108) states that the previously discussed elements form the basis of the evaluation. This evaluation must be carried out at three stages:

- Before intervention (the introduction of the methodology), to maximise both efforts and effectiveness.
- During intervention, because of the dynamic nature of the elements.
- After intervention, so that lessons can be learned about the elements.

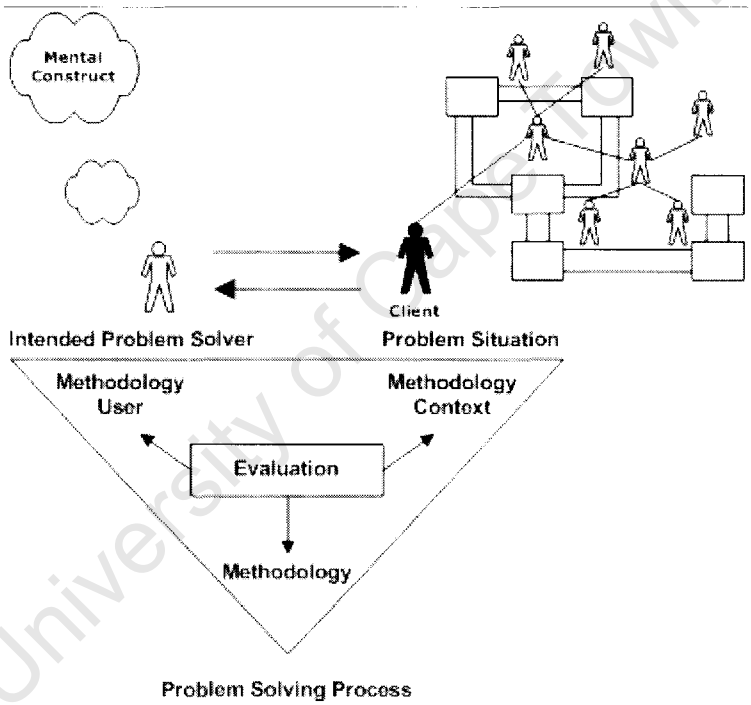

Figure 8: The NIMSAD Process (from Jayaratna, 1994, p. 54)

Avison and Fitzgerald (1995)

Avison and Fitzgerald (1995, p. 446) propose a generalised framework for comparing any software development methodologies, based on seven elements. The elements are:

- Philosophy, which is divided into the following sub-elements:
  - Paradigm.
  - Stated objective(s), whether this is the narrow view of computerisation, or "achieving solutions or improvements no matter what this implies" (Avison and Fitzgerald, 1995, p. 451).

    - Domain, or the situations that the methodology addresses.
    - Target, the applicability of the methodology, whether the methodology is said to be targeted at specific problems, environments or organisations, or is said to be general purpose.
- Model, or the basis of the methodology's view of the world and both a means of communication, a way of capturing the essence of a problem or design and a representation which provides insight into the problem (Avison and Fitzgerald, 1995, p. 452).
- Techniques and tools.
- Scope, the life cycle stages covered by the methodology.
- Outputs, the deliverables produced during each of the life cycle stages of the methodology.
- Practice, which is also divided into sub-elements.
    - Background, commercial or academic.
    - User base, the numbers and types of users of the methodology.
    - Players, the participants in the methodology.
- Product, or "what purchasers actually get for their money" (Avison and Fitzgerald, 1995, p. 457).

They also believe that a number of other elements could be added to the framework, specifically the speed at which systems can be developed using the methodology, the quality of the specifications and documentation produced, and the potential for modification of the methodology by users to suit their needs (Avison and Fitzgerald, 1995, p. 446).

Avison and Fitzgerald (1995, pp. 435-437) also summarise the views of a number of authors (Catchpole, 1987; Land, 1976) covering the important areas of concern when comparing or designing methodologies. They believe that a methodology should, amongst other things, have rules and formal guidelines to cover phases, tasks and deliverables. They should also offer total coverage of the systems development process, have documentation standards, support the separation of logical (descriptions and requirements) and physical design (specific software-related design), offer a high level of communication between life cycle phases and have support for problem analysis by a means of expressing and documenting problems and objectives. The aims of these criteria are to:

- Increase productivity on software development projects.
- Improve the quality of the products of the analysis, design and programming activities.
- Maintain the visibility of the emerging and evolving information system.

Bjorn-Anderson, (1984, in Avison and Fitzgerald, 1995, p. 438) proposes a checklist, composed of a number of criteria, relating specifically to values and society. These criteria include aspects such as

the methodology's paradigm, the underlying value systems of the methodology, the context where the methodology is to be used, the extent to which modification is possible, whether the societal environment is dealt with, and to what extent user participation is encouraged. Avison and Fitzgerald believe that this framework is useful "as it focuses attention on some wider issues that are often ignored" (1995, p. 438).

The framework proposed by Avison and Fitzgerald (1995) is one of the more complete frameworks surveyed, as it takes into account aspects such as the paradigms upon which methodologies are based. Methodology paradigms, which are described later, can be wildly divergent, especially with the advent of the agile paradigm. This can be viewed as a very important aspect in the process of selecting a methodology for a project. There are, however, aspects to their framework which can be difficult to quantify, such as user base.

The Object Agency (1995)

Another comparison framework for methodologies is that suggested by the Object Agency (TOA), which is intended to assist organisations in evaluating the availability and appropriateness of object oriented (OO) methodologies for their environment (TOA, 1995, p. 1). The framework considers six areas of the compared methodologies, namely concepts, notations, process, pragmatics, support for software engineering and marketability and consists of a series of questions (asked against the methodology) that are used to identify and quantify a methodology's support for specific development processes. In the context of this research, it is possible that not all of the criteria relating to the concepts area would be considered valid, as some of the concepts are specific to OO methodologies and would thus not be valid against non-OO methodologies, for example Class, Object etc. However, some of the criteria, specifically relating to the perspective of the methodology, could be considered broad enough to be included in a comparison of OO and non-OO methodologies.

TOA (1995, p. 8) specify a number of areas of a notation that should be analysed during the evaluation or comparison of a methodology and state that "in assessing a method it is necessary to consider the models the method requires and the notations required of each model". The areas they deem as important are:

- Expressiveness, the ability to create complete, easy to understand models.
- Syntax and semantics, specifically how well-defined they are.
- Scalability, the ability to represent models of differing sizes.

Another identified area is the process used by the methodology (TOA, 1995, p. 8). It's purpose is to characterise the contexts that the methodology is appropriate for, which phases of the life cycle are covered and what tailoring or heuristics are available. The areas of analysis identified within this are development context, support for reuse, life cycle coverage and the properties and attributes associated with the methodologies process, such as sequence of steps, required inputs and outputs, roles and interaction with other steps (TOA, 1995, pp.8-10).

In terms of pragmatics, the areas of analysis are defined by TOA (1995, p. 10) as resources, required experience, accessibility, language suitability and domain applicability. The areas of analysis for the methodology's support for software engineering (TOA, 1995, pp.10, 11) are concerned with evaluating the level of support for software engineering goals and principles such as reusability, testability, modifiability and conceptual integrity. Marketability is the final part of the framework suggested by TOA and is, according to them (1995, p. 11), something that is seldom made explicit in the comparison of methodologies. These criteria are largely organisation-specific, but can include issues such as price, whether an automated tool is required, and whether the methodology supports specific types of documentation (TOA, 1995, p. 11) .

As this framework is intended for use in evaluating OO methodologies, it includes paradigm-specific criteria, which is not applicable to methodologies based on other paradigms. However, there are aspects, such as process, which are not based on a specific paradigm, and could thus be used in a cross-paradigmatic approach.

Sorenson (1995)

Sorenson (1995) proposes a framework for evaluating the strengths and weaknesses of, and thus choosing, software development methodologies. These are classified into a number of criteria based on perceived positive and negative aspects of software development projects. These criteria include aspects relating to perceived 'strengths', such as whether the methodology can be reported on, whether the methodology can control costs and risks through prototyping, and whether the methodology provides functionality early in the project. They also include perceived 'weaknesses', such as whether the methodology requires a complete set of requirements at the beginning of development, whether the methodology is incompatible with a formal review and audit procedure and whether the methodology has a tendency for difficult problems to be pushed to the future so that the initial promise of the first increment is not met by subsequent products.

It could, however, be argued that because some of Sorenson's criteria are based on perceived strengths and weaknesses, other schools of thought might consider a perceived weakness of a methodology as a strength. As an example of this, Connors (1992, p. 46) states the United States Department of Defence were spending at least USD$30 billion annually by 1992 on software development, thus making them one of the largest-spending organisations in that area, and uses the DoD-STD-2167A standard, which requires that all specifications be complete before implementation can commence. Sorenson (1995) considers this a perceived methodological weakness.

Fitzgerald (1996)

Fitzgerald (1996) provides an empirically-grounded framework to describe the role of a software development methodology in the IS development process, based on a number of reasons for and against the use of formalised methodologies. This framework serves to describe how environmental aspects can shape the adoption and use of a methodology and takes into account aspects such as the development environment and context, the overt/intellectual and covert/political roles of the

methodology, developer-embodied factors and the methodology user (Fitzgerald, 1996, p. 107). This is shown in *Figure 9*.

Some interesting points to note from this research is that it appears to have an orientation towards method engineering, as it allows for the influence of the aspects described above in influencing, shaping and enacting/instantiating a methodology-in-action from an original, formalised methodology (Fitzgerald, 1996, p. 107). Based on this, Fitzgerald (1996, p. 111) states that "[t]he notion of a formalized development methodology seems not to be all that relevant in practice, in so far as a methodology-in-action appears to be uniquely enacted for each development project".

Kitchenham (1996)

Kitchenham discusses the DESMET approach for evaluating methodologies, an approach based on a number of evaluation methods as well as formal experiments (1996, p. 11). According to Kitchenham (1996, p. 11), the DESMET approach is concerned with evaluating methodologies or tools currently being used, or investigated for use, within an organisation, and allows an evaluator to plan and execute an evaluation that is unbiased and reliable, taking into account societal and managerial factors that could bias an evaluation.

The DESMET approach separates evaluations into two main types, specifically evaluating measurable effects of using a methodology or tool, and evaluating methodological appropriateness in an organisation (Kitchenham, 1996, pp.11,12). To perform these evaluations, Kitchenham (1996, pp. 12-15) describes nine distinct evaluation types, including quantitative and qualitative experiments, quantitative and qualitative case studies and benchmarking.

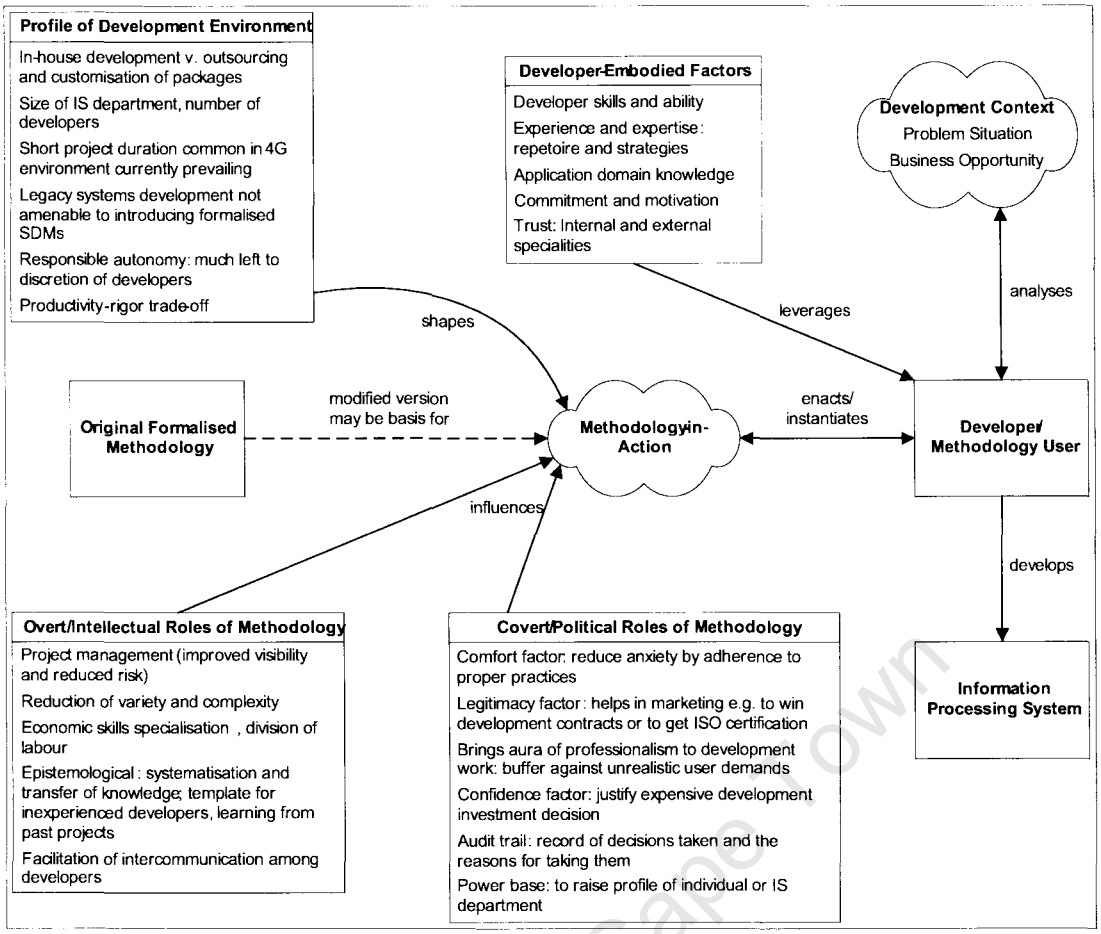

Figure 9: Fitzgerald's Framework for the IS Development Process (1996, p. 107)

While the DESMET framework is a useful tool for the evaluation of methodologies, it is probably too context-specific for a context-neutral comparison of methodologies, as Kitchenham (1996, p. 12) states that it is aimed at evaluating specific methodologies or tools in specific circumstances. As a result, the evaluation methods, with the exception of the survey method which is based on past experiences, are dependent on the methodology being implemented on a project.

Moody et al. (Method Evaluation Model) (2001)

The Method Evaluation Model was proposed by Moody (2000) in his PhD thesis, and incorporates two aspects of methodology success, specifically actual efficiency and adoption in practice, arguing that methodology success is thus based on a combination of these two aspects of a methodology (Moody, Sindre, Brasethvik, Sølvberg, 2001, p. 247). According to Moody *et al.* (2001, p. 247), the framework combines Methodological Pragmatism, a theory for validating methodological knowledge, and the Technology Acceptance Model (TAM), a theoretical model for explaining and predicting user acceptance of information technology. The framework is described by the following diagram (*Figure 10*).

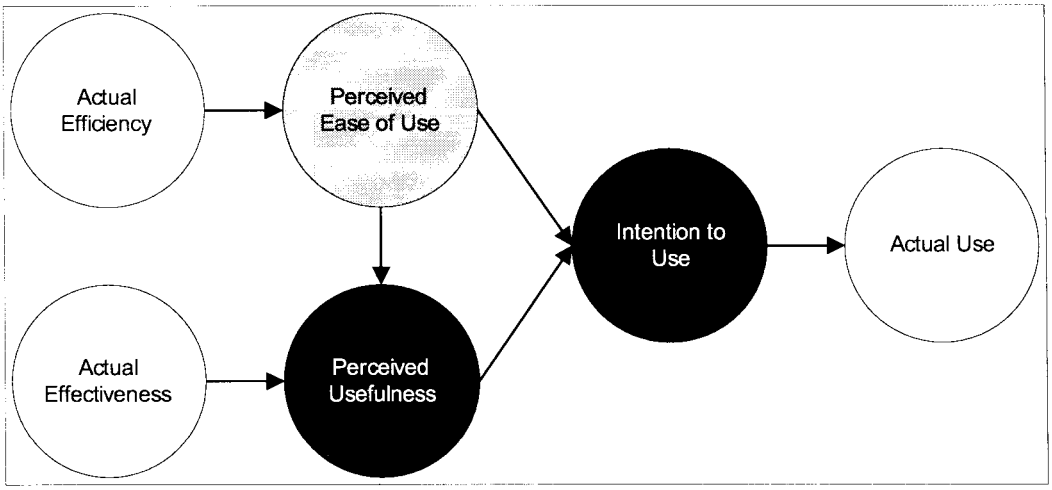

Figure 10: Method Evaluation Model (Moody et al., 2001, p. 297)

The model has six constructs (Moody, Sindre, Brasethvik, Sølvberg, 2001, p. 247), namely:

- Actual efficiency, or the extent to which the methodology reduces the effort required to perform a task.
- Actual effectiveness, or the extent to which the quality of the result is improved by the methodology.
- Perceived ease of use, or the extent to which a person believes that the methodology would be easy to use.
- Perceived usefulness, or the extent to which a person believes that the methodology would be useful.
- Intention to use, or the extent to which a person intends to use the methodology.
- Actual usage, or the extent to which the methodology is actually used.

One issue that could be raised with this framework in the context of comparing methodologies is that, being based on the TAM model, it is biased towards aspects around the use of the methodologies. These criteria are beyond the scope of this research, as they are difficult to measure without detailed quantitative data, regarding aspects such as actual efficiency and effectiveness. Secondly, aspects relating to perception (ease of use and usefulness) will differ on an individual basis, and would thus be difficult to include in an objective comparison without empirical, opinion-based data from expert users of methodologies. This model would seem more suited to evaluating the implementation and use of a single methodology.

McLeod and Roeleveld (2002)

McLeod and Roeleveld (2002, p. 1) propose a framework for methodology evaluation based on three criteria, namely methodology effectiveness, modeling capability and methodology complexity. Each criterion relies on one or more frameworks based on previous work in the field of methodology comparison, specifically the NIMSAD framework (Jayaratna, 1994), the Krogstie, Sindle and Lindland

quality assessment framework (Krogstie *et al.*, 2000), the Chart of Concepts approach (Castellani, 1998) and the Method Points methodology (McLeod, 1997). The framework was validated using an evaluation of the Rational Unified Process and the Inspire methodology, two commercially available methodologies used for software development projects (McLeod and Roeleveld, 2002), thus demonstrating a level of effectiveness in the same context of the framework proposed by this research.

In terms of comparing methodology effectiveness, the NIMSAD framework (Jayaratna, 1994) was used. As this framework is detailed above, it will not be described here. The framework proposed by Krogstie *et al.* (2000) was used to compare the modeling capability of the methodologies being compared. This framework addresses the constructs of the language (the concepts and the meta-model) and the notation of the language. It also derives opinions about a number of different quality types, including physical, empirical and syntactic quality.

The Chart of Concepts (Castellani, 1998) and the Method Points methodology (McLeod, 1997) were used to analyse and compare issues around the complexity of the methodology, such as ease of adoption and use. The Chart of Concepts approach was originally used to derive measures of size and complexity of UML, both in and between models (McLeod and Roeleveld, 2002, p. 2), and the Method Points methodology is described by McLeod and Roeleveld (2002, p. 2) as being akin to the Function Points count, and provides "a quantitative measure of complexity by counting the components of the generic structure of a method's deliverables". A number of separate criteria were also defined, but were later discarded due to a low response rate in a survey to rate them (McLeod and Roeleveld, 2002, p. 1).

While the approach proposed by McLeod and Roeleveld (2002) covers a number of aspects, one issue that could be raised is that there is a strong emphasis on the modeling capability of the methodology, specifically including the notation. While this criterion would be valid when comparing methodologies with distinct notations, it could be invalidated when comparing methodologies that use a standardised or common notation (such as UML, which is used by many Object Oriented methodologies, including the Rational Unified Process and is also often used in conjunction with agile methodologies) or by methodologies that do not prescribe a notation, such as Extreme Programming.

Abrahamsson, Warsta, Siponen and Ronkainen (2003)

In the study by Abrahamsson *et al.* (2003), a qualitative, comparative analysis of methodologies based on a specific paradigm, namely agile methodologies, was performed. The software development methodologies compared included Adaptive Software Development (ASD), Agile Modeling, the Crystal family of methodologies, Dynamic Systems Development Method (DSDM), Extreme Programming, Feature Driven Development, Internet-speed Development, Pragmatic Programming and Scrum. A framework consisting of five perspectives, or lenses, was used for the methodology analysis (Abrahamsson *et al.*, 2003, p. 247). These perspectives are as follows (*Table 2*):

Perspective	Description
Software Development Life Cycle	Which stages of the SDLC does the software development methodology cover?
Project Management	Does the software development methodology support project management activities?
Abstract Principles vs. Concrete Guidance	Does the software development methodology rely on abstract principles, or does it offer concrete guidance?
Universally Defined vs. Situation Appropriate	Is the software development methodology argued to fit *per se* in all agile development situations?
Empirical Evidence	Does the software development methodology have empirical support for its claims?

Table 2: Abrahamsson et al.'s Framework (2002, p. 247)

A point raised by Abrahamsson *et al.* (2003, pp. 251, 252) is that the comparison is based on methodologies whose usage is growing substantially, resulting in more, and more academic, research on them. As a result, the data available for perspectives such as empirical evidence and concrete guidance could grow, thereby rendering the results invalid except as a 'snapshot' view at the time of publication.

Singh and Kotze (2003)

Singh and Kotze (2003) evaluated methodologies based on a framework that concentrates on the human components, and specifically user involvement and human computer interaction (HCI) considerations, during the phases of the software development life cycle. Singh and Kotze (2003, pp. 39, 40) state that their evaluation methodology involved two steps, the first of which involved breaking down the selected software development methodologies into their process steps (i.e. planning, analysis, design and implementation in the case of Dennis and Wixom, 2000).

The second step involved determining whether each phase of the software development methodology took certain stakeholders into account and if so, how substantially. The stakeholders included User Interface Component, Internal Users, Customers, Suppliers, IT Department and Government (Singh and Kotze, 2003, p. 44). A qualitative value was given for each result, ranging from "No" or "Not Part Of", to "Yes" or "Actively Part Of" (Singh and Kotze, 2003, p. 44). Singh and Kotze (2003, pp. 39, 40) also argue that:

- Their evaluation methodology did not aim to compare terminology, and instead focused on how the compared methodologies addressed the elements of human interaction/stakeholders.
- By not comparing specific versions of methodologies, and instead general steps in processes, they avoided possible restrictions in the information reviewed.
- By not quantifying criteria, they attempted to remove bias from their evaluation.
- By avoiding overly-constricting definitions of the term methodology, they were able to determine whether the criteria for evaluation was addressed in generic terms.

The evaluation methodology proposed by Singh and Kotze (2004) is too specialised an approach for use in a generalised comparison of methodologies, as it concentrates on a specific aspect of the methodologies compared, namely the involvement of certain stakeholders. This is an important aspect, and one that has been emphasized by agile methodologies such as Extreme Programming, and is an aspect that should be taken into account in a framework intended for more general comparison.

Van Belle (2003)

Van Belle (2003) proposes a framework for the analysis and evaluation of enterprise models (described below in *Table 3*) which uses both absolute, or theoretical, and relative, or applied, measures (Van Belle, 2003, p. 92). These measures are in turn evaluated on the basis of criteria, organised in terms of their nature, which can be syntactic, semantic or pragmatic.

This framework, while intended for the analysis and comparison of enterprise models, has a wide range of criteria, many of which could be applied to the analysis and comparison of methodologies. Van Belle (2003, p. 243) specifically suggests that, "as a prime example of self-referential or meta-analysis", it should be possible to use the framework to "categorise and compare other different IS frameworks, software and other architectures or system development methodologies". It would, however, be necessary to use measures such as meta-models to compare the syntactic aspects of the methodologies.

	Syntactic	***Semantic***	***Pragmatic***
Absolute	Correctness / Error-Free Integrity / Consistency Conciseness / Efficiency Modularity / Structuredness Hierarchy	Genericity (universality and technical independence) Completeness (coverage of the domain) Expressiveness Similarity with other models	Validity (authority and user acceptance) Flexibility / Expandability / Portability / Adaptability
Relative	Complexity Architectural Style	Perspicuity / Comprehensibility / Understandability / Self descriptiveness Documentation	Purpose / Goal Appropriateness / Relevance Price Availability / Support

Table 3: Van Belle's Framework for the Evaluation of Enterprise Models (2003, p. 92)

Boehm and Turner (2004)

Boehm and Turner (2004) propose a risk-based decision framework intended to allow users of the framework to choose between agile and plan-driven methodologies. Agile methodologies include methodologies such as Extreme Programming and Scrum which "promise higher customer satisfaction, lower defect rates, faster development times and a solution to rapidly changing requirements" while plan-driven methodologies, such as Cleanroom, PSP, or CMM-based methods, are defined as methodologies which promise "predictability, stability, and high assurance" (Boehm and Turner, 2004, p. 1).

This framework is described in the following table (*Table 4*). While Boehm and Turner's work (2004) does not evaluate or compare methodologies, the decision framework is valuable for its risk-based approach, as it highlights how some methodologies address the issues of risk and uncertainty. Another interesting aspect is that it proposes the architecting of the methodology application to encapsulate both agile and plan-driven aspects into the methodology where needed (Boehm and Turner, 2004, p. 2).

Step	Task
Step 1	Rate the project's environmental, agile, and plan-driven risks. If uncertain about ratings, buy information via prototyping, data collection, and analysis.
Step 2a	If agility risks dominate plan-driven risks, go Risk-based plan-driven.
Step 2b	If plan-driven risks dominate agility risks, go Risk-based agile.
Step 3	If parts of the application satisfy 2a and others 2b, architect the application to encapsulate the agile parts. Go risk-based agile in the agile parts, and risk-based plan-driven elsewhere.
Step 4	Establish an overall project strategy by integrating individual risk mitigation plans
Step 5	Monitor progress and risks/opportunities, readjust balance and process as appropriate.

Table 4: Boehm and Turner's Decision Making Framework (2004, p. 2)

Matinlassi (2004)

Matinlassi (2004) provides a comparison of software product line (SPA) architecture design methodologies, specifically focusing on the COPA, FAST, FORM, KobrA and QADA methodologies. The interesting aspect of this research is that it uses an adapted version of Jayaratna's NIMSAD framework, which is described earlier in this chapter (Matinlassi, 2004, p. 2). The framework was adapted to include aspects such as the definition of the methodology and its elements, and the method contents (Matinlassi, 2004, p. 2). Method ingredients are described by Kronlof (1993, in Matinlassi, 2004, p. 2) as:

- An underlying model.
- A language.
- Defined steps and ordering of these steps.
- Guidance for applying the method.

Matinlassi (2004, p. 2) added tools to these ingredients, as they "help in execution of the methods". The intention of this research was not to rate the methodologies, but to provide an overview of the methodologies and to determine how the methods differ (Matinlassi, 2004, p. 2). The final framework, elements and questions were as follows (*Table 5*):

Category	Elements	Questions
Context	Specific goal	What is the *specific* goal of the method?
	Product line aspect(s)	What aspects of the product line does the method cover?
	Application domain(s)	What is/are the application domain(s) the method is focused on?
	Method inputs	What is the starting point for the method?
	Method outputs	What are the results of the method?
User	Target group	Who are the stakeholders addressed by the method?
	Motivation	What are the user's benefits when using the method?
	Needed skills	What skills does the user need to accomplish the tasks required by the method?
	Guidance	How does the method guide the user while applying the method?
Contents	Method structure	What are the design steps that are used to accomplish the method's specific goal?
	Artifacts	What are the artifacts created and managed by the method?
	Architectural viewpoints	What are the architectural viewpoints the method applies?
	Language	Does the method define a language or notation to represent the models, diagrams and other artifacts is produces?
	Variability	How does the method support variability expression?
	Tool support	What are the tools supporting the method?
Validation	Method maturity	Has the method been validated in practical industrial case studies?
	Architecture quality	How does the method validate the quality of the output it produces?

Table 5: Matinlassi's Framework for the Evaluation of SPA Methodologies (2004, p. 3)

In conclusion, this chapter has attempted to provide an overview of the literature relevant to the study of methodology evaluation. The next chapter details the research methodology to be followed and proposes a framework for the analysis and evaluation of software development methodologies.

Chapter 4: Research Methodology

Having reviewed the relevant literature in the previous chapter, the research methodology will now be discussed. This chapter uses the word methodology with two different meanings, namely software development and research. However, the intention is to ensure that any possible confusion resulting from this will be restricted to this chapter. This chapter consists of the following sections:

- *Strategy*, the research philosophy and strategy to be followed in this research.
- *Proposed Framework*, a discussion of the framework proposed by this research for the analysis and evaluation of software development methodologies.
- *Formal, Intrinsic and Pragmatic Criteria*, a description of the criteria that make up the proposed framework.
- *Sample Data,* a discussion of the data that will be used as the basis of the actioning and validation of the proposed framework.
- *Data Capture, Preparation and Analysis*, how the data discussed in *Sample Data* above will be captured, prepared and analysed during the actioning and validation of the framework.

Strategy

The framework for the present research is a qualitative, theory-building research philosophy, within which reference literature and pragmatic information will be used as the research data. Attributes of the research data (methodologies) will be analysed, compared and evaluated on the basis of formal, intrinsic and pragmatic analysis methods in accordance with the model proposed by Van Belle (2003). In addition, the meta-notation for analysing methodology processes, and the resulting methodology process models, as well as the notations suggested for use with the methodologies, will form the basis for the formal analysis of the methodologies.

In terms of the intrinsic and pragmatic analysis, Van Belle (2003) suggests a number of criteria and strategies with which to analyse model attributes and, while some may be appropriate, other criteria will be used in the place of those that are not appropriate for methodology analysis. The analysis will, however, focus on the reference literature as sample data for each of the methodologies analysed. As an example of methodology analysis for a single criterion, Abrahamsson *et al.* (2003) measure methodology completeness by determining how much of the traditional software development life cycle is covered by the methodologies that they compare, where the methodology reference literature was used as the research data.

Proposed Framework

Van Belle's framework for the analysis, comparison and evaluation of enterprise models (2003) suggests a number of evaluation variables for enterprise models, and is described in *Table 3* in the previous chapter. This framework will be used as an organising structure for this framework. However, some criteria described in the above framework are not suitable for the evaluation of

methodologies, such as *Genericity*, *Similarity with Other Models*, *Architectural Style*, *Portability/Adaptability and Purpose/Goal*. An important aspect to note is a change in the name of the criteria views. These views are specified by Van Belle (2003) as *Syntactic*, *Semantic* and *Pragmatic* and, while the *Pragmatic* view is not relabelled in this research, the Syntactic and Semantic views are thus renamed:

- *Formal*, to reflect the intention of Hong *et al.* (1993) who use methodology process models as a basis for formal analysis and evaluation.
- *Intrinsic*, to reflect the need to analyse those aspects of methodologies which are "inherent [or] essential" (Upshall, 1992, p. 406).

Architectural Style is, according to Van Belle (2003, p. 95), a relative measure involving the subjective evaluation and weighting of stylistic features of the form, shape or structure of a model. While this criterion could be considered useful and pertinent in terms of evaluating models where the majority of models are represented as a graphical product (see the graphical representations in Van Belle, 2003, p. 95), this is not the case with respect to methodologies. While many methodologies may have a graphical component, or notation, the evaluation of the architectural style of the notations would not take into account the other vital aspects of the methodology, such as processes. As a result, this criterion has not been included in the evaluation criteria.

A number of criteria have been added to the framework to make it more comprehensive and fitting for the evaluation of methodologies. Some formal and intrinsic changes have also been made to reflect aspects that are more closely related to methodologies, as opposed to models. The changes are shown in the table below (*Table 6*).

	Formal	***Intrinsic***	***Pragmatic***
Criteria	Process Model Size and Complexity	Paradigm, Process and Notation	Situationality
	Life Cycle Coverage/ Granularity	Completeness	Flexibility / Expandability / Enhanceability
	Process Roles and Inter-role communication	Best Practice Coverage	Effectiveness
	Modularity / Structuredness / Hierarchy	Support for requirements change	Validity (authority and user acceptance)
		Verification and validation tools/techniques	Availability / Price
		Project Management support	Support
		Documentation Quality	Currency / Maturity

Table 6: Proposed Framework (adapted from Van Belle, 2003)

The main changes to Van Belle's framework (2003) are as follows:

- The *Absolute/Relative* labelling has been removed because, as will be shown later, all criteria, bar *Paradigm, Process and Notation*, are more suited to relative evaluation as opposed to absolute description.

- *Process Model Size* and *Complexity* have been combined for reasons discussed below.
- *Paradigm, Process and Notation; Life Cycle Coverage*, *Process Roles and Inter-role Communication*, *Verification and Validation Techniques*, *Support for Requirements Change*, *Best Practice Coverage* and *Management Support* have been added for reasons described below.
- *Situationality* has been used to replace *Genericity* and *Portability/Adaptability* as a intrinsic criterion.
- *Effectiveness* has been moved from Van Belle's syntactic criteria (2003) to pragmatic criteria.
- *Perspicuity/Comprehensibility/Understandability/Self-descriptiveness*, *Expressiveness, Similarity with Other Models*, *Architectural Style*, and *Purpose/Goal* have been removed as they are not applicable.

These criteria which make up the proposed framework are defined and discussed below.

Formal Criteria

- *Methodology Process Model Size and Complexity:* Size is included as an evaluation criterion as it is applicable to "almost any construct, whether conceptual or physical" and, as there is a close association between size and complexity, the Size and Complexity criteria have been combined (Van Belle, 2003, p. 124). According to The Object Agency (TOA) (1995, p. 48), methodology complexity directly relates to the amount of skills and training required to use a methodology effectively and efficiently. Complexity is a relative measure, and that complexity is an important indicator of quality in that neither overt simplicity nor overt complexity is desirable (Van Belle, 2003, p. 95).
- *Life Cycle Coverage/Granularity:* This is an absolute criterion, relating to the number of elements per life cycle phase, which can be used to show which life cycle phase(s) the methodology places emphasis and is the formal equivalent of one of the Completeness criteria detailed later.
- *Process Roles and Inter-Process Role Communication:* Jayaratna (1994, p. 69), Avison and Fitzgerald (1995, p. 446) and McLeod and Roeleveld (2002, p. 11) consider the process roles played in the methodology an important evaluation criterion. This criterion will use the process roles defined in the methodology process models for each methodology to identify and count the process roles, both inter-project team and extra-project team, and the communication relationships between them. The communication relationships between process roles are identified as an important criterion by McLeod and Roeleveld (2002, p. 12), Catchpole (1982, in Avison and Fitzgerald, 1995, p. 437) and Bjorn-Anderson (1984, in Avison and Fitzgerald, 1995, p. 438).
- *Modularity/Structuredness/Hierarchy:* Modularity directly affects the ease with which methodologies can be implemented and changed. Van Belle (2003, pp.123-128) measures

both groups and diagrams, as well as the depth of inheritance trees (process trees could be measured in the same way), in determining their modularity, structuredness and hierarchy.

Intrinsic Criteria

- *Paradigm, Process and Notation:* These aspects have been identified as fundamentally important to the composition of methodologies by Alhir (1998:35), Microsoft (2003, p. 2) and Booch (1994, pp. 23, 171) and, as such, will be used as an analysis criterion, from an absolute perspective. While Notation is concerned with Formal aspects of the methodology, it is included in this criterion as a descriptive measure.
- *Completeness:* Completeness refers to the methodology's scope of coverage of the traditional software development life cycle (Abrahamsson et al., 2003, p. 247). While the use of the traditional SDLC may be considered to bias this criteria towards methodologies based on the structured paradigm, this approach has been used on methodologies based on the agile paradigm (by Abrahamsson *et al.*, 2003) and the OO paradigm (by TOA, 1995), as well as in an aparadigmatic setting (by Kitchenham, 1996).
- *Best Practice Coverage:* This criterion refers to how the best practices suggested by the methodologies analysed compare to a set of best practices derived from industry research.
- *Support for Requirements Change:* Changes to system requirements and design should be identifiable and actionable, as early as possible in the project process, as costs associated with change tend to increase as the development progresses (Avison and Fitzgerald, 1995, p. 436). As such, this criterion is important in the context of this research, as it intends to measure how changes to requirements and design are identified and actioned during the methodology process.
- *Verification and Validation Techniques:* This criterion relates to the tools and techniques made available by the methodologies analysed for verification and validation of the process as they are considered "desirable attributes of a well-defined methodology" (TOA, 1995, p. 10).
- *Project Management Support:* McLeod (1992) states that a methodology's support for project management activities such as planning, estimating, tracking, control and quality assurance are vital, and that there is a need for "a common management approach to these, so that skills can be transferred across project types, and so that reporting and judgement can be consistent". A second aspect to this is a consistent management view, so that, from the perspective of management, all projects have a similar look and feel, allowing comparisons and the accumulation of project management expertise (McLeod, 1992).
- *Documentation Quality:* This criterion refers to the quality, understandability and availability of the documentation accompanying the methodology (Van Belle, 2003, p. 178).

Pragmatic Criteria

- *Situationality: Situationality* (Avison and Fitzgerald, 1995, p. 452) is concerned with how applicable a methodology is to various types of organisations, problems and types and sizes of projects. While this criterion is similar to *Genericity* and *Appropriateness and Relevance*, it is a term used by authors such as Arni-Bloch (2005) and has become widely-used in the field of method engineering.
- *Flexibility/Expandability/Enhanceability:* McLeod (1992) states that full methods are often large and unwieldy. As a result, qualities of a methodology which allow only tasks and deliverables relevant to the current project to be extracted from the methodology and, if necessary, customised are very valuable. As long as this is achieved without a "loss of integrity and respecting dependencies" (McLeod, 1992). As such, the extent to which a methodology allows this would be the extent to which it is flexible, expandable and enhanceable.
- *Effectiveness:* Kelley and Rogers (1992, p. 177) states that a methodology's power, or effectiveness, is its ability to increase the productivity of the individuals to complete the project, and that this effectiveness is based on the methodology's capacity to manage life cycle phases, activities, roles and products.
- *Validity:* Van Belle (2003, p. 208) describes two types of validity that could be used to describe a methodology's validity, namely face validity and authoritative validity. Face validity refers to the acceptance of a methodology by practitioners in a field, while authoritative validity refers to the authority of the developers of the methodology. Van Belle (2003, p. 208) cautions that a problem with measuring authoritative validity is that, "in the absence of contrary information", the product of a well-respected academic research group or software company will be rated higher than and better accepted than that of lesser-known sources, which is "contrary to the established scientific practice of objectivity", but that it is a very real and pragmatic criterion used in the commercial world.
- *Availability/Price:* This criterion is largely concerned with the medium and state of availability of the methodology (Van Belle, 2003, p. 215), such as whether it is available in book or electronic format, and how it is made available. The majority of the methodologies used in the research are available in book format, and thus at a relatively low cost. While cost is not a pragmatic issue for the majority of methodologies used in this research (with the exception, perhaps, of the Rational Unified Process), it could be considered a valid evaluation criterion if commercial methodology products were being evaluated, and has thus been included.
- *Currency/Maturity:* This criterion relates to how well-updated the methodology is and how often changes and updates are made (Van Belle, 2003, p. 220). This is an important criterion as changes to the technology or platform of a methodology should be reflected in changes or updates to the methodology.

- *Support:* Van Belle (2003, p. 224) argues that three dimensions of support should be considered: product or tool support, vendor support and user base. Tool support refers to the tools that exist to assist in the use of the methodologies, such as CASE tools (TOA, 1995, p. 52), while vendor support refers to whether the author or supplier offers any assistance in using the methodology. User base, according to Avison and Fitzgerald (1995, p. 457) refers to the number and type of users of the methodology.

A Short Overview of the Methodologies Being Evaluated

Six methodologies were selected for evaluation in this research (reasons for selection are provided in the next section *Sample Data)*, specifically:

- Feature-Driven Development (FDD)
- The Microsoft Solutions Framework (MSF)
- Object-Oriented Analysis and Design (OOAD)
- The Rational Unified Process (RUP)
- Structured Systems Analysis and Design (SSADM)
- Extreme Programming (XP)

Feature-Driven Development

Feature-Driven Development (FDD) is considered an agile methodology and was introduced by Jeff De Luca on a large software development project in Singapore in 1997 with input from Peter Coad (Palmer and Felsing, 2002, p. xxi). Palmer and Felsing (2002, pp. 56, 57) define a *feature* as "a small, client-valued function expressed in the form: <action> <result> <object>" and state that FDD can be explained as follows:

- FDD starts with the creation of a domain object model in collaboration with Domain Experts.
- Using information from this activity and other requirements activities, a feature list is created.
- At this point a rough plan is created and responsibilities assigned.
- Finally, small groups of features are taken through a series of short design and build iterations until all features are complete.

The Microsoft Solutions Framework

The Microsoft Solutions Framework (MSF) was first introduced in 1994 by Microsoft and is a commercial methodology product based on a mix of the structured and agile paradigms (Microsoft, 2003, pp. 3, 4). The MSF consists of 5 phases, specifically (Microsoft, 2003):

- Envisioning, which involves the definition of project scope and vision and the project organisation.

- Planning, which involves the conceptional, logical and physical design process and the development of the functional specification.
- Developing, which involves building and testing the specified solution.
- Stabilising, which involves piloting the solution in preparation for production release.
- Deploying, which involves deploying the solution and ensuring that it is stable and usable.

Object-Oriented Analysis and Design

Object-Oriented Analysis and Design (OOAD) was proposed by Grady Booch in *Object-Oriented Analysis and Design with Applications* in 1992 and is a methodology based on the object-oriented paradigm. OOAD consists of 5 process phases, which can be performed iteratively, (Booch, 1994, pp.235, 249) based around two processes, the Macro and Micro processes. The Macro phases are concerned with the project life cycle as a whole, and the Micro phases are specifically concerned with low-level construction aspects of the system being developed.

The Rational Unified Process

The Rational Unified Process (RUP) was developed by Rational Software in 1998 and is an object-oriented methodology based on OOAD and the Objectory process (IBM, 2003). The RUP is a customisable methodology product, based on an iterative process consisting of four phases, namely (IBM, 2003):

- Inception in concerned with defining the project objectives and scope.
- Elaboration is concerned with developing and baselining the system architecture to provide a stable basis for the bulk of the design and implementation effort in the construction phase.
- Construction is concerned with clarifying the remaining requirements and completing the development of the system based on the baselined architecture.
- Transition is concerned with ensuring that the system is available for its end users.

Structured Systems Analysis and Design

Structured Systems Analysis and Design (SSADM) is a structured, data-driven methodology developed by the United Kingdom government in 1982 for the development of information systems by government departments, but which has since been made an open methodology and is widely used (Goodland and Slater, 1995, p. v). Since 1982, SSADM has been regularly updated and is now available in version 4 (Goodland and Slater, 1995, p. v). SSADM does not cover development and implementation, focusing only on analsys and design, although an optional feasibility stage is proposed (Goodland and Slater, 1995, p. 6).

Extreme Programming

Extreme Programming (XP) was developed by Kent Beck and is one of the most prominent of the agile methodologies. According to Beck (1999, p. 31), XP is a lightweight, iterative methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirement, and stresses four key values, namely namely communication, simplicity, feedback and courage. XP consists of five phases, namely (Beck, 1999):

- Exploration, which encompasses initial requirements and architectural modelling.
- Planning, which involves the structuring of iterations for construction.
- Iterations to Release, which involves the major development activities of modelling, programming, testing and integration.
- Productionising, which encompasses acceptance testing and the deployment of small releases.
- Maintenance, which involves the evolution, or updating, of the system over time.

Sample Data

The sample data used in this research will be a number of software development methodologies. These will be chosen based on a number of criteria such as the author's experience, availability of reference material and perceived user base and popularity. Data for the specific aspects of the methodology to be analysed and evaluated will be gathered from the methodology reference material (for Formal and Intrinsic criteria) and external sources (for Pragmatic criteria). For the purposes of this research, reference literature will be considered the literature that first described the methodology, such as *Extreme Programming Explained: Embrace Change* (Beck, 1999), or literature that is held in high standing by the users of the methodology, for example *A Practical Guide to Feature-Driven Development* (Palmer and Felsing, 2002).

In some cases, for example, the Extreme Programming methodology, there exists a reasonably sized pool of reference literature (a number of books are in print and are considered reference literature), whereas for others, such as Feature-Driven Development, the amount of reference literature is significantly smaller, being only one or two books in print. However, in the case of the Rational Unified Process, IBM provides an electronic version of the Rational Unified Process as an evaluation version, which will be used in conjunction with other reference literature.

The full list of reference material is as follows (*Table 7*):

Methodology	Reference Material Title	Author(s)	Edition/ Version	Year of Publication
Feature-Driven Development (FDD)	*A Practical Guide to Feature-Driven development*	Stephen R. Palmer, John M. Felsing	First	2002
Microsoft Solutions Framework (MSF)	*Analysing Requirements and Defining Microsoft .Net Solution Architectures*	Microsoft	First	2003
Object-Oriented Analysis and Design (OOAD)	*Object-Oriented Analysis and Design with Applications*	Grady Booch	Second	1994
Rational Unified Process (RUP)	*The Rational Unified Process*	International Business Machines (IBM)	1.2	2004
Structured Systems Analysis and Design (SSADM)	*SSADM Version 4*	Mike Goodland, Caroline Slater	First	1995
Extreme Programming (XP)	*Extreme Programming Explained*	Kent Beck	First	1999
	Extreme Programming Installed	Ron Jeffries, Ann Anderson, Chet Hendrickson	First	2000
	Extreme Programming Explored	William C. Wake	First	2000

Table 7: Sample Data

Data Capture, Preparation and Analysis

A number of approaches will be used to capture and prepare the data, based on the type of analysis being performed. Firstly, each methodology to be used as data will be mapped to a metamodel identified to describe methodology processes. A number of these models exist and are described in more detail in the following chapter. Second, the methodologies to be used as data will be analysed using intrinsic criteria, based on the reference literature for the methodologies.

The intrinsic criteria are detailed in the framework adapted from Van Belle (2003) and include criteria such as Completeness and Genericity. Finally, pragmatic aspects of the methodology will be captured, based on both the reference literature for the methodologies as well as other sources of pragmatic data such as Google PageRank[tm] and Amazon.com sales figure, and methodology prices.

In terms of the data analysis, the following table describes the approaches to be used to analyse and compare the methodologies for the criteria proposed above. The intention is to use Van Belle's framework (2003) as a structuring method and add, augment and substitute metrics from previous research where necessary.

Criterion	Measure
Process Model Size	A count of the Phases, Activities, Object Flows, Artefacts and Process Roles (elements) in the derived activity diagrams will be used.
Correctness and Integrity	As stated previously, this criterion will only be included if any inconsistencies are identified in the meta-modeling process.
Complexity	The following measures of complexity were used by Van Belle (2003, pp.129), and will be used in this research: • Connectivity, which takes into account the number of object flows (absolute), and the number of object flows to number of elements (relative). • System complexity, or coupling between elements, which is the number of non-inheritance-related couples between elements. • Fan-out, or the average number of object flows in which an element participates. • Grouper and diagram metrics, as suggested by Van Belle (2003, pp. 123-128), will be used to show the level of connectivity between the phase, activity, process role and artefact elements, and show the connection ratios between the elements. • Graph size, connectivity density, depth and width metrics (Van Belle, 2003, p. 126), as defined in the methodology process models, will be used to calculate inheritance metrics on the main process trees defined in the methodology process models.
Modularity/Structuredness/ Hierarchy	This criterion will be measured by mapping the model groups and entities described by Van Belle (2003, p. 130) to the Phases and elements as described by the OMG (2005, pp. 3-2, Glossary 1-2) and using the same technique as described by Van Belle (2003).
Roles and Inter-Role Communication	A count of the number of process roles, as well as a count of the number of object flows between the process roles will be used to measure this criterion.
Paradigm, Process and Notation	This criterion will focus on describing the aspects of the methodologies, as described in the methodology reference material, such as: • Paradigm; • Process, including process phases; and • Notation. No comparative techniques will be used.

Criterion	Measure
Completeness	The measures proposed by Abrahamsson et al. (2003) will be used to determine the coverage of the software development life cycle and how any associated processes (such as project management) support that coverage. An additional measure, such as methodology coverage of the Zachman framework will be used, as suggested by Van Belle (2003, pp.227-229).
Best Practice Coverage	The superset creation approach suggested by Hong, Van Den Goor and Brinkkemper (1993, p. 693) will be followed, using sets of best practices from the methodologies being analysed and other sources of best practices. From this a supermethodology will be created, consisting of the smallest common denominator of all methodological best practices. Once the supermethodology has been created, the individual methodologies will be compared against this supermethodology in terms of coverage.
Support for Requirements Change	The approach followed by TOA (1994, p. 54), described earlier, as well as a listing and discussion of the techniques available to manage these changes, will be used to determine how effectively methodologies support changes in requirements and design.
Verification and Validation Techniques	The approach followed by TOA (1995, p. 54) to measure method flexibility will be used to determine a methodology's verification and validation techniques. The measures involve determining, from the reference literature: • The level of discussion of verification and validation techniques. • Whether a process was described to enable verification and validation techniques. • Whether heuristics were provided for this process. • Whether examples were provided.
Management Support	The approach followed by TOA (1994, p. 54), which was described earlier, as well as a listing and discussion of the techniques available to measure project progress will be used to measure a methodology's level of support for management.
Documentation	The quality of documentation can be evaluated using the degree of documentation completeness for book-based methodologies as described by Van Belle (2003, p. 178), as well as through the use of readability indices (such as the Flesch Reading Ease Score and the Flesch-Kincaid Grade Level Score) for random excerpts of text from the reference literature, which Van Belle (2003, p. 181) states can be used as measures of documentation quality.
Situationality	This criterion will attempt to approach the analysis of methodological situational applicability from as descriptive a perspective as possible, discussing the aspects that Fitzgerald (1998, pp. 326, 327) states are

Criterion	Measure
	relevant to the adherence to methodologies, namely: • Project size/scale. • Project type. • Project system.
Flexibility/Expandability/ Enhanceability	The measures used by TOA (1995, p. 54) will be used to determine method flexibility. The measures involve determining, from the reference literature: • The level of discussion of methodology flexibility. • Whether a process was described to enable methodology flexibility. • Whether heuristics were provided for this process. • Whether examples were provided. Some of the criteria described by Van Belle (2003, p. 221), such as whether the methodology is available in digital format, and the level of implementation independence, will also be used and can also be determined from the reference literature. These results will be mapped against a framework for methodology tailoring, such as that suggested by Keenan (2004, p. 1) to determine the level of methodology flexibility.
Effectiveness	The NIMSAD framework (Jayaratna, 1994) will be used in the same manner as was used by McLeod and Roeleveld (2002) to evaluate effectiveness.
Validity	The measures proposed by Van Belle (2003, p. 209) will be used as a practical way to compare methodology validity. These measures can include: • Cited publications by lead author. • Google ranking. • Amazon.com book sales. • An informed estimate of methodology use, based on surveys and empirical evidence.
Availability and Cost	Van Belle's criteria for availability, size and cost (2003, pp.215-217) will be used to evaluate the methodologies. These criteria are: • Availability method. • Size, in pages/Mb. • Cost.. • Availability in digital format (yes/no).

Criterion	Measure
Currency and Maturity	The data for assessing currency and maturity of a methodology will be derived from meta-information about the relevant literature and will constitute criteria such as maturity, date last changed and whether there are updates (from Van Belle, 2003, p. 222).
Support	Van Belle (2003, p. 224) argues that three issues around support should be considered, namely product support, vendor support and estimated user base. These issues will be investigated against both the reference literature and other sources such as user forums.

In conclusion, this chapter has described the approach and research methodology that will be used, as well as a proposed framework, based on Van Belle's framework (2003) for the analysis and evaluation of software development methodologies. While it was necessary to change some of the individual criteria that made up Van Belle's framework (2003), the original intention and aim of the framework was preserved and will be used in an isomorphic setting. The following three chapters (Formal Analysis, Intrinsic Analysis and Pragmatic Analysis) will see the framework actioned and validated using the data described above.

Chapter 5: Formal Analysis

This chapter discusses the criteria composing, and evaluation using, the first view of the proposed framework, the formal view. Chapters Six and Seven will then discuss the remaining views, the intrinsic and pragmatic. The formal view is, in the same way as Van Belle's syntactic view (2003, p. 124), concerned with the purely structural aspects of the object being studied which are, in this case, methodology process models. This chapter involves a short overview of the formal criteria and a brief discussion on the use of metamodels, after which the results of the formal analysis are presented.

Formal Criteria

The criteria used in the formal view are:

- *Methodology Process Model Size*, the number of constituent entities that compose the process model of the methodology being analysed, based on the application of a selected metamodel.
- *Methodology Process Model SDLC Coverage/Granularity*, the extent to which the methodology process model covers, and is oriented to, the phases of a traditional SDLC.
- *Methodology Process Model Roles and Inter-Role Communication*, the degree to which there is communication between the process roles detailed within the methodology process models.
- *Methodology Process Modularity*, the degree to which the phases of the methodology are modular.
- *Methodology Process Model Complexity*, the level of inherent complexity within the methodology process model.

Metamodeling Approach

As discussed previously, metamodels are regularly used to describe methodologies, from both a data and a process perspective (e.g. Hong *et al.*, 1993; Gnatz *et al.*, 2001; Van de Weerd, 2005). Hong *et al.* (1993, p. 689) state that by "using the same metamodeling constructs for all methodologies", a uniform and formal representation of more than one methodologies can be obtained. This is necessary because, to be accurate and objective, methodologies "should be compared on a uniform, formal and unbiased basis" (Hong *et al.,* 1993, p. 689).

For the purposes of this research, the idea of a uniform modeling construct, as suggested by Hong *et al.* (1993, p. 689), will be used, and all methodology processes mapped against a single metamodel for methodology processes. A number of metamodels have been proposed to describe methodology processes, including those by Van de Weerd (2005, p. 22), Gnatz *et al.* (2001, p. 184) and Henderson-Sellers *et al.* (2006). Excerpts of these metamodels are shown below (*Figures 11 to 13*).

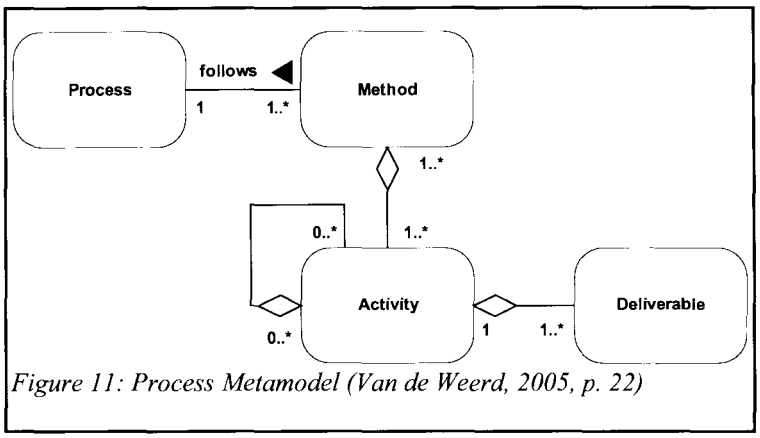

Figure 11: Process Metamodel (Van de Weerd, 2005, p. 22)

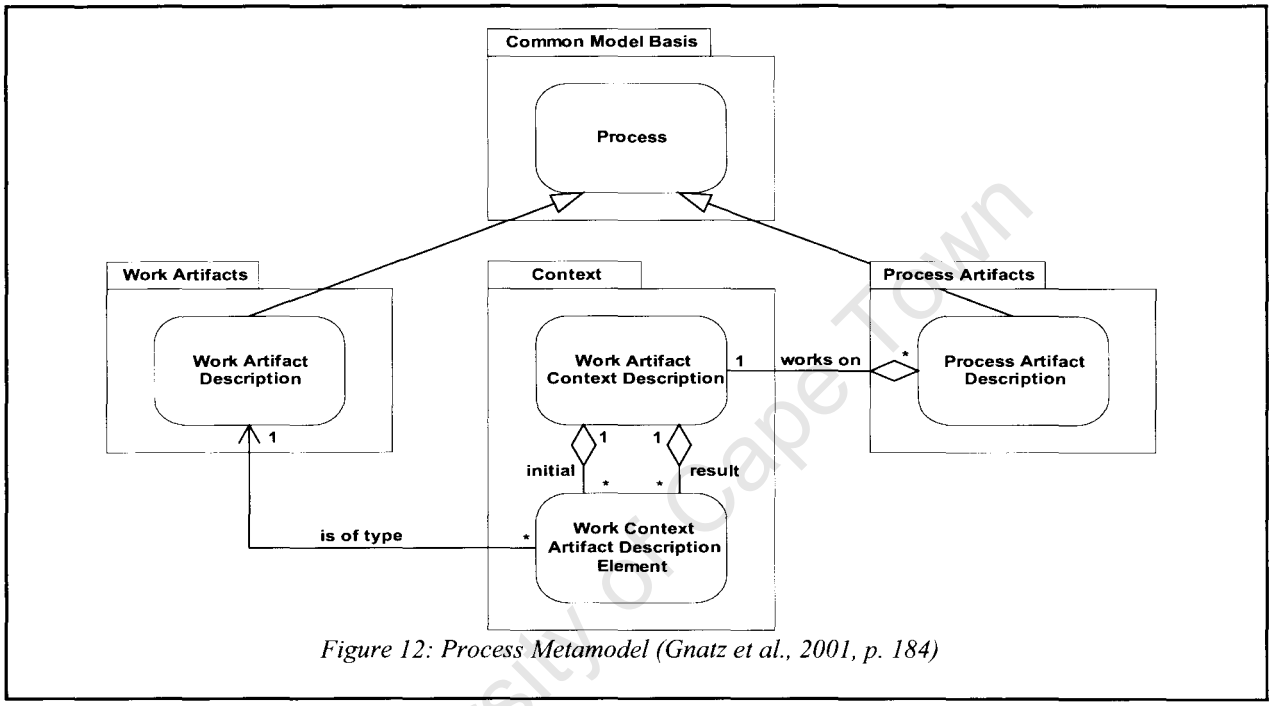

Figure 12: Process Metamodel (Gnatz et al., 2001, p. 184)

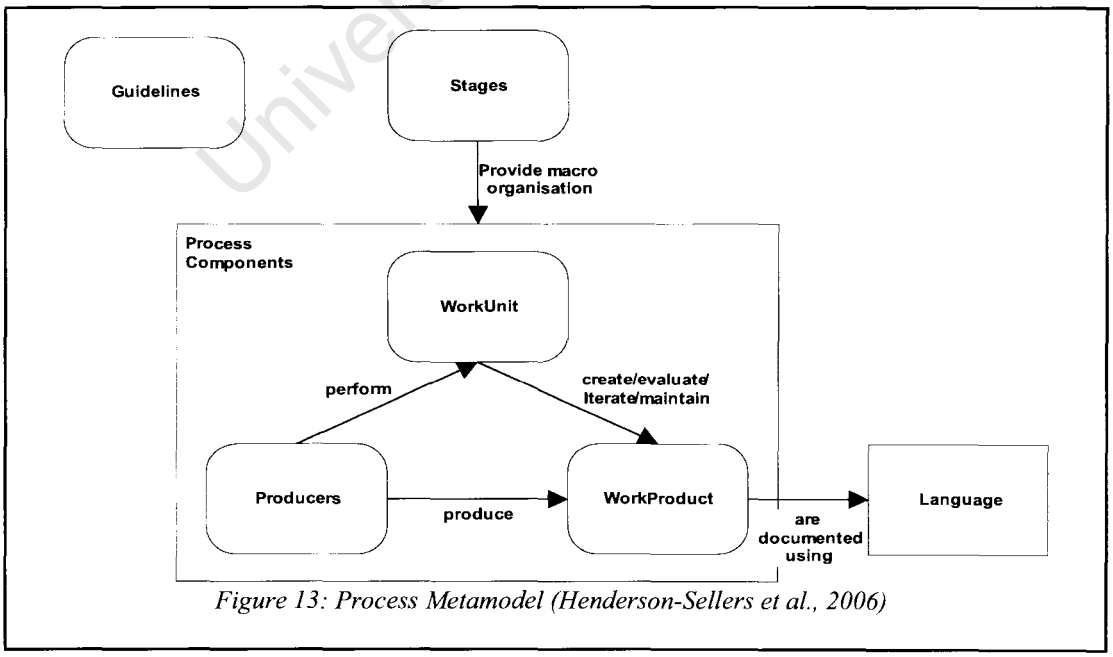

Figure 13: Process Metamodel (Henderson-Sellers et al., 2006)

However, the following metamodel was described by the Object Management Group (OMG) (2005) and was chosen to be used as the basis for the analysis (*Figure 14*):

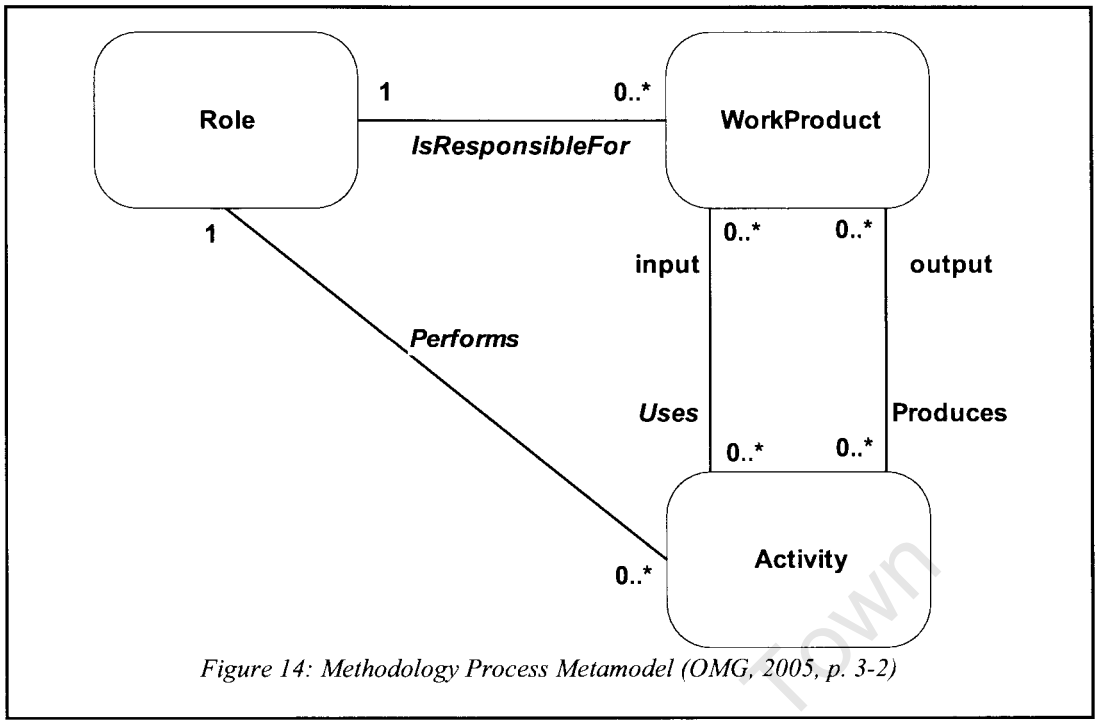

Figure 14: Methodology Process Metamodel (OMG, 2005, p. 3-2)

According to the OMG (2005, p. C-1), this metamodel can be used to instantiate methodology processes using the following structure:

```
Phase:
  Process:
    Subactivities:
      Activity:
        ProcessRole:
          WorkProduct:
          WorkProduct:
          ..
```

The OMG (2005, Glossary 1-2) provides the following definitions of the elements:

- *Activity:* A Work Definition describing what a Process Role performs. Activities are the main elements of work.
- *Phase:* A high-level Work Definition, bounded by a Milestone.
- *Process:* A Process is a complete description of a software engineering process, in terms of Process Performers, Process Roles, Work Definitions, Work Products, and associated Guidance.
- *Process Role:* A Model Element describing the roles, responsibilities and competencies of an individual carrying out Activities within a Process, and responsible for certain Work Products.

- *Work Product:* A Work Product is a description of a piece of information or physical entity produced or used by the activities of the software engineering process. Examples of work products include models, plans, code, executables, documents, databases.

Accordingly, for the analysis of the methodology processes, the methodologies were modelled using UML Activity Diagrams. UML Activity Diagrams "lie within the behavioural view of a system and render the activities or actions" of the actors participating in the behaviour (Alhir, 1998, p. 205). Activity diagrams thus allow the identification of Activities, Processes, Process Roles and Work Products, and the Object Flows between them, as described by the OMG above (2005, Glossary 1-2). The Unified Modeling Language was used as the notation, as Ledeczi *et al.* (2001, p. 1) state that UML represents a good choice for constructing metamodels. Accordingly, this analysis involved using a methodology process metamodel as a basis to model the processes. The OMG (2005) offer the example of an activity diagram (shown in *Figure 15*); the notation thereof is described in the table below (*Table 8*). The table and diagram show the following:

- The OMG representations/icons for the activity diagram elements described above (OMG, 2005, p. 11-8,9), which constitute the metamodel to be used in this research (*Table 8*).
- An example activity diagram to show how the icons described above are used in conjunction to describe a process (OMG, 2005, p. 12-6) in *Figure 15*.

Once represented as models, the methodologies were analysed on a formal level, or a level "concerned with the purely structural aspects of the model, regardless of the underlying meaning of the model and its elements" (Van Belle, 2003, p. 124). Full diagrams of the process models are included on the accompanying compact disk. In terms of model analysis, all number values relating to process model entities are counts of the respective entities from the methodology activity diagrams, unless otherwise stated (i.e. ratios, percentage values).

This section has attempted to provide brief insight into the metamodel and modeling approach used in this research. The following five sections will attempt to describe the results of the metamodel based analysis, focusing on the structure and elements of the derived activity diagrams. These sections detail aspects such as process model size, process model granularity, process model inter-process role communication, process model modularity and process model complexity.

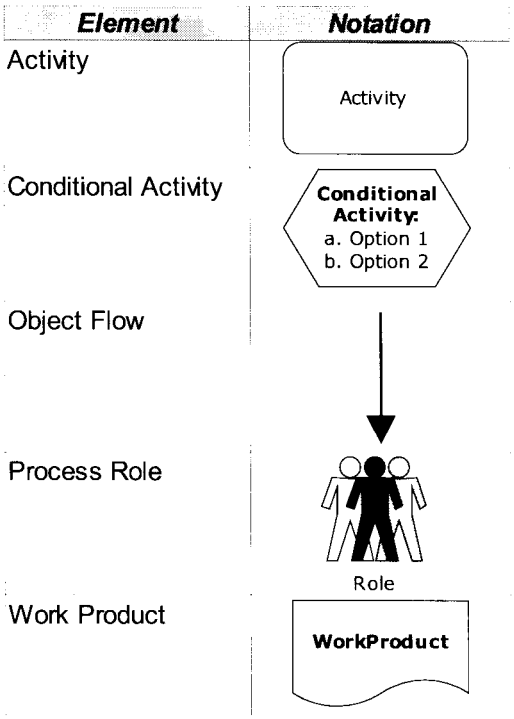

Table 8: Activity Diagram Notation (adapted from OMG, 2005, p. 11-8,9)

Methodology Process Model Size and Complexity

Van Belle (2003, p. 124) motivates for the inclusion of size as an evaluation criterion for "almost any construct, whether conceptual or physical". Methodology size can be evaluated in a number of ways, namely:

- Evaluating the physical size of the methodology, whether this is in number of pages if the methodology is available in book format, or megabytes if the methodology is available in electronic format. This approach is used by authors such as Wagner (2003) and, while included in this criterion, is expanded upon in the Pragmatic analysis of the methodologies.
- Evaluating the model size of the methodology, which requires that the methodology can be represented as a model. This approach is used by Van Belle (2003, pp. 124,125), who states that, in the context of measuring enterprise model size, the "*number of model entities* is perhaps the most intuitively acceptable count from a structural point of view". In this case, the methodology instantiations of metamodel entities can be easily counted. The following categories will thus be used as the entities counted:
  - *Activity.*
  - *Object Flow.*
  - *Phase.*
  - *Process Role.*
  - *Work Product.*

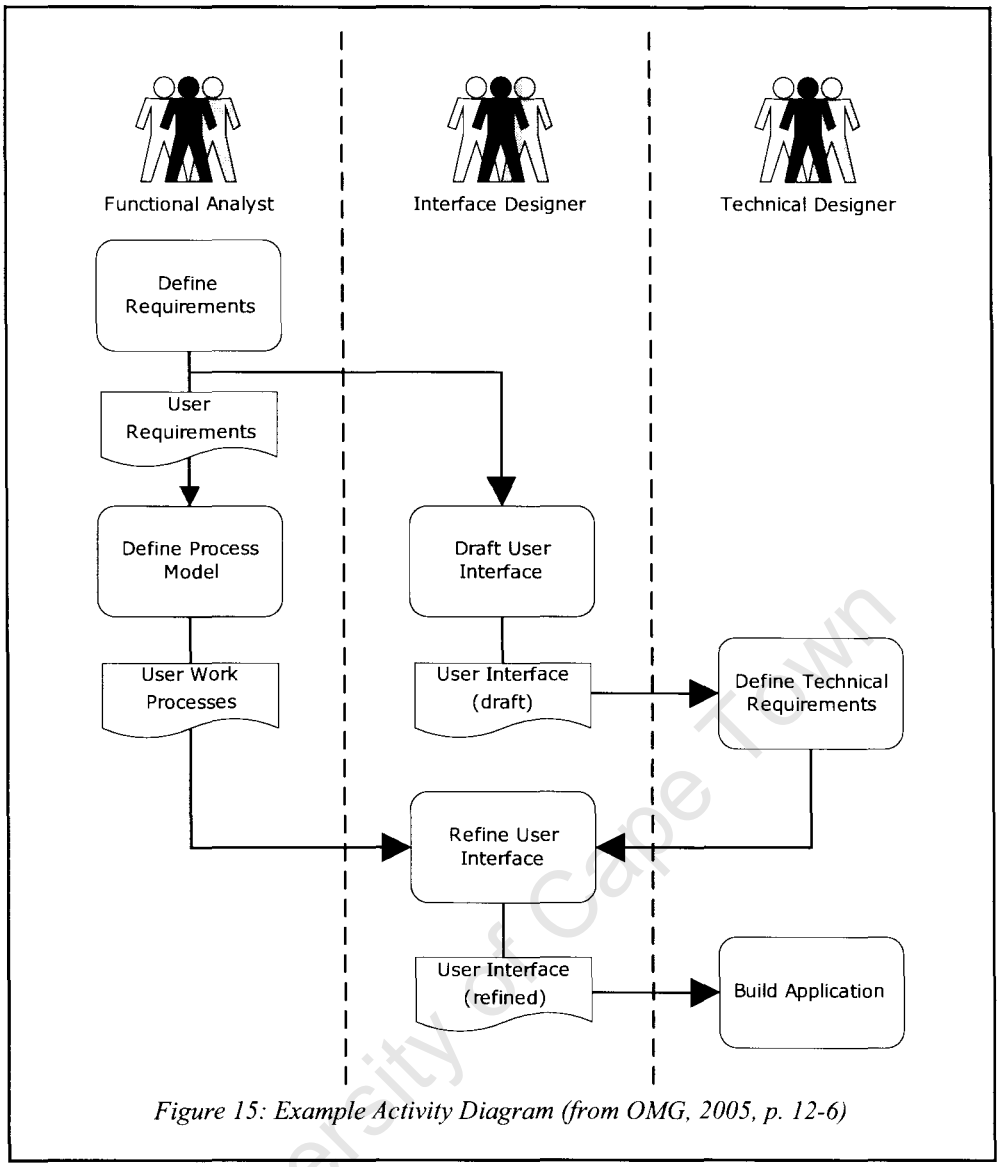

Figure 15: Example Activity Diagram (from OMG, 2005, p. 12-6)

The following table describes the sizes of the process models for the evaluated methodologies on a per-element basis. These are based on the entities identified through the representation of the methodology processes, described in the methodology documentation using the metamodels. The values shown below (*Table 9*) are the total counts of the entities existing in the methodology activity diagrams. Please note that the ***bold-italic*** format will be used to designate the highest value in a table row throughout this and the following chapters.

Element	FDD	MSF	OOAD	RUP	SSADM	XP
Process Roles	10	12	13	***20***	6	9
Phases	5	5	***9***	4	6	5
Activities	40	53	***118***	78	81	51
Activity Steps	36	31	***93***	64	72	40
Decision Steps[2]	4	22	***25***	14	9	11
Object Flows	34	74	***133***	89	95	58
Work Products	7	22	12	***48***	29	7
Total Entities	96	166	***285***	239	217	130
Physical Format	Book	Book	Book	Electronic	Book	Book
Physical Size	304 Pages	495 Pages	***608 Pages***	12 Mb Zip File	534 Pages	Between 144 and 288 pages

Table 9: Methodology Process Model Size

In terms of total entities, the largest methodology is Booch's OOAD (1994), while the smallest is Feature-Driven Development (Palmer and Felsing, 2002). The Rational Unified Process (IBM, 2004) is the second-largest methodology and SSADM the third-largest. The more agile methodologies (MSF, FDD and XP) are all smaller than the more rigid and prescriptive methodologies (OOAD, RUP and SSADM).

In a more detailed analysis of the element counts, one of the most striking differences is the number of Process Roles. The RUP has 20, or 7 more than the next-highest, OOAD. However, the RUP does not specify Team Process Roles, only Individual Process Roles, while all other analysed methodologies do. Both MSF and SSADM specify substantially more Team Process Roles than Individual Process Roles (SSADM specifies no Individual Process Roles, and MSF specifies five times as many Team Process Roles as Individual Process Roles) while the Object-Orientated methodology also specifies a larger number of Team Process Roles than Individual Process Roles (it has 60% more Team Process Roles than Individual Process Roles). Other potentially interesting aspects identified in this analysis are the following:

- OOAD has the largest number of Activities and Object Flows (251), but few Work Products (12).
- The RUP has more than double the possible Work Products than the next highest methodology (MSF), while two of the agile methodologies (XP and FDD) have the lowest number of Work Products. This is congruent with the Agile Manifesto which states that agile methodologies value working software over comprehensive documentation (Boehm, 2006, p. 19). It is important to note that, while not all of the RUP's Work Products are considered mandatory, all have been included for the purposes of this analysis.

2 Please Note: The distinction is drawn between Activity Steps and Decision Steps at this point, but will be fully described during the discussion of Methodology Complexity.

One unexpected result of this analysis was the correlation between physical size and number of entities, which is represented in *Table 10*. *Table 10* shows the following:

- Total number of methodology process model entities.
- A ranking based on number of process model entities, where the largest is ranked 1 and the smallest is ranked 6.
- The physical size of the methodology reference material, in pages or hard disk space.
- A ranking based on the physical size of the methodologies, ranked in the same way as above.
- A combined element/physical size ranking (CEP Ranking), based on the average of the element size and physical size rankings.

	FDD	***MSF***	***OOAD***	***RUP***	***SSADM***	***XP***
Total Elements	96	166	***285***	239	217	130
Physical Size	304 Pages	495 Pages	***608 Pages***	12 Mb Zip File	534 Pages	Between 144 and 288 pages
Element Size Ranking	6	4	***1***	2	3	5
Physical Size Ranking (Largest = 1, Smallest = 6)[3]	4	3	***1***	N/a	2	5
CEP Ranking[4]	5	3	***1***	N/a	2	4

Table 10: Element Size Comparisons

While this may appear to be a slightly diversionary analysis, statistical techniques, namely correlation analysis, show that there is a significant strong positive relationship between Total Entities and Physical Size with the values

- $r^2 = 0.8111$
- $p = 0.037$
- $n = 5$

Traditional statistical beliefs (StatSoft Inc., 2004) hold that if the value for p (significance, or probability of error in the hypothesised relationship) is less than 0.05 then there is a 95% probability that the relationship is truly significant. The r^2 (correlation coefficient) value explains the strength of the relationship between the two variables, or the "proportion of common variation in the two variables", where any value of r^2 over 0.75 is considered a very strong correlation. It is, however, important to note that the size of this sample is small, being only 5 sets of data.

3 Assuming 288 pages as a single XP reference.
4 The RUP was not counted, as it is in electronic format and thus cannot be easily compared to methodologies in book format.

As size is closely related to complexity (Van Belle, 2003, p. 124), methodology complexity will now be analysed. Heylighen (1996) states that a structure is complex if it is made of "closely connected parts", with mutual dependencies and an end result more than the sum of the parts. A large number of complexity measures in the sphere of software development have been proposed, such as cyclomatic complexity and fan-out.

Mannino, on the other hand, equates methodology complexity with training time, or the skill required to "successfully apply their ideas" (1987, p. 29) and believes that diagramming techniques or notations add to methodology complexity. Purao and Vaishnavi (2003, p. 216) state that cyclomatic complexity is one of the traditional measures of complexity for software programs but could, according to McCabe and Butler (1989, p. 1416), be applied to anything that is process/procedure-based and involves conditional statement or branches. In this case, it could be used to measure alternate paths based on Conditional Activities in the methodology process models. Van de Weerd (2005, p. 39) describes Conditional Activities as follows:

> *Conditional activities are activities that are only carried out if a pre-defined condition is met. This is graphically represented by using a branch. Branches are illustrated with a diamond and can have incoming and outgoing transitions. Every outgoing transition has a guard expression, the condition. This guard expression is actually a Boolean expression, used to make a choice [about the] direction to go [in]. Both activities and sub-activities can be modelled as conditional activities.*

Conditional Activities can be represented on an Activity diagram as follows (*Figure 16*):

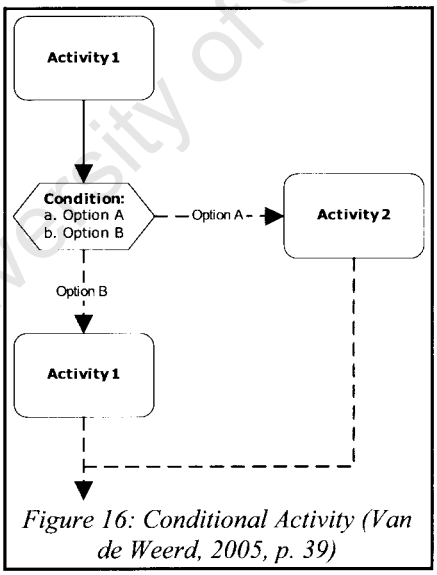

Figure 16: Conditional Activity (Van de Weerd, 2005, p. 39)

Fan-out (Olsen and Wood, 2004, p. 231; Van Belle, 2002) measures relational aspects such as the number of relationships that an object participates in, the number of procedures that a procedure calls, the number of files that a file depends on or, in the field of robotics, the number of robots that can be controlled by an individual. Fan-out relates directly to complexity, in that a high fan-out value implies high complexity (Olsen and Wood, 2004, p. 237). The fan-out metric can be applied to Conditional Activities in methodology process models in the same way that Van Belle (2003, pp.141-142) applied it to enterprise model entities, with the exception that, in this case, it will be used to

measure the maximum and average number of Object Flows which originate from Conditional Activities in methodology process models.

Other measures of model complexity are, according to Van Belle (2003, p. 129), absolute and relative connectivity. Absolute connectivity is a measure of the number of Object Flows in a model, and the relative connectivity metrics describe the ratio of Object Flows to Activities. For the purposes of this research, all references to complexity in this chapter refer to methodology process model complexity, as opposed to other, intrinsic and pragmatic complexity measures such as document readability, which will be discussed in the chapter relating to intrinsic criteria.

Criterion	*FDD*	*MSF*	*OOAD*	*RUP*	*SSADM*	*XP*
Absolute Connectivity	34	74	***133***	89	95	58
Relative Connectivity	0.94	***2.39***	1.43	1.39	1.32	1.45
Max. Fan-Out	2	2	3	3	3	***4***
Average Fan-Out	2	2	2	2.07	2.11	***2.27***

Table 11: Complexity Results

In terms of the above metrics (*Table 11*), both absolute and relative connectivity metrics show that Feature-Driven Development is the least complex of the methodologies evaluated. In terms of relative Connectivity, MSF is the most complex. The next most complex methodology, in terms of average relative connectivity is SSADM, but it is not significantly more complex than OOAD, XP or the RUP. However, it is clear that, in some cases at least, absolute connectivity is not an adequate measure of complexity. Using this metric, OOAD would be considered the most complex of the methodologies, followed by SSADM. Absolute connectivity does, however, serve to confirm that FDD is the least complex methodology analysed. The fan-out metrics, which are concerned with Object Flows, do not reinforce the Connectivity metrics. Fan-out metrics relate to Conditional Activities and are thus almost a relative value for Cyclomatic Complexity. XP has the highest fan-out values but there is no methodology that clearly has the lowest fan-out values (although OOAD appears to have the lowest average fan-out). It would appear that these metrics do not provide a comprehensive evaluation in terms of methodology complexity, as no methodology obviously emerged as more or less complex than the rest.

Another set of metrics that could be used to determine methodological complexity is that of the grouper and diagram metrics, given that they approach the issue of process model size from a relative perspective. These metrics take into account the number of methodology elements per methodology phase, and it could be argued that the larger the methodology phase (in terms of number of elements per phase), the more complex the methodology. The results of these metrics are shown in the following table (*Table 12*), but it is interesting to note that the RUP is the highest ranking (and thus arguably most complex) in all criteria, and FDD is lowest ranking (or arguably least complex) in three criteria, and second-lowest ranking in one criteria, which would imply that it is the least complex.

Average per Phase	FDD	MSF	OOAD	RUP	SSADM	XP
Phases	5	5	***9***	4	6	5
Process Roles	2	2.4	1.44	**5**	1	1.8
Activities	8	10.6	13.1	**19.5**	13.7	10.2
Object Flows	6.8	14.8	14.8	**22.25**	15.8	11.6
Work Products	1.4	4.4	1.3	**12**	4.8	1.4
All Entities	19.2	31.7	31.7	**59.75**	36.2	26
Ranking	1	4	3	***6***	5	2

Table 12: Grouper and Diagram Metrics

There is a definite case for the RUP being a complex methodology, as Wagner (2003) makes reference to the complexity of the RUP due to its size, stating that it is too big and too heavyweight. Another point to note is that the agile methodologies, FDD and XP, were the least complex of the methodologies evaluated and the only methodologies with a entities per phase value of under 30. The grouper and diagram metrics would also appear to be the most valid and telling of the complexity metrics used in this section.

Conclusion

Formal methodology size metrics, especially when combined with physical size metrics, proved an adequate measure of methodology size. The relationship between physical size and model size, despite being based on a small sample size, was an unexpected discovery and one which should be validated using a larger sample size. Methodology size, and more specifically physical size, is a slightly less adequate measure when comparing methodologies where the reference material is an electronic resource, such as the RUP, to traditional methodology reference materials, as there is no adequate way to compare these sizes.

In terms of methodology process model complexity, absolute connectivity would not appear to be an adequate metric, as it is an absolute measure, based almost entirely on methodology size. This measure contrasted to the other metrics which, based on the methodologies analysed, appeared to correspond to one another. Accordingly, they could be considered valid measures for methodology complexity. However, formal measures of complexity will not cover measures of complexity such as ease of learning and concept complexity, which would require alternative measures, probably based on intrinsic or pragmatic aspects of a methodology.

Methodology Process Modularity

According to Humphries (1997), the relevance of modularity to methodologies comes from:

- The ability to decompose complex systems into manageable parts.
- The isolation of information pertinent only to aspects or parts of the complex systems.

Sullivan *et al.* (2001, p. 100) state that the advantage of building modularity into systems or processes is that it creates "an option to invest in a search for a superior replacement and to replace the currently selected module with the best alternative discovered, or to keep the current one if it is still the best choice". Accordingly, it can be assumed that the higher the degree of modularity in methodology processes, the easier these processes can be substituted by others that offer better fit.

Modularity in methodologies is a criterion that allows for situational method engineering, or the tailoring of methodologies to fit situations (Arni-Bloch, 2005, p. 2). Van Belle (2003, p. 130) states that, in terms of modularity, most models "group similar entities in logical sub-units or chunks", for example by functional area and that this "is indicated by means of grouper constructs which are usually focused on the collection of like *entities"* and that, in terms of measuring model modularity, it "usually makes more sense to calculate the average number of entities per grouper construct and the average number of relationships per diagram". Thus, in terms of methodologies, it would appear logical to map the model groups and entities described by Van Belle (2003) to the Phases as described by the OMG (2005, pp. 3-2, Glossary 1-2). Accordingly, in order to measure methodology process modularity, the following criteria will be used:

- Number of Phases
- Number of Sub-Phases, or phases within a phase.

The following table (*Table 13*) describes a number of group and diagram metrics determined from the methodology process models. The group and diagram metrics are based on an average number of process model entities per Phase, and the methodologies are ranked from most modular (1) to least modular (6).

	FDD	***MSF***	***OOAD***	***RUP***	***SSADM***	***XP***
Phase	5	5	5	4	***6***	5
Sub-Phases	0	0	***4***	0	0	0
Total	5	5	***9***	4	6	5
Ranking	3	3	***1***	6	2	3

Table 13: Methodology Modularity/Structuredness

In terms of methodology process modularity, the following conclusions can be drawn from these results:

- The RUP is the least modular of the methodologies, as it has the least Phases and no sub-Phases.
- There is very no difference between FDD, the MSF, OOAD and XP in terms of number of Phases but OOAD is the only methodology that has sub-Phases[5], and is thus the most modular of the methodologies analysed.

Conclusion

This measure of formal methodology modularity is possibly not adequate, but the question remains regarding what could be used an adequate measure. The measures used by Van Belle were not applicable in an isomorphic setting and no measures of methodology modularity were suggested in the literature surveyed. As such, the validity of these metrics, and possibly this criteria, should be questioned, but a better measure may put a more persuasive case for this criteria.

5 As a result, for the purposes of all other Formal criteria, OOAD's sub-Phases have been considered phases.

Methodology Process Model Life Cycle Coverage/Granularity

Abrahamsson *et al.* (2003, p. 248) define a software development life cycle (SDLC) as the "sequence of processes that an organisation employs to conceive, design and commercialise a software product" and believe that using this view as a lens of analysis for software development methodologies is necessary to determine which phases of the SDLC the methodologies apply to. This view is very similar to the methodology evaluation criterion proposed by Kitchenham (1996, p. 11), namely the extent of methodology impact, which is concerned with how the effect of the methodology will be felt over a software development life cycle (SDLC). Related to this criterion, Kitchenham (1996, p. 11) states that methodology granularity refers to whether the methodology applies to "the development (or maintenance) of a software product as a whole" (low granularity) or "individual parts of the product such as modules or documents" (high granularity).

While Kitchenham (1996, p. 11) recommends that these criteria be measured using qualitative case studies or formal experiments, the approach followed by Abrahamsson *et al.* (2003, p. 248) and The Object Agency (TOA) (1995, p. 40) involves mapping aspects (in this case process model Entities) of the methodologies to a formal SDLC. TOA (1995, p. 40) state that this criterion informs methodological completeness (discussed later in the Intrinsic Analysis) as well as providing insight into the ability to "mix and match" components from one methodology with another, based on SDLC phases. According to the TOA (1995, p. 40). For the purposes of this criterion, the SDLC is that used by Abrahamsson *et al.* (2003, p. 248) and contains the following phases:

- Project Initiation.
- Requirements Specification.
- Design.
- Code (Implementation, or Construction).
- Unit Test.
- Integration Test.
- System Test.
- Acceptance Test.
- System in Use.

The results for the orientation of Activities are shown in *Table 14*. The metrics used above show the following in terms of Activities:

- XP appears to be oriented towards the Requirements Specification and Code phase, as it has more activities in these phases than in others.
- FDD appears to be oriented towards the Requirements Specification and Design phases.

- The RUP appears to be oriented towards the Project Initiation, Requirements Specification and Design phases.
- OOAD appears to be oriented towards the Design and Code phases.
- The MSF appears to be oriented towards the Project Initiation and Requirements Specification phases.

SSADM is heavily oriented towards the Requirements Specification and Design phases, with no orientation towards the phases after the Design phase. This is in keeping with one of the stated goals of the SSADM, as Goodman and Slater (1995, p. 442) state that it is specifically an analysis and design methodology aimed at "developing a full specification of a system", and thus does not cover the other stages of the SDLC.

Methodology	Proj. Init.	Req. Spec	Design	Code	Unit Test	Integ. Test	Sys. Test	Accept. Test	Sys. In Use
FDD	2	7	***21***	4	3	3	0	0	0
MSF	13	***17***	0	9	0	0	0	7	7
OOAD	8	16	17	***70***	0	0	2		5
RUP	17	13	***24***	11	4	0	7	0	2
SSADM	3	***42***	36	0	0	0	0	0	0
XP	0	11	8	***17***	4	4	0	7	0

Table 14: Life Cycle Phase Orientation: Activities

The results for the orientation of Process Roles are shown in the following table (*Table 15*):

Methodology	Proj. Init.	Req. Spec	Design	Code	Unit Test	Integ. Test	Sys. Test	Accept. Test	Sys. In Use
FDD	1	3	***7***	2	1	2	0	0	0
MSF	***4***	3	0	3	0	0	0	1	2
OOAD	2	***5***	3	3	0	0	1	0	2
RUP	2	4	***5***	4	1	0	4	0	0
SSADM	2	***8***	***8***	0	0	0	0	0	0
XP	0	2	***3***	2	1	1	0	***3***	0

Table 15: Life Cycle Phase Orientation: Process Roles

The Process Role orientation for FDD is consistent with its orientation of Activities, as described above. With the RUP, the orientation is consistent in terms of the Design phase, but not in terms of the Project Initiation phase, as there are the same number of Process Roles (four) orientated towards the Requirements Specification, Code and System Test phases. XP's Process Role orientation differs substantially with its Activity orientation, with more Process Roles involved in the Design and Acceptance Test phases.

OOAD orients its highest Process Roles usage towards the Requirements Specification phase, followed by the Design and Code phases, which differs from its Activity phase orientation. The orientation of Process Roles in MSF and SSADM are in keeping with the orientation of Activities, with

the slight exception being that the MSF specifies the use of the same number of Process Roles for the Code phase as for the Requirements Specification phase. (Please note that, for the purposes of this metric, Process Roles can span more than one Phase).

A more illuminating view results from determining the split of Activities and Process Roles per phase (and thus allocating an equal weighting to Activities and Process Roles, rather than a cumulative, score-based rating) and showing this in percentages. The results of this are shown in *Figure 17*. In the above case, the results obtained from analysing Activity orientation are confirmed. However, Extreme Programming shows very little (or no) difference in terms of orientation after the Code phase, with Requirements Specification, Design and Acceptance Testing achieving very similar scores. The results for the 3 highest scoring phases for Feature-Driven Development bear out the results from the Activity orientation metric. Feature-Driven Development is very highly oriented towards the Design phase (49.7%) with Requirements Specification being the next highest scoring phase. The Design phase (29.4%) is the highest scoring for the RUP, followed by the Requirements Specification (18.1%) and Code (16.6%) phases. These results do not bear out the results from the Activity orientation, but are closer to the Process Role orientation.

The following points should be noted:

- The MSF specifies that all kinds of testing should be performed, but provides no detail (Microsoft, 2003, p. 398), thus its testing activities were included in the Acceptance Testing phase.
- The MSF also includes what could be considered design tasks, but these are specifically included in the Planning phase (Microsoft, 2003, p. 20).
- The MSF deployment phase was included in the System in Use phase, as this was the most applicable place for it.
- OOAD has a very low orientation towards testing and XP a very high orientation.
- SSADM is specifically an analysis and design methodology, and thus does not cover the other stages of the SDLC. Goodland and Slater (1995, p. 6) state that SSADM corresponds to a traditional Systems Development life cycle (SDLC) as shown in *Figure 18*.

Conclusions

It would appear, from the analysis of the methodologies, that this measure provides a satisfactory view of methodology life-cycle coverage, specifically from a formal perspective as the intrinsic Completeness criteria provides an equally satisfactory view from a different perpective, and methodology granularity. While this measure of granularity, especially when considered on an Activity/Process Role per Life Cycle Phase, allows a detailed and accurate view into how a methodology covers the SDLC, there is a potential aspect of bias in the use of the SDLC, as this could favour methodologies based on the Structured paradigm. This bias did not, however, appear to manifest in this research.

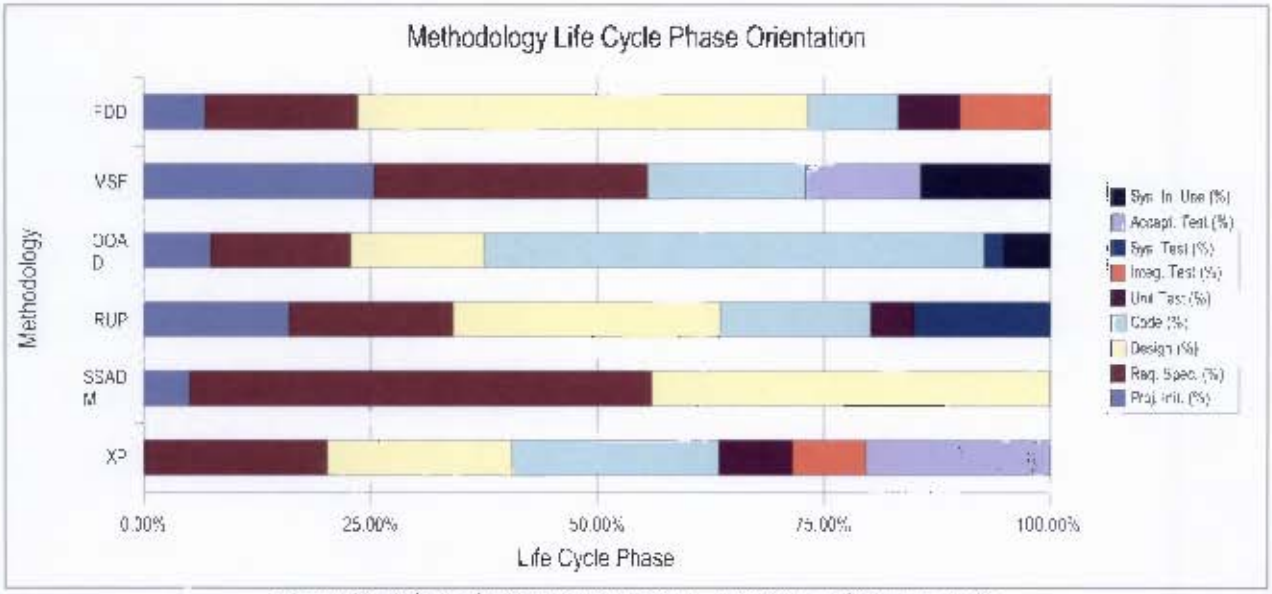

Figure 17: Life Cycle Phase Orientation: Activities and Process Roles

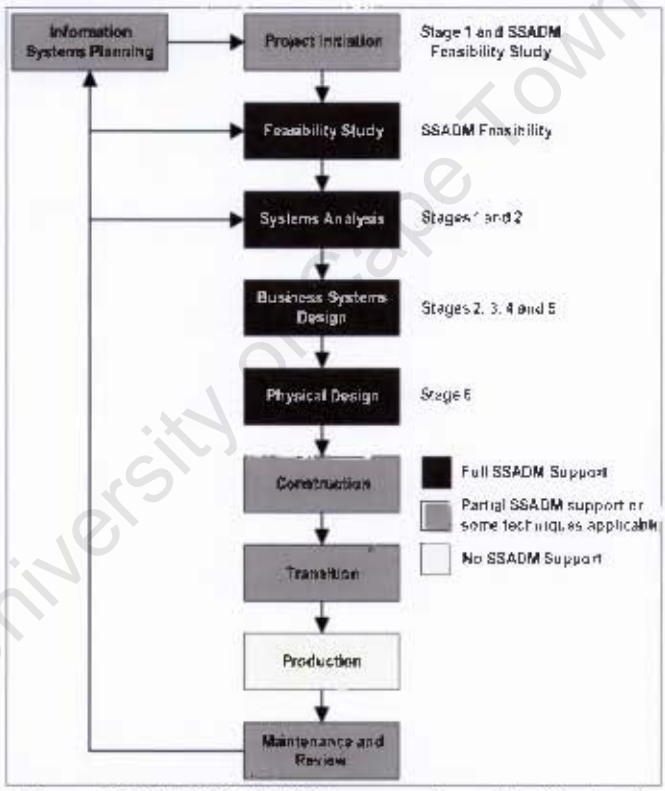

Figure 18: SSADM - SDLC Correspondence (Goodland and Slater, 1995, p. 6)

Formal Methodology Process Communication

Effective communication is essential to project success, as any misunderstandings or omissions in the communication process can result in costly, compounded defects later in the project (Atwood et al., 1995, p. 65; Bruegge and Dutoit, 1997, p. 272). Accordingly, Bruegge and Dutoit (1997, p. 280) argue that levels of mandated communication between team members in the methodology process "had a statistically significant relationship with [project] outcome", where higher levels of

communication resulted in more successful project outcomes. This view is shared by Heil, who states that effective teamwork, which she describes, in this context, as a combination of professional communication and a software development methodology process, results in "increased productivity and improved quality of output" (1999, p. 110). High-levels of intra-team communication, including formal communication, have also been shown to correlate with high-levels of job satisfaction amongst software development professionals (Javed, Maqsood and Durrani, 2004). Communication, specifically formalised communication, is also very important in distributed software development environments, and this communication must be driven by the process being followed (Gopal et al., 2002; Shami *et al.*, 2004; Mikulovic and Heiss, 2006; Ramesh *et al.*, 2006).

Intra-project communication has been measured using the volume and complexity of information exchanged amongst participants in the projects, specifically relating to intra-team communication, based on aspects such as changes in requirements, changes in development environment and change in organisation (Bruegge and Dutoit, 1997, p. 273). This criterion thus attempts to measure the formal, or mandated, inter-project communication between methodology users, or Process Roles as defined by the methodology reference material. This is based on the methodology process models as determined during the metamodeling exercise. In this case, the activity diagrams used to model the methodology processes can be used to show the flow of information between Process Roles (this is described in more detail in the metamodeling section) and the flows of information will be used as the basis for this criteria, essentially in order to determine the levels of inter-Process Role communication mandated by the methodology as part of the methodology process.

The following metrics will be used to measure this criterion:

- The number of Process Roles, broken down into:
    - The number of Individual Process Roles, or individual methodology users, for example: Project Manager, from FDD (Palmer and Felsing, 2002, p. 61).
    - The number of Team Process Roles, or methodology users in functional groups, for example: Modeling Team, from FDD (Palmer and Felsing, 2002, p. 61).
    - The number of client Process Roles, or representatives of the client given a role on the methodology's project team.
- The number of Object Flows between Process Roles.
- The ratio of inter-Process Role Object Flows to Process Roles.
- The ratio of inter-Process Role Object Flows to Individual and Team Process Roles.

The results are described in the following table (*Table 16*):

Criterion	FDD	MSF	OOAD	RUP	SSADM	XP
Number of Individual Process Roles	6	2	5	***20***	0	7
Number of Team Process Roles	4	***10***	8	0	6	2
Total Process Roles	10	12	13	***20***	6	9
Client Process Roles	***1***	0	0	0	0	***1***
Number of relationships flowing between Process Roles	17	26	17	***42***	30	23
Ratio of inter-role communication to Individual Process Roles	2.83:1	***13:1***	3.4:1	2.1:1	-	3.29:1
Ratio of inter-role communication to Individual and Team Process Roles	1.7:1	2.17:1	1.31:1	2.1:1	***5:1***	2.56:1

Table 16: Process Roles and Inter-Process Role Communication

From the results above, it is clear that the RUP has the highest number of Individual Process Roles (and the highest number of Process Roles in total), while the MSF has the highest number of Team Process Roles. SSADM prescribes no Individual Process Roles, and thus has the lowest number of Individual Process Roles and Team Process Roles in total. This can be explained by the fact that SSADM only covers the Analysis and Design aspects of the SDLC and thus does not cover aspects such as development and testing where different roles are required.

The RUP also has the highest number of Object Flows between Process Roles, as a consequence of the large number of Process Roles prescribed by the RUP. However, SSADM has the highest ratio of inter-role communication to Individual and Team Process Roles, which could be explained by SSADM's low number of prescribed Process Roles, which thus forces a high level of communication between the roles. This conclusion is confirmed by the methodology with the second-highest ratio of inter-role communication to Process Roles, XP, which has the second lowest number of Individual and Team Process Roles. This is supported by Beck's assertion that communication is the first value of XP (1999, p. 30). A conclusion that could be drawn from this is that these methodologies could be considered better at facilitating inter-Process Role communication, but their ratios of inter-Process Role communication are not significantly higher, with the exception of SSADM, than the other methodologies.

Another aspect of this communication, and one viewed as important by McLeod and Roeleveld (2002, p. 3) is that of communication between members of the project team and the client. Both XP and FDD include client representatives on their project teams and thus assign them Process Roles as Onsite Client and Domain Experts respectively (Beck, 1999, p. 52; Palmer and Felsing, 2002, p. 30). The addition of these Process Roles could be said to enhance communication between the project team and the client.

Conclusion

This proposed measures for this criteria appear to provide an adequate view of formalised communication within a methodology and it has been shown that this type of communication is a critical aspect in project success. As such, this criteria could be considered a useful one and valid in the context of this research.

In conclusion, this chapter has involved the following:

- A brief introduction to the criteria that make up the formal aspect of the proposed framework.
- A discussion of the metamodel and modeling approach used in this research.
- A discussion of the measures used for each criteria, and analysis and evaluation results for the methodologies analysed and evaluated for each of the criteria that make up the formal aspect of the proposed framework.

The following chapter addresses the second aspect of the proposed framework, the intrinsic view.

Chapter 6: Intrinsic Analysis

This chapter discusses the second view of the proposed framework, the intrinsic view. Chapter Five discussed the formal view and chapter Seven will discuss the remaining view, the pragmatic. The intrinsic view is concerned with the inherent and essential aspects of the methodologies being studied, specifically in terms of how the methodologies are described in their reference material. This chapter involves a short overview of the intrinsic criteria, followed by the results of the intrinsic analysis.

Intrinsic Criteria

The criteria used in the intrinsic view are:

- *Methodology Paradigm, Process and Notation,* the constituent aspects of the methodology.
- *Methodology Completeness*, the degree to which the methodology is considered complete.
- *Methodology Best Practice Coverage*, the degree to which the methodologies analysed cover identified best practices.
- *Methodology Support for Requirements Change*, the ability of the methodologies analysed to support changing requirements throughout the methodology life cycle.
- *Methodology Verification and Validation Techniques*, the techniques provided by the methodology to verify and validate the quality of the deliverables of the methodology.
- *Methodology Support for Project Management Activities*, the level at which the methodology supports traditional project management activities.
- *Methodology Documentation*, the quality of the methodology reference material documentation.

More detailed information on the criteria and the results of the analysis are provided below.

Paradigm, Process and Notation

Aspects identified as fundamentally important to the composition of methodologies are:

- *Paradigm*, or a collection of related components that form the foundation of methodologies and define a set of concepts forming ideas or notations associated with subjects (Alhir, 1998, p. 35).
- *Process*, or the structure that guides the order of project activities and represents the life cycle of a project (Microsoft, 2003, p. 2).
- *Notation*, or the language for expressing system models and concepts such as system behaviour or details of system architectures (Booch, 1994, pp. 23, 171).

Hackathorn and Karimi (1988, p. 202) state that, as a result of a widespread bias for the use of formalised methodologies in the development of software, the "search for the holy paradigm" has arisen from continual efforts to identify the "one best way" to develop systems (Fitzgerald, 1996, p. 10). This has resulted in an environment where, according to Fitzgerald (1996, p. 10), large numbers of competing methodologies, based on a small set of paradigms, have been developed, a trend labelled "methodolotary" by Beynon-Davies (1989, in Fitzgerald, 1996, p. 10). Tagg (1983, in Fitzgerald, 1996, p. 10) goes so far as to draw a parallel between this trend and the proliferation of religious sects stating that:

> *Despite the fact that they are 95% agreed in their aims and their broad areas of getting there, they nevertheless manage to stay separate. Each sect jealously guards its own style and magic ingredients.*

According to Fitzgerald (1996, p. 10), this literature bias is "especially problematic" as it creates a "circular pressure ... in that even though the literature may not *reflect* actual practice, it certainly influences it, thus creating a significant additional pressure in support of the use of formalised methodologies". While large numbers of methodologies exist, many exist as a result of insignificant and artificially contrived differences between one another "often based on product differentiation, personal ego and territorial imperative", and most share the same paradigms (Constantine, 1989, in Fitzgerald, 1996, p. 13).

However, there are substantial differences between some methodologies, according to Fitzgerald (1996, p. 13), in terms of philosophy, objective and techniques, and often fundamentally in terms of paradigm and focus. Some follow a 'hard' scientific, structured and rationalistic approach (such as SSADM), while others are 'soft' or human-oriented, and some follow a less-structured, agile paradigm. The main methodological paradigms relating to software development that have arisen over the last few decades are the structured paradigm, the object-oriented (OO) paradigm, and the agile paradigm. These paradigms have resulted in a multitude of methodologies, based on the principles and concepts that they define.

Another important component of a methodology relevant to its paradigm is its process model, which is described by Booch (1994, p. 23) as the activities leading to the orderly construction of the system's models. Microsoft (2003, pp.2, 3) state that the two main methodology process models in use are the waterfall and spiral, or iterative, model. Microsoft (2003, p. 3) defines the waterfall process as a process that uses milestones as transition and assessment points where, to begin a phase, all tasks comprising the previous phase must be complete. This process is considered to be most appropriate for projects where requirements can be clearly defined early in the project and are not liable to change in the future (Microsoft, 2003, p. 4). The spiral or iterative model is, according to Microsoft (2003, p. 4), based on the continual need to refine the requirements and estimates for a project, and most effective when used for rapid application development or for small projects, as it does not incorporate clear checkpoints.

While the detailed methodology process models have been analysed and discussed in detail in the previous chapter, this section aims only to delineate waterfall and spiral process models in terms of the methodology analyses, given their relevance to the methodological paradigm.

In terms of the final methodological aspect discussed in this section, IBM (2004) states that notations, such as UML, allow the level of abstraction to be raised in system description, while maintaining rigorous and standard syntax and semantics. In this way, notation standardises and improves communication amongst the project team (and other stakeholders) as the system design is formed and reviewed, allowing the reader to reason about the design, as well as providing an unambiguous basis for implementation (IBM, 2004). Booch (1994, pp. 171, 172) expands upon this sentiment, arguing that having a well-defined and expressive notation is vital to the process of software development as it removes ambiguity. This allows readers to concentrate on the problems inherent in the content (as opposed to the notation of the content) and "eliminates the tedium of checking the consistency and correctness" of designs by using automated tools (Booch, 1994, pp. 171, 172). Accordingly, this criterion will focus on describing the aspects of the methodologies described above based on information from the reference material. These aspects are:

- Paradigm.
- Process, including process phases.
- Notation, which, while mainly relating to the formal aspects of the methodology, is included in this section on a descriptive basis and because it relates to the semantic richness of the methodology.

As there is, according to Fitzgerald (1996, p. 10), no "holy paradigm" or "one best way", it could be considered misguided to use a relative perspective in the analysis of the methodologies for this criterion; an absolute, non-comparative perspective will be used instead. The results are presented in the table below (*Table 17*):

Methodology	Paradigm	Process	Number of Phases	Notation
FDD	Agile	Iterative	5	UML
MSF	Structured Agile	Iterative waterfall	5	UML and ORM
OOAD	OO	Iterative	9	OO Notation
RUP	OO	Iterative	4	UML
SSADM	Structured	Waterfall	6	Logical data models, data flow models, entity life histories
XP	Agile	Iterative	5	None

Table 17: Methodology Paradigm, Process and Notation information

In terms of paradigm, XP and FDD are both based on an agile paradigm, with XP being one of the first of the agile methodologies (Abrahamsson *et al.*, 2003, p. 246). FDD (Palmer and Felsing, 2002, p. 73) is described as an agile, highly-iterative methodology, while Beck (1999, pp. 6, 35, 46, 47) points to iterations as one of XP's most vital aspects, underpinning practices such as The Planning Game, Continuous Integration and Small Releases. In terms of process phases, XP consists of five phases, namely Exploration, Planning, Iterations to Release, Productionising and Maintenance (Beck,

1999). FDD consists of five phases, which were described in Chapter 2, and the Design by Feature and Build by Feature processes which are undertaken iteratively, according to Palmer and Felsing (2002, p. 57). SSADM is the only methodology analysed in this research that can be strictly described as being based on the structured paradigm and using a waterfall process (Goodland and Slater, 1995, pp. 4, 6).

SSADM consists of six phases specifically oriented around analysis and design activities (Goodland and Slater, 1995, p. 9), such as investigating the current organisational environment, exploring business system options, requirements definition, exploring technical system options and logical and physical design. An optional Feasibility phase is mentioned, but not described in any detail in the reference material and so has been excluded from this research. The MSF, on the other hand, is based on a mix of the structured and agile paradigms (Microsoft, 2003, pp. 3, 4), using a process that combines aspects of the waterfall and iterative processes, such as milestone-based planning and rapid feedback. OOAD and the RUP are both based on the object-oriented paradigm, with the RUP being described as a business process for OO software engineering (IBM, 2004: *Process Structure*). The RUP is, however, more closely aligned towards the agile paradigm than the other methodologies not based on the agile paradigm, with IBM stating that "RUP and the processes of the Agile community have a similar view of the key best practices required to develop quality software, for example, applying iterative development and focusing on the end users" (2004: *Agile Processes and RUP*).

Both the RUP and OOAD are based on an iterative process (Booch, 1994, pp.235, 249, 259; IBM, 2004: *Best Practice: Develop Iteratively*), with Booch's phases and process described in the *Figure 3* (Chapter 2). OOAD process phases (Booch, 1994, pp.235, 249) are based around two processes, the Macro and Micro processes. The Macro phases are concerned with the project life cycle as a whole, and the Micro phases are specifically concerned with low-level construction aspects of the system being developed. The RUP (IBM, 2004), on the other hand, specifies only four phases namely Inception, Elaboration, Construction and Transition.

One of the most interesting results of this analysis is the reliance of the methodologies analysed on the UML notation. FDD (Palmer and Felsing, 2002, p. 22) and the RUP (IBM, 2004:*Best Practice: Model Visually*) use UML exclusively, and the MSF uses UML in conjunction with the Object Role Modeling (ORM) notation (Microsoft, 2003, p. 57). OOAD uses Booch's OO notation, which was a forerunner to the UML (Booch, 1994, p. 171; Alhir, 1998, p. 10). In fact, the only methodologies that do not specify the use of an object-oriented-based notation are XP, which does not specify the use of any notation and SSADM, which uses a notation consisting of three types of views (and many more types of diagrams), namely:

- Logical Data Models.
- Data Flow Models.
- Entity Life Histories

Jeffries *et al.* (2000, p. 87) do, however, refer to the use of UML in XP reference material, stating that it may be beneficial to "do a CRC design with a few cards, or sketch some UML on the whiteboard or a sheet of paper" but no notation is mandated. In fact, documentation is almost frowned upon as Jeffries *et al.* (2000, p. 153) argue that documents or artefacts that are not used should not be produced. UML is the prescribed notation for FDD, the RUP and the MSF (in conjunction with the Object Role Modeling notation), is "flourishing" and has become the *de facto* modeling language for many methodologies (Alhir, 1998, p. 13). Interestingly, Booch's OO notation was one of the forerunners of the UML (Alhir, 1998, p. 5).

Conclusion

While this criteria should probably be considered to be of value, it can only really be of value from a descriptive perspective, as no attempt has been made to rank the methodologies analysed. There do not appear to be any valid alternative measures that can be used to rank methodologies based on their paradigm or processes, bar a purely subjective one. There are, however, criteria to measure notations, based on a formal perspective, but this type of analysis was beyond the scope of this research. For an example of this type of approach, see McLeod and Roeleveld (2002).

Completeness

In order to measure methodology completeness, two measures will be used:

- Degree of coverage of the Software Development Life Cycle (SDLC), where methodologies will be measured against the traditional SDLC. This criterion is intended to measure the completeness of a methodology's process.
- Degree of coverage of the Zachman Framework for Information Systems Architecture (Zachman, 1987), where methodologies will be mapped to, and measured against, this framework. This criterion is intended to measure the completeness of a methodology's artefacts.

When both of these evaluations have been completed, each methodology will be assigned a composite measure of completeness, based on their assessed completeness in the measures described above. The motivations for the use of these measures are discussed below. Other measures used to evaluate methodological completeness include that suggested by Mannino (1987) who measures methodology completeness based on the deliverables of a methodology.

For example, Mannino (1987, p. 5) states:

> *In terms of completeness, JSD is best because it delivers a completed system. SASD and the Box Structure Methodology fall behind JSD because they fail to deliver a completed system. Application Prototyping falls short in this category, because of the lack of in-depth development tools.*

Methodology Process Completeness

Abrahamsson *et al.* (2003, p. 247) state that "[a] software development life cycle is a sequence of processes that an organization employs to conceive, design, and commercialize a software product" and that a "life cycle perspective is needed to observe which phases of the software development process the agile methods cover". The Object Agency (TOA) (1995, p. 40) state that determining a methodology's coverage of the traditional SDLC is a useful tool for assessing methodological completeness, arguing that complete life cycle coverage is preferable to more limited life cycle coverage. This view is shared by Kelly (1987, p. 242), who states that the higher the level of life cycle coverage completeness, the easier training of personnel and communication between personnel, due to the reuse of central concepts and the use of "conceptual threads" throughout the methodology process. Accordingly, it could be argued that limited life cycle coverage results in incomplete, limited methodologies (McLeod, 1992).

As the SDLC phases used by Abrahamsson *et al.* (2003, p. 247) have already been provided, they will not be provided again here. However, using the base results determined in the analysis of SDLC Coverage, the life cycle orientation of the methodologies analysed are shown in *Figure 19*. Accordingly, if this criterion is to be used as a measure of completeness, XP, FDD, the RUP and OOAD can be said to be equally complete, as they cover the same number (six) of the life cycle phases described by Abrahamsson *et al.* (2003), albeit different actual phases. The only methodologies that cover fewer of these phases are the MSF and SSADM, which cover five and three phases respectively. As stated before, in the evaluation of formal criteria, it is interesting to note that the MSF contains activities that could make up a Design phase, but that these are specifically included in the MSF's Coding phase, and hence included under the Code phase for the purposes of this evaluation.

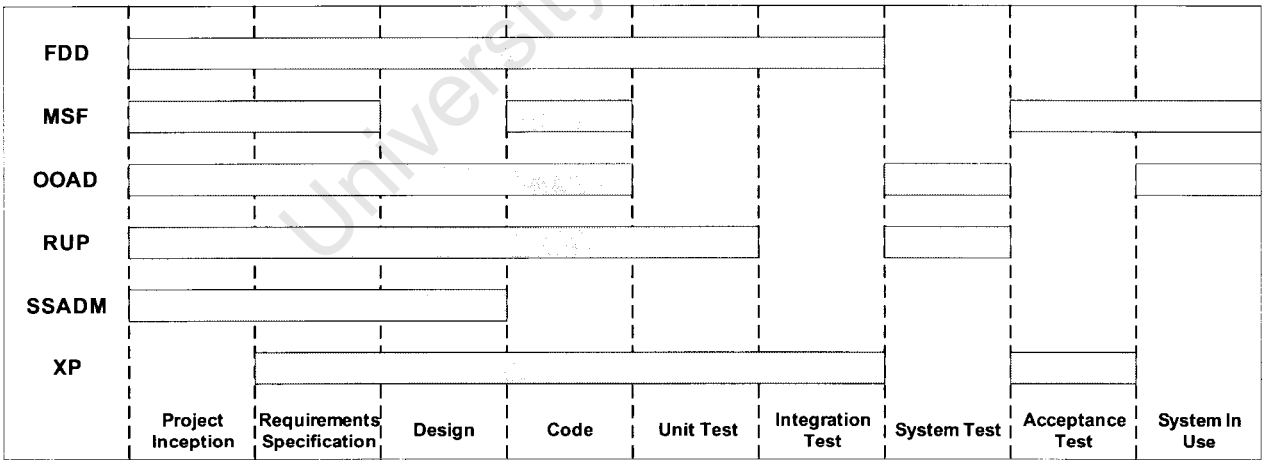

Figure 19: Methodology Life Cycle Coverage

XP, FDD, the RUP and OOAD cover the Requirements Specification, Design and Coding phases (as defined by Abrahamsson *et al.*, 2003) but FDD, the RUP and OOAD also cover the Project Inception phase. The MSF and SSADM also cover the Project Inception phase, and the only phase covered by all the methodologies is the Requirements Specification phase.

The most interesting differences occur in the testing phases. In terms of the types of testing explicitly covered, the evaluated methodologies differ substantially, specifically in the following ways:

- XP covers Unit Testing, Integration Testing and Acceptance Testing.
- FDD covers Unit Testing and Integration Testing.
- the RUP covers Unit Testing and System Testing.
- the MSF covers Acceptance Testing only.
- OOAD covers System Testing only.
- SSADM does not cover testing at all.

It is also interesting to note that none of the methodologies other than OOAD the MSF cover the phase after testing, namely System Use, or Maintenance. From this evaluation, it can be seen that there is actually very little difference in methodology completeness in terms of life cycle phases covered, with the exception of the SSADM, which is intended to cover only the Requirements Specification and Design life cycle phases.

All of the other methodologies evaluated cover at least the Requirements Specification, Coding and one testing life cycle phase, and all, bar XP, cover the Project Inception life cycle phase.

Zachman Framework Coverage Completeness

A second measure of methodology completeness could be the methodology's coverage of the Zachman Framework for Information Systems Architecture (Zachman, 1987). This framework is intended to measure information systems architectures along two axes:

- *Human*, which encompasses six dimensions:
  - Scope, or ballpark.
  - Owners, or enterprise model.
  - Designers, or system model.
  - Builders, or technology model.
  - Out of context, or detailed.
  - Operational, or functioning system.
- *Foci*, which encompasses another six dimensions:
  - What, or data.
  - How, or function.
  - Where, or locations.
  - Who, or people.

- When, or time.
- Why, or motivation.

The dimensions combine into a matrix (shown below in *Table 18*), which offer a "powerful tool for analysing software engineering deliverables" (de Villiers, 2001, p. 3). de Villiers (2001, p. 3) contends that, using this framework, an organization can "assess the coverage of its software development process ... within the context of both business and information system strategies". Accordingly, the Zachman framework can be used to measure the completeness of methodology artefacts

In order to measure the methodologies against the Zachman framework, the approach followed by de Villiers (2001, pp. 7-9) was used, where methodology artefacts, in this case artefacts from the RUP, were mapped against the Zachman framework. In this case, artefacts are similar to deliverables, in that they are used, produced or modified during the methodology process, but may not be actual methodology deliverables (de Villiers, 2001, p. 5). Once the methodologies had been mapped against the Zachman framework, an analysis was performed regarding their coverage of the framework. The results are shown in the following tables (*Tables 19 and 20*), and followed by a discussion.

A few points to note are that de Villiers' analysis of the RUP against the Zachman framework (2001, pp. 7-9) was taken 'as is' for the purposes of this research and Zachman's Operational View was not used, as this view is process independent (de Villiers, 2001, p. 8) and, as the functioning system is the product under development, its elements will be present no matter what process model is used. Full methodology mappings to the Zachman framework are shown in *Appendix A*.

	What (Data)	How (Function)	Where (Locations)	Who (People)	When (Time)	Why (Motivation)
Scope {contextual} Planner	List of things important to the business	List of processes that the business performes	List of locations in which the business operates	List of organisations important to the business	List of events/cycles important to the business	List of business goals/strategies
Enterprise Model {conceptual} Business Owner	e.g. Semantic Model	e.g. Business Process Model	e.g. Business Logistics System	e.g. Workflow Model	e.g. Master Schedule	e.g. Business Plan
System Model {logical} Implementer	e.g. Logical Data Model	e.g. Application Architecture	e.g. Distributed System Architecture	e.g. Human Interface Architecture	e.g. Process Structure	e.g. Business Rule Model
Technology Model {physical} Implementer	e.g. Physical Data Model	e.g. System Design	e.g. Technology Architecture	e.g. Presentation Architecture	e.g. Control Structure	e.g. Rule Design
Detailed Representation {out-of-context} Subcontractor	e.g. Data Definition	e.g. Program	e.g. Network Architecture	e.g. Security Architecture	e.g. Timing Definition	e.g. Rule Definition
Functioning System	e.g. Data	e.g. Function	e.g. Network	e.g. Organisation	e.g. Schedule	e.g. Strategy

Table 18: The Zachman Framework for Information Systems Architecture (Zachman, 1987)

Methodology	What	How	Where	Who	When	Why	Total
FDD	4	3	0	4	2	0	13
MSF	***5***	***5***	***4***	4	***5***	***4***	***27***
OOAD	4	4	1	0	2	0	11
RUP	***5***	***5***	3	***5***	***5***	3	26
SSADM	4	***5***	0	***5***	2	1	17
XP	0	2	0	0	0	2	4

Table 19: Zachman Framework: Six Foci Coverage

Methodology	Scope View	Owners View	Designers View	Builders View	Detailed View	Total
FDD	3	3	4	2	1	13
MSF	***6***	***6***	***6***	***6***	3	***27***
OOAD	0	2	3	4	2	11
RUP	5	***6***	***6***	5	***4***	26
SSADM	2	3	4	4	***4***	17
XP	1	0	2	1	0	4

Table 20: Zachman Framework: Views Coverage

From this analysis, it is possible to observe a number of points:

- The MSF is, followed by the RUP, the most complete methodology in terms of its adherence to the Zachman framework.
- XP does not map well to the Zachman framework, and covers only four cells (see *Table 40* in *Appendix A*). This may be due to the fact that XP is a practice-based methodology, as opposed to an activity based methodology.
- While FDD, OOAD and SSADM can be mapped to the Zachman framework, they appear to map to specific areas, mainly the Who, What (with the exception of OOAD) and How dimensions in the Foci, and do not cover dimensions such as When, Where and Why very well, or at all.
- Almost no methodologies, with the exception of the MSF and the RUP, cover the Where focus, and the When focus is similarly badly-represented. The Why focus is also not very well represented.
- The best represented views appear to be Designers and Builders views, followed by the Owners view, while the Detailed view is the least represented view, followed by the Scope view.

Composite Completeness Value

As discussed previously, a composite measure of methodology completeness will be suggested, composed of the process and artefact completeness values. This value will be composed as follows:

CCV = AVERAGE(SDLCC + ZFC)

where:

- ***SDLCC*** represents process completeness, given as a normalised, percentage-based value to represent the number of SDLC phases covered. For example, FDD, in that it covers 6 out of 9 SDLC phases, would be assigned an SDLCC value of 67%.
- ***ZFC*** represents Zachman framework coverage, or artefact completeness, given as a normalised, percentage-based value. For example, the MSF, which covers 27 of the 30 cells used in the evaluation, would be assigned a ZFC value of 90%.

The SDLCC and ZFC values are given as percentage values in order to give them equal weighting. Accordingly, the CCV will be also be shown as a percentage value. Finally, a ranking will be assigned to the methodologies based on their completeness, with the value 1 being assigned to the most complete, and 6 to the least complete.

The results are shown in the table below (*Table 21)* :

Metric	FDD	MSF	OOAD	RUP	SSADM	XP
SDLCC	***67%***	56%	***67%***	***67%***	33%	***67%***
ZFC	43%	***90%***	37%	87%	57%	13%
CCV	55%	73%	52%	***77%***	45%	40%
Ranking	3	2	4	***1***	5	6

Table 21: Composite Completeness Results

From this analysis, it can be seen that the RUP is the most complete of the methodologies analysed, followed by the MSF. XP is, by the same criteria, the least complete, scoring only 9%. This is obviously due to its lack of artefacts and coverage of the Zachman framework, as it covers six SDLC phases, which is as high as any of the other methodologies.

Conclusion

Given that both of these measures have been suggested and used in previous literature, it would appear that they are valid measure of methodology completeness. The combination of the two measures offers a relatively complete view of methodology completeness, as it extends to both process completeness and artefact coverage. However, it could be argued that this measure does neglect the agile methodologies which value "working software over comprehensive documentation" (Boehm, 2006, p. 19). As such, while this criteria is of value to the framework and the analysis and evaluation of methodologies, this limitation should be noted.

Best Practice Coverage

Cortada (1998, p. 2) defines best practices as "processes which are recognised as being the best by function or within an industry", while ManagementUpdate.info (2007) defines a best practice as a "process of developing and following a standard way of doing things that is more effective at delivering a particular outcome than any other ways and which multiple organisations can adopt and utilise". In terms of software engineering or development, Withers (2000, pp. 432, 433) states that the United States Department of Defence (DOD) has a long-standing interest in the success of software engineering projects and, as a result, has undertaken numerous initiatives to measure and improve the success of software engineering projects. McGrath (1998, in Withers, 2000) defines a best practice as a management or technical practice, in the context of software engineering, which has been shown to consistently improve one or more of:

- Productivity.
- Cost.
- Schedule.
- Quality.
- User satisfaction.
- Predictability of cost and schedule.

At present, there is a large amount of industry literature available concerning best practices in specific areas – for example, the ZDNet white paper repository offers over two thousand white papers concerning best practices and Microsoft's Developer Network (msdn.microsoft.com) offers a huge archive of best practices for application developers using Microsoft products. This criterion is concerned with the analysis of stated methodological practices, which are necessarily, at least in the eyes of their creators (for example, Perks, 2003), best practices. The intention of this criterion is to measure the methodologies analysed against other sets of best practices, such as those suggested by the United State Department of Defence through the Airlie Software Council (Withers, 2000), to determine how closely the methodologies analysed correspond to these sets of best practices.

In order to gain an accurate view of the best practice coverage of the methodologies analysed, a variety of other data sources were used, such as Ambler's Enterprise Unified Processes (2005), methodology-independent sets of best practices such as those proposed by Perks (2003), the set suggested by the United State Department of Defence through the Airlie Software Council (Withers, 2000), as well as the sets of best practices suggested as part of the methodologies. The sets of best practices all contained between six and sixteen individual best practices and, bar those proposed by Withers (2000), Beck (1999) and Jeffries, Anderson and Hendrickson (2000), were not organised into categories. As the MSF, OOAD and SSADM do not specify best practices, they were not used in this analysis.

Approach Followed

Hong, Van Den Goor and Brinkkemper (1993, p. 693) suggested a superset-creation approach involving the development of categories for data to be explored (in this case process, concepts and techniques) from the methodologies studied. From this, a supermethodology was created, consisting of the smallest common denominator of all methodological activities. Once the supermethodology was created, the individual methodologies were compared against this supermethodology in terms of coverage.

The superset of best practices is defined as a relative common denominator of proposed best practices, which is slightly different to that suggested by Hong *et al.* (1993, p. 693). At this point, it is necessary to offer a definition for the data to be used. A set of best practices can be defined as a number of best practices proposed by the same author in the same publication, related in some way to a central concept. For example, Palmer and Felsing (2002, p. 36) propose a set of 8 best practices related to the central concept of Feature-Driven Development. In the case of this research, the data gathered will be collated and divided into categories, according to the activities upon which the best practices are based. Once the data has been collated, a superset of best practices for software development will be created, and the individual sets of best practices compared against the superset in terms of determining the completeness of the individual sets of best practices.

In essence, if a best practice appears in two or more of the sets of proposed best practices used as data, it will be included as a best practice in the superset of best practices. The best practices defined in the superset will be defined in less detail than those suggested in the sets of best practices used as

data. This is necessary as the sets of best practices used as data are often specifically based on paradigms and, as a result, two best practices, while slightly different in detail due to the paradigm used by their authors, can be very similar when viewed aparadigmatically. The categories to be used in the evaluation will relate to activities that are prevalent and considered valuable in software development, some of which are defined by Jacobson (1992, p. 472) as pragmatics. These categories include activities such as project management and testing as well as activities or techniques such as system architecture and configuration management. The aim of the superset will be to create a common set of best practices from the data, against which the individual sets of best practices can be compared in terms of their completeness.

Best Practice Sets

The sets of best practices selected for this analysis are those suggested by:

- The Rational Unified Process (RUP) (IBM, 2004) [Set 1].
- The Enterprise Unified Process (EUP), an extension of the RUP (Ambler, 2005) [Set 2].
- Feature-Driven Development (FDD) (Palmer and Felsing, 2002) [Set 3].
- Perks (2003) [Set 4].
- The Airlie Software Council (Withers, 2000) [Set 5]
- Extreme Programming (XP) (Beck, 1999, Jeffries *et al.*, 2000) [Set 6].

These sets of best practices are described in more detail in *Appendix B*.

Superset Creation

Although only two of the above authors categorised their individual best practices, three categories could be defined, based on the activities suggested in the sets of best practices used as data. These categories are:

- Product quality activities, which are based on ensuring the quality of the end product of the software development project, such as continuous testing.
- Development process activities, which are based on the best process activities which contribute to the development of the end product of the software development process, such as configuration management.
- Project management activities, which are based on managing the software development process and aspects possibly extraneous to this, such as tracking project progress.

The second activity after the capturing of the sets of best practices used as data was the creation of the superset of best practices, a relative smallest common denominator of best practices included in the sets of best practices used as data. It is important to note that, in the interests of objectivity, the practices suggested by Pierce (2005) were not included, as these best practices have been defined, almost to a word, by IBM (2004) and, given Pierce's possible vested interest in the RUP, the superset of best practices would have been skewed towards the RUP.

The superset of twelve best practices, and their source sets, are detailed as follows:

- Iterative development – Best Practice sets 1, 2.
- Manage requirements – Sets 1, 2, 4, 5;
- Continuously verify product quality through testing – Sets 1, 2, 4, 5, 6.
- Manage change – Sets 1, 2.
- Collaborative development – Sets 2, 3, 6.
- Consideration for system operations and support – Sets 2, 4.
- Continuous integration – Sets 2, 3, 6.
- Code inspections/reviews – Sets 3, 4, 5, 6.
- Progress tracking – Sets 3, 4, 5.
- Risk management – Sets 2, 5.
- Quality management through defect tracking – Sets 4, 5.
- Configuration management – Sets 3, 4, 5.

There were, however, some paradigmatic clashes in what is considered a best practice by different authors. For example, Palmer and Felsing (2002) propose individual class and program code ownership, while Beck (1999) and Jeffries *et al.* (2000) state that there should be collective ownership of program code. Secondly, ASC (Withers, 2000) defines one best practice as that of designing twice and coding once, which is at odds with Beck's practice of simple design (1999).

Another interesting aspect to note is the fact that, because they are based on the same underlying principles and, in essence, the Enterprise Unified Process is based on the Rational Unified Process (Ambler, 2005), a higher percentage of the best practices suggested in these sets was included in the superset of best practices, 75% of best practices for the RUP, and 80% for the EUP. However, only 2 best practices were not also suggested by other authors and, of these, one (iterative development) is, if not stated as a best practice, considered a cornerstone of two other data sources; Extreme Programming and Feature-Driven Development (Beck, 1999, p. 6; Palmer and Felsing, 2002, p. 39). Secondly, two best practices in both the RUP and the EUP, although similar, were not included in the superset of best practices, as the differences could be considered significant.

These best practices were proven/component-based architecture, which also differs slightly but significantly to Perks' (2000) choosing the correct architecture, and modeling (visually, in the case of the RUP). The superset of best practices is thus categorised in *Table 22* and the results of the analysis are shown in *Table 23*.

Category	Related Best Practices	Number of BP's
Product quality activities	Continuously verify product quality through testing, code inspections/reviews, quality management through defect tracking	3 (25%)
Development process activities	Iterative development, collaborative development, continuous integration, configuration management	4 (33%)
Project management activities	Manage requirements, manage change, consideration for system operations and support, progress tracking, risk management	5 (42%)

Table 22: Categorised Superset of Best Practices

Best Practices Completeness Results

The following criteria relating to the results of the analysis are shown in the following table (*Table 23*):

- *Number of BPs* (Best Practices) is a count of each datasets BPs.
- *BPs in Superset* describes the number of the dataset BPs that were included in the superset of BPs.
- *% BPs in Superset* describes the percentage of the dataset BPs that were included in the superset of BPs;
- *% of Superset BPs in Set* shows the percentage of dataset BPs that make up the superset; and
- *Best Practice Coverage* is thus the average of *% BPs in Superset* and *% of Superset BPs in Set*.

Set	Number of BPs	BPs in Superset	% BPs in Superset	% of Superset BPs in Set	Best Practice Coverage
ASC	16	8	50%	67%	59%
EUP	10	8	80%	67%	74%
FDD	8	4	50%	33%	42%
Perks	16	7	44%	58%	51%
RUP	6	4	67%	33%	50%
XP	12	4	33%	33%	33%
Superset	**12**	**12**	**100%**	**100%**	**100%**

Table 23: Best Practice Coverage

In terms of datasets included in this research for the benefit of this criteria, namely the EUP (Ambler, 2005), Perks (2003) and the ASC (Withers, 2000), the EUP has the greatest coverage of the superset of best practices. 80% of the best practices that the EUP suggests are included in the superset (FDD and the RUP are the only other data sets that have 50% or more of their best practices included in the superset). It is important to note that only the EUP, Perks and the ASC had a best practice coverage of over 50%; however, neither Perks' nor the ASC's coverage (51% and 59% respectively) was substantially higher than the RUP's.

Of the methodologies analysed throughout this research (FDD, the RUP and XP), the RUP had 50% coverage of the superset, FDD had 42% coverage and XP had 33% coverage, or the lowest. This could be explained by Beck's (1999, p. 54) assertion that XP's practices, while simplistic and long-used, have been "abandoned for more complicated, higher overhead practices, as their weaknesses have become apparent". Beck (1999, p. 54) believes that when used in tandem by XP, these practices support one another effectively, despite their intrinsic weaknesses. Stephens (2004) states that these practices form a "self-referential safety net" or a "ring of poisonous snakes, daisy-chained together", although this may depend on your viewpoint.

Conclusion

While this criteria could be a good relative measure of methodologies, possibly relating to methodology similarity or general methodology quality, the analysis above shows that it will be inadequate unless comparing methodologies that suggest best practices. For example, a comparison of FDD, the RUP and XP would provide valuable results, while a comparison of the MSF, OOAD and SSADM would not be possible, thus rendering this criteria invalid in that context. As such, this criteria should be considered an optional criteria, based on the methodologies analysed.

Support for Requirements Change

The accelerating pace of change, characteristic of the business environment currently facing organisations, has also impacted on software development and has resulted in the need for methodologies to accommodate short-term change in requirements, as well as traditional long-term, large-scale development approaches spanning a significantly longer time frame (Fitzgerald, 1996, p. 19). Methodological support for changes in requirements results in systems that are more extensible, modifiable and maintainable, and that the ease with which "changes necessitated by new requirements, error corrections, new environments, and enhancements, may be introduced into a product" speaks to the quality of the methodology used (TOA, 1995, p. 53). The belief that methodologies need to support changes in requirements extends to the idea that the use of a methodology that does not support changes in requirements can actually "hinder essential organisational needs" (Fitzgerald, 1996, p. 19). This is however contingent on the methodology's ability to provide this support in a structured, formal manner, as opposed to an *ad hoc* process, which can be as dangerous as offering no support for changes in requirements.

This support for requirements changes is often implemented through iterative or evolutionary processes or an accelerated development approach (Fitzgerald, 1996, p. 20). TOA (1995, p. 54) use four criteria to determine the level of support for requirements change in the methodology reference material, namely:

- Whether the concepts of supporting changing and additional requirements were discussed.
- Whether a process was described, although not necessarily formally defined, to deal with changing or new requirements.

- Whether heuristics were provided, with reference to accommodating changing and additional requirements, to users of the methodology.
- Whether examples discussing the accommodation of changing requirements were provided.

To determine a relative value for each methodology's support for changes in user or system requirements, a composite metric (*SRC Value*) will be used, calculated from the degree to which the individual criterion are met. This value will be calculated as follows:

> ***SRC = (Concepts Discussed) + (Process Described) + (Heuristics Provided) + (Examples Provided)***

Where Yes = 1, and No = 0.

In this case, none of the criteria were weighted and the results are shown in *Table 24*. In terms of the results, Extreme Programming is, at least according to the reference material, a very change-friendly methodology, as Jeffries *et al.* (2000, p. 152) states: "Don't try to freeze requirements before you start implementing ... XP lets you use a development and planning approach that allows for change". Beck (1999, p. 119) believes that XP's iterative nature and practices of the Planning Game and small releases gives the customer "natural points to change the direction of the project" while allowing the scope of the project to be continuously negotiated. While no specific requirements change process is defined, according to Beck (1999, p. 119), XP allows customers to define specifically which requirements they require on a per-iteration basis, thus integrating the change control process into the project planning process. As a result, XP could be considered to provide a process to manage requirements change. However, no real heuristics or examples are provided within the XP reference material to support the concept and process discussion.

Methodology	Concepts Discussed?	Process Described?	Heuristics Provided?	Examples Provided?	SRC Value
FDD	Yes	Yes	Yes	No	3
MSF	No	No	Yes	Yes	2
OOAD	No	No	No	No	0
RUP	Yes	Yes	Yes	No	3
SSADM	Yes	No	No	Yes	2
XP	Yes	Yes	No	No	2

Table 24: Support for Requirements Change Results

The Rational Unified Process offers the most formalised process around managing requirements change, in that they describe a formal change request process (IBM, 2004:*Concepts: Change Request Management*). While the MSF provides both requirements change heuristics and the example of a requirements change management section from a Project Structure Document (Microsoft, 2003, pp. 104-105, 109), it discusses neither the concepts of requirements change, nor does it provide a process to manage these changes. Based on this analysis, OOAD is deficient at accommodating changes in requirements, as almost no mention is made of it in the reference material.

In contrast, FDD (Palmer and Felsing, 2002, pp. 234, 235) is thorough in its coverage of this aspect of software development, discussing the concepts of managing requirements changes, providing a process to manage these changes, and heuristics for dealing with changes in requirements. FDD is, along with the RUP, the methodology best equipped to support changes in requirements. SSADM appears to view requirements change, and the management thereof, as a project management task (Goodland and Slater, 1995, p. 449), beyond the scope of the analysis and design phases that SSADM supports, and thus can be said to be almost unconcerned with it. Goodland and Slater (1995, p. 449) propose that a configuration management process should be implemented to manage changes, and that all changes should follow formal configuration management control procedures as Requests for Change or Off-Specification reports and an example of this is provided.

Based on the results of this analysis, it can be seen that there is a difference in the levels at which methodologies support changes in requirements. Some, specifically FDD and the RUP, appear to have taken into account the maxim that the only constant is change, and thus are better prepared than the others. OOAD, on the other hand, appears to treat requirements as entirely fixed and does not offer any support for changes in them. The middle ground, where the MSF, SSADM and XP reside, does appear to accept that changes in requirements are a necessary part of software development and make provision for them, but is often not completely.

Conclusion

The measures suggested for this criteria appear to provide an adequate view of methodology support for requirements change. It could be argued that assessing aspects such as whether a process is provided to support changing requirements verges on the formal aspects of methodology analysis in the context of this research, however, this aspect does take a descriptive approach, merely assessing whether a process exists in the methodology reference material. One area in which these measures could be criticised is detail, as they do not venture into substantial, comparative detail in terms of the quality of the methodology support for requirements change, preferring instead to focus on the presence of aspects of support for requirements change.

Verification and Validation Techniques

McLeod (1992) argues the following about the need for verification and validation techniques in methodologies, and methodology process and output quality:

> *Quality is an underlying philosophy and cannot be grafted on afterwards. It is implicit in every task performed, and every product produced. Consequently, within the description of the method, every task and deliverable must have associated quality standards. In addition, formal quality review points and an error detection/correction scheme are built into the management process. These apply to the method itself as well as the product being produced by the project.*

TOA (1995, p. 10) state that "available mechanisms and heuristics for verification and validation of the process are also desirable attributes of a well-defined methodology". TOA (1995, p. 44) define verification techniques as rules that allow the verifying of correctness for developed products, and

validation techniques as some form that allows independent validation of the development products with the customer, independent of the methodology's notation itself.

Verification and validation techniques can, according to Kelly (1987, p. 241), extend to determining how well the methodology "catches deviations from the requirements, inconsistencies, and incompleteness in the design before going to code", as well as checking interfaces and semantics to ensure that the product is being built according to the requirements. These techniques may consist of design reviews, used in conjunction with automated tools and manual techniques (Kelly, 1987, p. 241). In order to determine methodology support for verification and validation techniques, TOA (1995, p. 44) follow the approach of investigating the methodology reference material for discussion of these techniques, based on a number of criteria, the results of which are determined from the reference material. The criteria are as follows:

- Whether verification rules are provided.
- Whether a verification/validation process is described.
- Whether heuristics are provided.
- Whether examples are provided.

The methodologies will be assigned a per-question score as follows:

- *Yes*, if the methodology reference material meets the criteria.
- *Some*, if the methodology reference material meets the criteria in some aspects, but is obviously incomplete (this answer is only applicable if, for example, verification and validation heuristics or examples appear to be provided on a per-Phase basis and provided for some of the methodology Phases, but not for others).
- *No*, if the methodology does not meet the criteria.

This is the same approach followed by Kelly (1987, p. 241) and Matinlassi (2004, p. 2) to evaluate methodology verification and validation techniques. One aspect that was not performed in any of the research mentioned above was that of assigning a composite index based on the methodology results. This index, the VVT index, is based on assigning each question/answer pair a score based on the following criteria:

- *Yes* = 2.
- *Some* = 1.
- *No* = 0.

and is the sum of the values assigned to the questions per methodology. The results of this analysis are shown in the table (*Table 25*) below.

Methodology	Verification Rules Provided?	Process Described?	Heuristics Provided?	Examples Provided?	VVT Score
FDD	Yes	No	Yes	No	4
MSF	No	Yes	Yes	Yes	6
OOAD	Yes	No	Some	Some	4
RUP	Yes	Yes	Yes	No	6
SSADM	Yes	Yes	Yes	No	6
XP	Yes	No	No	No	2

Table 25: Verification and Validation Technique Analysis Results

The reference material for FDD discusses verification techniques in terms of each of the five main processes, but does not describe any actual verification processes, merely the activity to be performed to verify the quality of the result of the process (Palmer and Felsing, 2002, pp. 132-133, 143, 154, 175-177, 192). These techniques often lack detail, for example, the verification discussion for the Develop and Overall Object Model process states: "Domain Experts actively participating in the process provide internal or self-assessment. External assessment is made on an as-needed basis by referring back to the business (users) for ratification or clarification of issues that affect the model" (Palmer and Felsing, 2002, p. 132), but are supplemented by heuristics for the resources concerned.

The Rational Unified Process (IBM, 2004), on the other hand, provides a full description of the concepts of verification and product quality (*Best Practice: Continuously Verify Quality*), verification processes to be followed (for example: *Activity: Review The Design*, IBM, 2004) and evaluation and review heuristics, provided in Milestone Reviews for *Life Cycle Objectives Milestone*, *Lifestyle Architecture Milestone*, *Initial Operational Capability Milestone* and *Product Release Milestone*, to guide the verification process. However, no examples are provided in the RUP reference material.

Jeffries *et al.* (2000, pp. 103-105) discuss verification and validation concepts, and XP describes a number of practices which provide verification and validation (although no specific verification or validation *processes* are provided). These include pair programming, where two programmers allow for a level of full-time code and design review (Wake, 2000, p. 75); refactoring, where programmers validate and, if necessary, restructure and redesign the system to improve the quality of the program code (Beck, 1999, p. 47); and combining the practices of having an onsite customer and performing continuous integration. The latter allows the user in charge of accepting or rejecting the system to continuously verify the quality of the system being produced (Jeffries *et al.*, 2000, p. 35). Another aspect of Extreme Programming concerned with verification and validation is the emphasis placed on unit tests, with the simple heuristic of "You must have unit tests for everything that could possibly break, and they must always be at 100%, for every release of code by every programmer" (Jeffries *et al.*, 2000, p. 161).

Microsoft, on the other hand, does not discuss verification and validation concepts, but does describe validation processes for all main deliverables and processes, complete with heuristics (Microsoft, 2003, pp. 102-103, 145, 151, 161-162, 194, 203-204, 260, 273-274, 419). Examples are also

provided on an accompanying CD. SSADM (Goodland and Slater, 1995, p. 3) state that quality assurance reviews must be held to ensure that the end products for each stage are "scrutinized for quality, completeness, consistency, and applicability by users, developers, and by experienced systems staff external to the project", and believe that these processes ensure that systems are flexible, amenable to change and meet requirements (Goodland and Slater, 1995, p. 4).

All processes described by SSADM include descriptions of the products to be reviewed and activities to be performed to review these products, along with reviewer roles. For example, Goodland and Slater (1995, pp. 133 and 134) describe the review process and provide heuristics for the review of Stage 1 products, namely the *Requirements Catalogue*, *Data Catalogue*, *Current Environment and Logical Data Model*, *Context Diagram* and *Logical Data Store/Entity Cross Reference*. However, no examples are given.

Booch (1994) discusses a number of aspects of quality, such as design quality (On Building Quality Classes and Objects, p.136) and quality assurance and metrics (p.278) and states that "software quality doesn't just happen: it must be engineered into the system". Measures for each high-level process are described, but in relatively low detail and with no process description. For example, the verification measure for the Evolution high-level process is discussed as the "primary measure of goodness is therefore to what degree we satisfy the function points allocated to each intermediate release, and how well we met the schedules established during release planning". Some level of heuristics and examples are provided, but only regarding design (Booch, 1994, i.e. pp. 138, 139, 141).

Conclusion

The same criticism that could be levelled against the measures of support for requirements change could be levelled against this criteria, namely that it does not provide a comparative assessment of the quality of methodology verification and validation techniques, focusing instead on assessing the presence of these aspects. A possible way to overcome this would be to weight the criteria composing the VVT score, but this may require the apportioning of subjective or contextual values, which is not within the scope of this research.

Support for Project Management

This criterion is concerned with the degree to which the methodologies analysed provide support for project management activities. Griffiths (1978, p. 41) states that "what makes or mars a methodology in the market-place is how close it comes to satisfying the principal project management requirements". Abrahamsson *et al.* (2003, p. 247) argue that project management is a support function that provides the backbone for efficient software development. Methodology support for project management activities is very important as it relates directly to methodology efficiency, and that project management activities are required in order to enable the "proper execution of software development tasks" (Abrahamsson *et al.*, 2003, p. 247).

Project management tasks and activities are necessary within a methodology but that many methodologies do not provide sufficient support for these tasks and activities (TOA, 1995, p. 5). These activities are (Thayer, 2002, pp. 227-267):

- *Planning*, determining a course of action for accomplishing the organisational or project objectives.
- *Organising*, creating the relationships among work units in order to accomplish the project objectives, and granting the responsibility and authority to others to obtain those objectives.
- *Staffing*, selecting and training people for positions in the organization.
- *Directing*, creating a working atmosphere that allows and motivates people to achieve the desired project end results.
- *Controlling*, establishing, measuring, and evaluating the project performance toward its planned objectives.

Abrahamsson *et al.* (2003, p. 247) approach the measurement of project management support in the same way that they approach the measurement of methodological support for life cycle phases: by mapping the methodology activities to a Software Development Life Cycle (SDLC). As such, their results reflect the SDLC phases when project management activities occur, rather than the actual degree of project management support. TOA (1995, p. 41), on the other hand, use a "rigorous" comparison mechanism for methodological criteria based on the following approach:

- Specify one or more criteria-relevant questions for the evaluation to answer (where the answers will traditionally be binary, or Yes/No).
- Ask these questions of the methodology being evaluated.
- Assign a value to each question-answer pair, such as Yes = 1, No = 0.
- Record the results for comparison and calculate the score per methodology.

Accordingly, a set of criteria will be used to evaluate the methodology reference material with respect to support for project management activities, including questions of the methodologies as to whether:

- The concepts of Planning, Organising, Staffing, Directing and Controlling are discussed.
- A Planning Process or Activity is described.
- An Organising Process or Activity is described.
- A Staffing Process or Activity is described.
- A Directing Process or Activity is described.
- A Controlling Process or Activity is described.
- A project manager Process Role is described.

Finally, a value (SPM) will be assigned to each methodology, based on their support for project management. This value will be a composite, based on the their results for the previous seven criteria, with scores awarded where *Yes* responses will be assigned the value 1, and *No* responses will be assigned the value 0. The results are shown in *Table 26*.

Method-ology	Concepts Discussed	Planning Process	Organi-sing Process	Staffing Process	Directing Process	Control-ling Process	Project Manager Role	SPM Value
FDD	Yes	Yes	Yes	Yes	No	Yes	Yes	6
MSF	Yes	Yes	Yes	No	No	No	No	3
OOAD	Yes	No	Yes	Yes	No	No	Yes	4
RUP	Yes	Yes	Yes	Yes	Yes	Yes	Yes	7
SSADM	No	No	No	No	No	No	No	0
XP	Yes	Yes	No	No	No	Yes	Yes	3

Table 26: Support for Project Management Results

It is important to note that SSADM (Goodland and Slater, 1995) does not contain any project management activities. However, the concepts (1995, p. 442); the project manager role (1995, p. 444); and *Planning*, *Organising* and *Controlling* activities (1995, pp. 446-448) are discussed in the reference material, but only in terms of the PRINCE (PRojects IN a Controlled Environment) project management methodology. This methodology is separate to SSADM and thus these results are not included in the table above.

OOAD discusses the *Planning*, *Organising* (Booch, 1994, pp. 268-270) and *Staffing* (Booch, 1994, pp. 271-275) activities, grouping both the *Planning* and *Organising* activities under planning, where he refers to determining, scheduling and allocating work tasks to be performed. Booch (1994, p. 274) also describes the role of the project manager, albeit very briefly. FDD has a specific focus on *Planning*, informed by its Plan By Feature process (Palmer and Felsing, 2003, pp. 67-68, 150-154), which also includes *Organising* activities such as the allocation of tasks to team members. The role of the project manager is described (Palmer and Felsing, 2003, p. 28, 161-164) as is the *Staffing* activity through the formation of specific project teams, such as the Forming Feature Team Task. The *Controlling* activity is represented through the Code and Design Inspection Milestones (Palmer and Felsing, 2003, p. 77), and the Tracking Progress by Feature task (Palmer and Felsing, 2003, pp. 77-78). The Directing activity is not, however, covered. Interestingly, of the methodologies evaluated, FDD offers the second highest level of support for project management.

The RUP offers the most support for project management of the methodologies evaluated, meeting all of the criteria used for evaluation. Project management concepts are discussed in *Project Management: Overview* (IBM, 2004), the project manager role described in *Role: Project Manager* (IBM, 2004), and project management activities are described as follows (from IBM, 2004):

- Planning: *Workflow Detail: Plan the Project.*
- Organising: *Activity: Define Project Organisation and Staffing.*

- Staffing: *Activity: Acquire Staff.*
- Directing: *Activity: Initiate Iteration.*
- Controlling: *Workflow Detail: Monitor and Control Project.*

The other methodologies that could be considered to offer less, or no, support for project management are the MSF and XP. The MSF (Microsoft, 2003) discuss the concepts of project management (2003, pp. 9-10) but do not specify the role of a project manager, stating specifically that the MSF team model "does not contain a project manager role" (2003, p. 10). The *Planning* activity is discussed as part of the Planning phase of the MSF life cycle (Microsoft, 2003, pp.17-19), and *Organising* activities are described within the MSF Envisioning phase (pp.15-17). *Staffing*, *Directing* and *Controlling* activities are not discussed. Like the MSF, XP does not discuss the *Staffing* and *Directing* activities, focusing instead on the *Planning* activity, which is included as part of The Planning Game task (Beck, 1999, pp.47-48), and the *Controlling* activity, as discussed by Jeffries *et al.* (2000, pp.171-181) as 'Steering'. Jeffries *et al.* (2000, pp. 14-16) also describe the role of the project manager, under the more generic title of 'manager'.

From the evaluation above, it can be seen that there is a substantial difference in the level of support for project management within the methodologies evaluated. SSADM, on one hand, offers no intrinsic support for project management, but is structured to work in conjunction with a project management methodology such as PRINCE. Other methodologies, specifically the RUP and FDD, offer substantial support for project management activities. None of the methodologies evaluated, with the exception of SSADM (which had some level of project management discussion, but only from the perspective of the PRINCE methodology), disregard project management as necessary project activity, and all offer some level of support for it.

Conclusion

It would appear that the suggested measures of support for project management are applicable and valid, but it may be of benefit to take into account methodology dependence on external project management frameworks and methodologies. Another possible measure for this would be an analysis of methodology progress tracking techniques, but this may not be of sufficient detail, or cover all aspects of project management support.

Documentation Quality

According to Smart (2002, p. 132), the following are quality dimensions for documentation:

- *Easy to Use*, encompassing aspects such as task orientation, accuracy and completeness.
- *Easy to Understand*, encompassing aspects such as clarity, concreteness and style.
- *Easy to Navigate*, encompassing aspects such as organisation, retrievability and visual effectiveness.

Van Belle (2003, p. 172) suggests that the following are easily-measured attributes of documentation quality:

- *Completeness,* the proportion of methodology concepts defined in the documentation. In an ideal case, all methodology concepts will be defined. This attribute links to Smart's *Easy to Use* and *Easy to Navigate* dimensions (2002, p. 132).
- *Extensiveness or depth*, the amount of detail or description provided for each methodology concept. Van Belle (2003, p. 172) reasons that the more explicit/comprehensive the definitions, the better the quality of the documentation. This attribute links to Smart's *Easy to Understand* dimension (2002, p. 132).
- *Examples*, the provision of real-world examples or instances. This attribute links to Smart's *Easy to Navigate* dimension (2002, p. 132).
- *Readability of the descriptions or definitions*, the measure of the language used in methodology documentation against the language used by the target audience. This attribute links to Smart's *Easy to Understand* dimension (2002, p. 132).

In terms of measuring these aspects Dufty *et al.* (2004, pp. 14, 15) state that, of the quantitative measures available to measure readability and text quality, "the most influential of these are the Flesch Reading Ease Score and the Flesch-Kincaid Grade Level". The Flesch Reading Ease (FRE) and Flesch-Kincaid Grade Level (FKGL) measures combine the number of syllables per word and average sentence length to produce a readability measure where the (Dufty *et al.,* 2004, pp. 14, 15):

- FRE produces a score between 0 and 100 with higher numbers indicating texts that are easier to read and comprehend.
- FKGL assigns a number between 1 and 12 that is intended to be an approximation to the appropriate grade level of readers of the text in question, where the higher the result, the more difficult the text is to read and comprehend.

Other techniques suggested to measure text quality include the use of automated tools (Dufty *et al.*, 2004, p. 15) such as E-rater, which focuses on evaluation of completed documents on three levels: rhetorical structure theory, formal structure, and topical analysis; and Coh-Metrix, which evaluates documents on text-based cohesion and mental model-based coherence.

Completeness

The following table (*Table 27*), describes the result of the documentation completeness analysis of the methodology reference material. Where possible, other methodology reference material was used to give a more complete picture of documentation completeness. These reference materials are discussed further in the results discussion.

Methodology	Index	Text	Glossary	OCV	Overall Completeness
FDD	Yes	Yes	No	4	Very Good
MSF	Some	Yes	Yes	5	Excellent
OOAD	Yes	Yes	Some	5	Excellent
RUP	Yes	Yes	Yes	6	Perfect
SSADM	Yes	Yes	Some	5	Excellent
XP	Yes (3/3)	Yes (3/3)	Yes (1/3)	4.67	Very Good

Table 27: Documentation Completeness

In this case, the measure of Overall Completeness (OCV) is similar to Van Belle's (2003, p. 172) and is a composite index where:

OCV = [Index] + [Text] + [Glossary]

Each variable (*Index*, *Text* and *Glossary)* will be assigned the value:

- *2* if the coverage of the criteria is complete (i.e. the Text score is *2* if the methodology reference material explicitly defines or explains all, or most, of the methodology entities.
- *1* if the coverage of the criteria is partial (i.e. the Index score would be *1* if the methodology reference material has an index which includes some, but not all of the methodology elements).
- *0* if there is no coverage of the criteria, (i.e. the Glossary value would be *0* if the methodology reference material does not contain a glossary).

The Overall Completeness value (***OCV***) is thus a subjective interpretation of the documentation completeness, based on the composite score for the criteria, ranging from *0* (None) to *6* (Perfect). From the results shown in table 29, the reference materials all obtain high scores, with one set of reference materials, the RUP, having a perfect OCV score. Three other sets (OOAD, the MSF and SSADM) have an OCV value of Excellent and two (XP and FDD) have overall completeness values of Very Good. In the case of this criterion, there is little to differentiate the methodologies analysed.

Use of Examples

The Extreme Programming reference materials use a substantial number of examples, with Wake (2000) and Jeffries *et al.* (2000) also providing examples in programming languages and pseudo-code. However, no examples are specified in the Rational Unified Process (IBM, 2004). Most of the examples in the Feature-driven Development reference material are examples of specific deliverables, such as Work Packages (Palmer and Felsing, 2002, p. 174) and Code Inspection Reports (Palmer and Felsing , 2002, p. 187). The MSF provides a complete set of examples on an accompanying CD, as well as providing examples throughout the text in the form of an ongoing exercise (Adventure Works Cycle Application, p. 25). Both SSADM and the OOAD use examples throughout the reference material – the SSADM frequently uses figures to describe deliverables (i.e. p. 108, where an Entity Description form is shown).

Readability

In terms of measuring readability, the following measures were used, as they are both easily available in the Microsoft Word word-processing application:

- the *Flesch Reading Ease* (FRE) score, which rates textual documents on a scale of 0 to 100, where the higher the number, the easier the document is to understand (Hargis, 2000, p. 126); and
- the *Flesch-Kincaid Grade Level* score, which rates textual documents at United States school grade level, from 0 (grade 0) to 12 (grade 12). The lower the FKGL value, the more readable the textual document is (Dufty *et al.*, 2004, p. 15).

The FRE uses the following formula:

FRE = 206.835 – (1.015 x Average Sentence Length [ASL]) – (84.6 x Average Syllables per Word [ASW])

Van Belle (2003, p. 176) states that, for most standard documents, one should aim for a score of approximately 60 to 70 in general writing.

The FKGL, on the other hand, uses the following formula:

FKGL = (0.39 x ASL) + (11.8 x ASW) – 15.59

The readability analysis was performed against the methodology reference material. In this case, more than one set of reference material was used for XP, as more than one set of reference material was used throughout the analysis of the methodology. However, average scores are shown in the table below (full results are shown in Appendix C). The data used for the analysis was based on two to four samples of 2 full pages each, selected at roughly 50 page intervals. Thus, a 200 page book would, theoretically, have samples drawn from pages 49 and 50, 99 and 100 and 149 and 150. Finally, where selected sample pages contained diagrams or large expanses of whitespace, alternate sets of pages were selected, as close in page number to the selected sample pages as possible.

The scores for the methodology reference material are as follows (*Table 28*) along with composite documentation quality results, where the OCV and FKGL have been normalised as percentage values in the following manner:

- **OCV% = OCV / 6**
- **FKGL% = (12 – FKGL)/12**

The FRE is already represented as a value out of 100, and thus does not require normalisation to be shown as a percentage. Also included is an overall documentation quality value (DQV), which is an average of the OCV%, FRE% and FKGL%.

	FDD	MSF	OOAD	RUP	SSADM	XP
FRE %	45%	32%	27%	36%	33%	***61%***
FKGL	12	12	12	12	12	***8.53***
FKGL %	0%	0%	0%	0%	0%	***28%***
OCV %	67%	83%	83%	***100%***	83%	78%
DQV	37%	38%	37%	45%	39%	***56%***
Rank	5	4	5	2	3	***1***

Table 28: Readability and Overall Documentation Quality Results

The readability results are illuminating, specifically in terms of the results for the Agile methodologies which are the two highest and, in the case of XP, substantially higher than all other methodologies. All of XP's reference materials score around 60 on the *FRE* scale, while none of the other methodologies reference materials, with the exception of Feature-Driven Development (44.5), score over 40. In fact, the average for the other methodologies is 35.65, more than 25 points less than the average value for the XP references. Booch's OOAD has the lowest *FRE* value, almost five points less than the next-lowest. Also important to note is that the XP references are the only reference materials which score less than 12 on the *FKGL* scale (average: 8.53). It is important to note that all three XP reference materials score below 9 on the *FKGL* scale. It can be seen from this analysis that XP reference materials are substantially more readable than any other methodology references, which are found to have low readability values.

In terms of the overall results, while XP and the RUP were methodologies with the highest documentation quality, but there is very little difference between the other methodologies. XP is arguably highest ranked due to it's readibility scores, which were substantially higher than the others, while RUP is the second highest due to the completeness of it's documentation.

Conclusion

The *Flesch Reading Ease* and *Flesch-Kincaid Grade Level* would appear to be good measures of methodology documentation quality. However, when taking into account the level of sophistication of the intended audience, the *Flesch-Kincaid Grade Level* measure would probably not be of substantial value as the methodology reference material are mostly aimed at an audience with at least a Grade 12 education. Other measures of this criteria could include the use of other automated tools such as e-Rater, which would probably return results of the same type, if not the same results. The other measures used, specifically documentation completeness measures, appear to be satisfactory ancillary measures, most effective when used in combination with the *Flesch Reading Ease* scale.

In conclusion, this chapter has presented the following:

- An introduction to the criteria that make up the intrinsic aspect of the proposed framework.
- A discussion of the measures used for each criteria, and results for the methodologies analysed and evaluated for each of the intrinsic criteria.

The following chapter addresses the final aspect of the proposed framework, the pragmatic view.

Chapter 7: Pragmatic Analysis

This chapter discusses the criteria for, and the evaluation of, the final dimension of the proposed framework: the pragmatic. Chapters Five and Six discussed the formal and intrinsic view and the following chapter will present the conclusions. The pragmatic view, according to Van Belle (2003, p. 216), is concerned with aspects of the objects being studied which cannot be assessed purely on the information contained within the reference material, but which requires the consideration of information regarding the use, environment and context of the methodologies. This chapter involves a short overview of the pragmatic criteria, after which the results of the pragmatic analysis are presented.

Pragmatic Criteria

The criteria used in the pragmatic view are:

- *Methodology Situationality,* the degree to which the methodology is suitable for specific situations or contexts.
- *Methodology Flexibility*, the degree to which the methodology can be manipulated and adapted for different contexts or domains.
- *Methodology Effectiveness*, the ability of the methodology to provide solutions to problems.
- *Methodology Validity*, the degree to which the methodology is considered valid.
- *Methodology Availability and Cost*, the degree to which the methodology is available, and the cost of purchasing reference materials.
- *Methodology Currency and Maturity*, the degree to which the methodology has undergone levels of revision, and the regularity with which the methodology is revised for changing environments.
- *Methodology Support*, or the level of support that is available for the methodology.

The criteria are discussed in more detail, and the results of the analysis of the methodologies described, below.

Situationality

Jayaratna (1994, p. 231) states that methodology creators "should explicitly consider what environmental conditions/settings are suitable for the application of the methodologies and what dimensions of the organisation they aim to transform". In doing this, Jayaratna (1994, p. 231) argues that methodology creators help their methodology users to question what kind of situations they face and what, if any, changes could or should be made to adapt the methodology for that situations.

McLeod (1992) states that a methodology's success in a specific situation is contingent on the quality of the methodology as well as upon its appropriateness to the application domain, the political climate regarding its use, the skills and backgrounds of the methodology users, the effectiveness of the

methodology training, and the attitude of the methodology user community. Arni-Bloch (2005, p. 2) refers to the applicability of the environmental context of methodologies as situationality, and argues that, in order for methodologies to be completely effective in all situations, situational method engineering (SME), or the construction of a methodology to adapt perfectly to its situation, is required.

In order for SME to be applied, Rolland (1997, p. 16) proposes that it is imperative to understand the situation, or project environment, in which the resulting methodology will be applied. While this research does not extend to the realms of method engineering, methodology situationality may well be an important attribute in the selection of a methodology for a specific project.

However, it appears that no research exists to compare the merits of the situations that methodologies believe they are most applicable for, as this type of relativist approach may not be suitable. Accordingly, this section will attempt to approach the analysis of methodological situational applicability from as descriptive a perspective as possible, discussing the aspects that Fitzgerald (1998, pp. 326, 327) states are relevant to the adherence to methodologies, namely:

- *Project Size/Scale*, or the size and scale of the project and team recommended for and/or against by the methodology reference material.
- *Project Type*, or the type and attributes of the project for which the methodology is recommended for and/or against by the reference material.
- *Project System*, or the type of system that the methodology is recommended for and/or against the development of by the reference material.

Some of these criteria are also used by Mannino (1987, pp. 33, 34), who used a slightly more relativist perspective than the one used in this research. One of the results of his comparative research into structured methodologies was that some are not suitable for certain project sizes, and some are more applicable to the development of certain types of systems (those listed include real-time, data processing/database, science/engineering and operating systems) (Mannino, 1987, pp. 33, 34). Both of these results are consistent with the results described below, although an interesting point to note is that none of the methodologies mention specific types of project systems that they are not recommended for, rather focusing of projects and project types where they have been used with success (see OOAD for an example of this).

FDD

Palmer and Felsing provide very little detail regarding the project attributes described above, but do imply that the use of FDD is not limited by team size (2002, p. 261).

The MSF

The only descriptions of project attributes provided by Microsoft are that the MSF does not function well under rigid, dictatorial project management as it requires decisions to be made by consensus (2003, p. 10) and that the MSF is intended to work regardless of the type of project being deployed (2003, p. 411). No other information regarding intended suitability is provided.

OOAD

Booch (1994, p. v), in describing the second edition of Object-Oriented Analysis and Design with Applications, states that one of the reasons for updating the material was due to OOAD being used successfully on projects as diverse as the administration of banking transactions, management of public utilities, mapping of the human genome and the automation of bowling alleys, and that, as of 1994, "many of the next generation operating systems, database systems, telephony systems, avionics systems and multimedia applications [were] being written using object-oriented techniques". This would imply that OOAD is suitable for the development on most, if not all, types of systems, provided that they are, according to Booch (1994, p. 4), "industrial strength software", or complex applications with rich sets of behaviours, often processing or managing large amounts of data, and allowing the project team to "engineer an illusion of simplicity" (1999, p. 25).

One of the prerequisites of OOAD is, of course, the use of an object-oriented programming language, such as Smalltalk, C++, Java, Object Pascal or Eiffel (Booch, 1994, p. 32). Booch (1994, p. 234) implies that OOAD can work with teams of almost any size, stating that smaller, more tight-knit teams require less formality in the process, while large, geographically-dispersed teams require more formality in the process. Booch does note that in terms of team sizes, teams consisting of "fewer and better people" are preferable (1994, p. 274). In terms of client system attributes, OOAD is described by Booch (1999, pp. 285, 286) as being suitable for projects involving:

- Detailed user interface requirements, which are handled through iterative prototyping.
- Different types of database access, such as distributed and pre-existing schemas, which are handled through the principle of separation of concern.
- The development of real-time systems, through simple, optimised architectures.
- Integration with legacy systems, through well-defined, modular interfaces.

The RUP

IBM (2004:*About This Configuration*) discusses the suitability of the RUP template configuration used for this research stating that it is suitable for small projects of up to 15 people that are located in the same building and lasting less than a year. However, they (IBM, 2004:*Activity: Tailor the Process for the Project*) also state that a full RUP implementation would normally entail a level of process tailoring, based on:

- The development organization's process maturity.
- The size of the project in terms of calendar time and number of development resources.
- The project members previous exposure to similar processes.
- The formality requirements of the project.

No mention is made of the types of project situations that the RUP is most suited to, or the types of system it is most suitable for developing

SSADM

SSADM provides very little detail regarding project suitability, although one aspect that is discussed is it's support for structured project management methodologies such as PRINCE (Goodland and Slater, 1995, p. 442). Another aspect discussed, albeit in very little detail, is that SSADM was the government standard methodology for projects in the United Kingdom (Goodland and Slater, 1995, p. 5), although this may not still be the case, given that its last update was 1995 (Maybank, 1999).

XP

XP has quite rigid requirements for project suitability, and Beck (1999, pp. 8, 116-122) goes into much more detail regarding these than the other methodologies analysed, stating that XP is designed to work with projects that can be built by teams of two to ten programmers, that aren't sharply constrained by the existing computing environment, and which is of the kind of size where tests to cover all functionality can be run in a very short time (specifically a "fraction of a day"). Beck also discusses how XP can be used to support both outsourced and insourced projects, projects with completion bonuses and early termination, and the development of frameworks and shrinkwrap products (1999, pp. 121, 122), in addition to the types of projects that XP is normally used on. XP is, however, not suitable for projects where a "complete specification or analysis or design" is needed before programming begins, or where the team consists of more than about 20 programmers, as "the amount of functionality to be produced and the number of people producing it don't have any sort of simple linear relationship" (Beck, 1999, p. 116).

Beck (1999, pp. 117-121) provides substantial detail of the other types and attributes of projects for which XP is not recommended. These are typically projects where:

- A technology with an exponential cost curve, that requires substantial planning ahead in architecture and design, is to be used.
- A long time is needed to gain feedback, such as systems which take extended times to compile and link, as continuous integration is hampered, or impossible. Beck states that "[i]f you have to go through a two-month quality assurance cycle before you can put software in production, you will have trouble learning enough to be successful" (Beck, 1999, p. 118).
- Programmers are geographically dispersed, especially if the project team are dispersed within the same building.
- The project is on a fixed price or time and materials basis, as there is often underlying tension due to the goals of the supplier being at odds with the goals of the customer.

Conclusions

As shown above, the methodology reference materials cited describe ideal situations for the methodologies analysed in varying degrees of detail. Some, like XP (Beck, 1999) and OOAD (Booch, 1994), go into enough detail to suggest optimal project situations or how the methodology can be situationally adapted to cope with the requirements of specific project types. Others, like FDD, the MSF and SSADM go into very little detail, perhaps preferring to err on the side of genericity. It is not

within the realm of this research to argue against this, but it may be worthwhile to point out that the practice of situational method engineering (as described by Arni-Bloch, 2005, and Rolland, 1997) exists as a result of the need to tailor methods not applicable for certain situations. XP, on the other hand, refuses to err on the side of genericity, as Beck spends an entire chapter of *Extreme Programming Explained* describing when XP should not be used (1999, pp. 116-118), concluding with "there is absolutely no way you can do XP with a baby screaming in the room" - which may well be applicable to all of the methodologies analysed in this research.

Methodology Flexibility

McLeod (1992) states that full methodologies are often large and unwieldy and, as a result, qualities of a methodology which allow only tasks and deliverables relevant to the current project to be extracted from the methodology, and specialised, are very valuable, as long as this is achieved without a "loss of integrity and respecting dependencies". As such, the extent to which a methodology allows this would be the extent to which it is flexible, customisable and enhanceable. Methodology flexibility is also often associated with modifiability and extensibility.

The Object Agency (1995, p. 53) describe extensibility as an attribute of something that allows it to last or continue, or to be expanded in range or scope; modifiability is defined as the extent to which a methodology facilitates the incorporation of changes, once the nature of the desired change has been determined. Kelley and Rogers (1992, p. 177) describes methodology flexibility as:

> *extensibility, tailorability, and scalability. A framework should support incremental building so that portions of the environment can be acquired as they are rather than all at once. The framework should also be tailorable to accommodate the specific functions of an organization as well as to accommodate new opportunities that might arise. Scalability is the ability of a framework to scale up or down to meet the specific needs of a project.*

As an example of tailorability, Keenan (2004, p. 1) describes a process for agile methodology process tailoring, which is shown in *Figure 20*. This process is applicable on two levels, namely the framework level, where knowledge about the process exists, and the application level, whether the methodology is applied to a specific problem. During the use of the methodology, or the application of the process, Keenan (2004, pp.1, 2) argues that there should be feedback to adapt the process on the framework level, and that there should be a separate feedback loop to the framework level as part of a monitor and control step.

This will also feed back into the design development process step, which should allow *dynamic tailoring* of the process. The information gathered as part of the application and monitor and control steps should also then be used, on the framework level, as part of a *static tailoring process* to update the methodology framework (Keenan, 2004, p. 1). This process is similar to that described by McLeod (1992), who believes that methodologies should not remain static over time, but should be updated on a per-release basis and should instead evolve and become tuned to the organization, its culture, infrastructure and technology. McLeod (1992) believes that this can be achieved by "having the

method explicitly defined in the model, and incorporating method review points into the life cycle". In this way "projects can take advantage of lessons which were only learnt in the previous phase of a concurrent project" (McLeod, 1992).

A number of ways of measuring methodology flexibility have been suggested. Kelley and Rogers (1992, p. 178) recommend the use of a test case-based approach, where test cases are implemented using methodologies selected for analysis. The test case must necessarily contain aspects intended to assess methodology flexibility, amongst others (Kelley and Rogers, 1992, p. 178), and the result of the application of the test case should determine the areas in which the methodology "performed well and those in which it performed poorly". These can then be used to provide a set of recommendations.

In the case of this analysis however, the approach suggested by TOA (1995, p. 53), using criteria derived from TOA (1995) and Keenan (2004), was followed, as it specifically focuses on the evaluation of reference material. The following aspects of methodology flexibility were selected for analysis and evaluation:

- The level of discussion of methodology flexibility/customisability.
- Whether a process was described to support:
  - Static; and/or
  - Dynamic methodology tailoring (Keenan, 2004, in *Figure 20* below).
- Whether a specific project role is identified to drive the customisation process.
- Whether there are tools available to support the customisation or enhancement of the methodology (further detail on these tools is provided in the Support section of the pragmatic analysis).
- Whether heuristics were provided for the methodology customisation process, or processes described above.
- Whether examples were provided for the methodology customisation process, or processes described above.

Finally, a composite value (*MFC*) was given to each methodology based on their results in each criteria where Yes = 1, No = 0. No weightings have been ascribed to the values. The results of this analysis are described below, in *Table 29* .

	FDD	MSF	OOAD	RUP	SSADM	XP
Concepts Discussed	Yes	No	Yes	Yes	Yes	No
Process (Static)	No	No	No	Yes	Yes	No
Process (Dynamic)	No	No	No	Yes	No	No
Role Defined	No	No	No	Yes	No	No
Supporting Tools	No	No	No	Yes	Yes	No
Heuristics (Static)	No	No	No	Yes	Yes	No
Heuristics (Dynamic)	Yes	No	No	Yes	No	No
Examples (Static)	No	No	No	No	Yes	No
Examples (Dynamic)	No	No	No	No	Yes	No
MFC	2	0	1	***7***	5	0
Ranking	3	6	4	***1***	2	6

Table 29: Methodology Flexibility Analysis Results

From the results described above, it may be interesting to note that FDD offers some level of flexibility in discussing the concepts and providing heuristics, albeit at a low detail (Palmer and Felsing, 2003, pp. 259-261). The most customisable and flexible methodologies are the RUP and SSADM. The RUP is the only methodology that approaches methodology flexibility from both the static tailoring and dynamic tailoring perspectives. SSADM, in keeping with its structured nature, only approaches methodology tailoring from the static perspective, describing how to adapt the process before it is put into practice (Goodland and Slater, 1995, pp.458 – 461). Both SSADM (Goodland and Slater, 1995, pp.454-456) and the RUP (IBM, 2004:*Concepts: RUP Tailoring*) contain discussions of customising the methodologies for use and the reasons why this activity would be undertaken, and both provide a process for this activity. The process described by the RUP is arguably more thorough in their coverage of methodology customisation in that they provide a specific activity (IBM, 2004: *Activity:Tailor the Process for the Project*), specify the Process Engineer Role and provide guidance for small projects customisation (IBM, 2004:*Concept Tailoring the Process for a Small Project*).

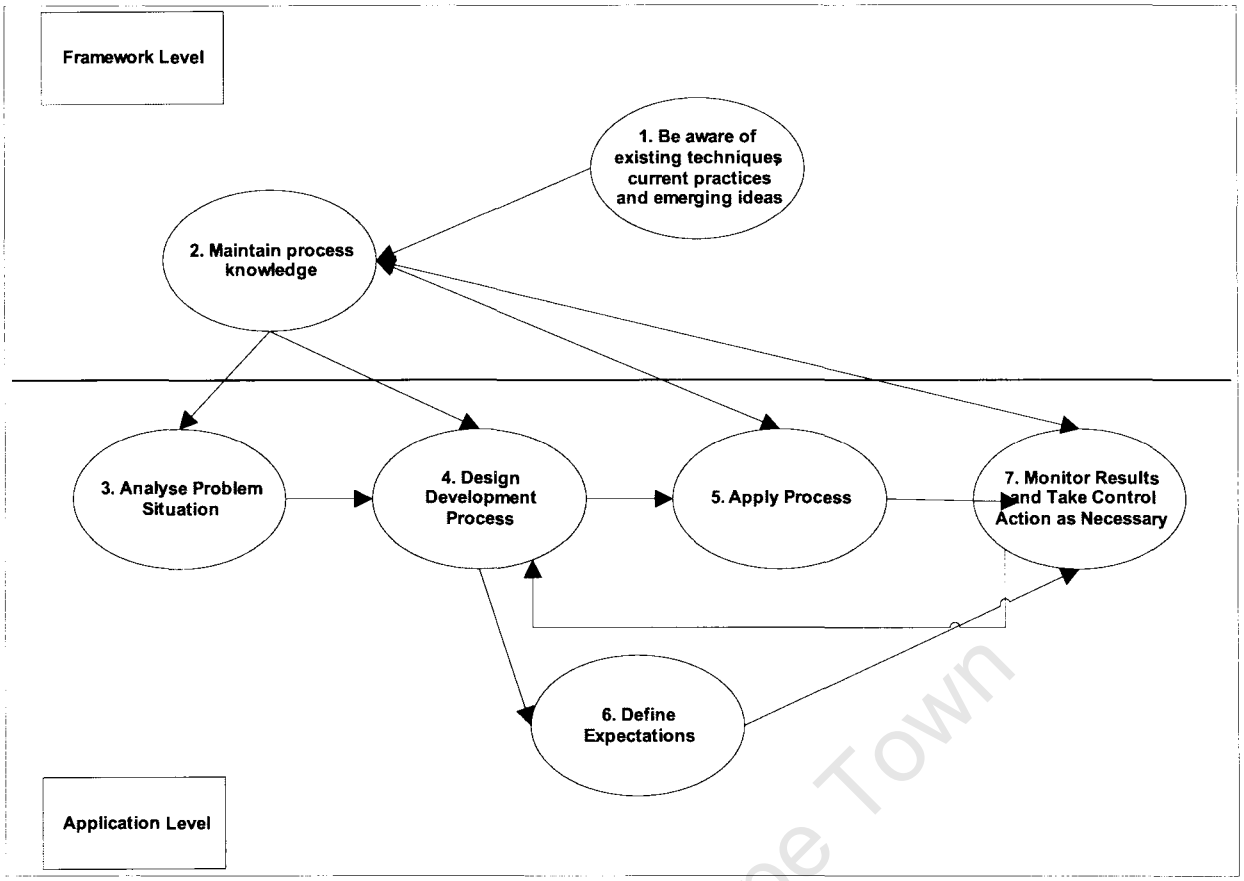

Figure 20: Agile Process Tailoring (Keenan, 2004, p. 1)

Both the RUP and SSADM provide related heuristics, and SSADM provides some examples, although lacks detail (Goodland and Slater, 1995, pp. 458 – 461). While the RUP is available in an electronic format, SSADM is available in book format and in commercial electronic format, and both have associated commercially-available customisation tools (an example of an SSADM tool for this purpose is Aonix Europe's Select SSADM). While Booch (1994, p. 234) broaches the subject of methodology flexibility, it is at a very low level of detail, stating that "every project is unique, and hence developers must strike a balance between the informality of the micro process and the formality of the macro process", and that too much formality can stifle innovation and too little "could lead to chaos", so a balance should be struck on a per-project basis.

The least flexible methodologies, in terms of this analysis, are XP and MSF which do not discuss methodology tailoring or customisation at all. Despite being an agile methodology, XP is significantly inflexible: Beck (2000, p. 54) states that XP provides 12 supporting activities that must be followed together as the weaknesses of each are made up for by the strengths of others. There is no discussion of process tailoring, customisation or adaptation in the XP reference material. FDD (Palmer and Felsing, 2003, pp. 259-261), on the other hand, is more flexible, as it does discuss the concepts of tailoring the methodology to specific projects. FDD provides some heuristics relating to dynamic process tailoring, specifically in terms of changing project processes gradually during the life cycle of a project, but in a low detail (Palmer and Felsing, 2003, pp. 259-261). However, Palmer and Felsing (2003, p. 259) state that "the ability of the process to change and adapt to people, technology, and business issues is critical".

In terms of anecdotal evidence to support the conclusions drawn above, Fitzgerald (1997, pp. 6, 7) found that, through an empirical study of 162 organisations, SSADM was the most popular basis for methodologies developed by organisations internally, but based on commercial methodologies. In a second study, Fitzgerald (1997, p. 3) found that a quarter of organisations surveyed used a methodology derived from SSADM. A conclusion could be drawn from this that SSADM is a methodology that lends itself well to customisation. The RUP is a another method that appears to lend itself to customisation or adaptation, as it has formed the basis of a number of methodologies such as OpenUP (the Open Unified Process) and the EUP (Scott Ambler's Enterprise Unified Process) (Ambler, 2006[1]).

In terms of the agile methodologies, Ambler (2006[2], p. 2) found that, while XP had been adopted at about 23% of respondents' organisations, the level of adoption of XP's practices at respondents' organisations differed in the following ways:

- Code refactoring was adopted in around 36% of respondents' organisations.
- Pair programming was adopted in around 14% of respondents' organisations.
- Active stakeholder participation, or on-site customer, was adopted in around 23% of respondents' organisations.
- Common coding guidelines, or coding standards, was adopted in around 39% of respondents' organisations.
- Continuous integration was adopted in around 27% of respondents' organisations.

From these figures, an inference could be drawn that XP's usage tends to be based around the adoption of individual practices (such as coding standards and code refactoring), as opposed to a wholesale adoption of all XP practices, which may imply that it is not as inflexible as it appears.

Conclusion

The measures for this criteria, while almost intrinsic in their basis, appear to have sufficient face validity to warrant it's use. An advantage offered by a measure of this type is that it is customisable based on the tailoring framework used – a different framework could suggest the evaluation of different aspects of methodology flexibility, and this would not drastically affect the validity of the measure, provided that the quality and applicability of the framework was satisfatory.

Effectiveness

Kelley and Rogers (1992, p. 177) state that a methodology's power or effectiveness, is its ability to increase the productivity of the individuals using the environment built upon the methodology to do what is required to complete the project. Methodological effectiveness involves the methodology's capacity to manage life cycle phases, activities, roles and products (Kelley and Rogers, 1992, p. 177). According to McLeod and Roeleveld (2002, p. 1), a methodology's effectiveness is based upon its ability to potentially solve a business problem related to the implementation of an information system.

According to McLeod and Roeleveld (2002, p. 2), Jayaratna's (1994) Normative Information Model-based, Systems Analysis and Design (NIMSAD) framework rates a method's capacity to formulate a problem, design a solution, and implement the solution design, as well as focusing on the context for applying the method, the method user, and the method, or problem-solving process, itself.

As such, McLeod and Roeleveld (2002, p. 2) state that, as systems analysis and design method is a structured technique for solving a particular kind of problem, the NIMSAD framework can be used to determine the effectiveness of such methods to solve business problems. On the other hand, Kelley and Rogers (1992, p. 177) believe that a methodology's effectiveness is best measured by using the methodology to implement a reference model and then measuring "the overall leverage provided by the framework to do complex, undesirable tasks, or repetitive tasks" which can be weighted "against the overhead associated with using the framework". If the net result is positive, Kelley and Rogers (1992, p. 177) argue, the framework provides effectiveness and power to the overall project.

As this research does not include an empirical aspect, the NIMSAD framework will be used to assess the effectiveness of the methodologies analysed. The NIMSAD framework (Jayaratna, 1994) examines methodologies from three perspectives:

1 The problem situation, or scoping and framing the problem.

2 The method user, which has two elements:

  2.1 The apparent usability of the methodology in the eyes of the users (the mental construct).

  2.2 The ability of the methodology's mental construct to explain the products of the methodology (the desirability of the mental construct).

3 The problem solving process, which has a number of elements:

  3.1 Problem formulation, containing:

    3.1.1 Understanding the situation of concern, or the problem area.

    3.1.2 Performing the diagnosis, or the understanding the need for a new system

    3.1.3 Defining the prognosis outline, or determining the desired situation.

    3.1.4 Defining problems, or investigating the feasibility of the desired solution.

    3.1.5 Deriving notional systems, or defining hypothetical systems to solve the business problem from different perspectives.

  3.2 Solution design, containing:

    3.2.1 Performing the logical/conceptual design.

    3.2.2 Performing the physical design.

    3.2.3 Implementing the design.

There are, however, a number of points to note. Firstly, there appears to be a level of subjectivity in performing a NIMSAD-based analysis of a methodology. For example both McLeod and Roeleveld (2002) and Wagner (2003) performed NIMSAD-based analyses of the RUP with results that differed based on the lenses used. Wagner's analysis (2003) appeared to be descriptive, while McLeod and Roeleveld's analysis (2002) was comparative, where the RUP was compared against the Inspired Method. McLeod and Roeleveld's analysis of the RUP (2002) was used as a 'test case' for the purposes of this research, since this research has a similar, relativistic approach, while Wagner's (2003) analysis was not used, and the RUP was not re-analysed for the purposes of this research. The results of this analysis are discussed below. In the interests of brevity, the Performing the Logical/Conceptual Design and Performing the Physical Design NIMSAD elements have been incorporated into a single element for the purposes of this research.

Problem Situation

Jayaratna (1994, p. 57) states that it is important for a methodology to take into account the organisational context in which the project will occur. The RUP, for example, takes aspects of this into account during the Requirements Gathering activity during the Inception phase, and optional Business Case, and represents the organisational context with Use Case diagrams (IBM, 2004:*Requirements:Overview*; McLeod and Roeleveld, 2002, p. 2). McLeod (2002, p. 2) notes, however, that the RUP does not cover the managing of political problems within the organisation. XP appears to be more concerned with user requirements, specifically derived from an Onsite Customer (who may provide some insight into the problem context and environment), than with understanding the current organisational context (Beck, 1999, p. 52).

While FDD (Palmer and Felsing, 2002, p. 57) specifically focuses on developing a model of the future system, and not the current problem situation, part of this process involves domain owner providing insight into their domains. No real effort seems to be expended in either XP or FDD processes to understand the situation of concern. OOAD also appears unconcerned with understanding the situational aspects of the project – while Booch (1994, p. 253) states that the project team should "attempt to benefit from the experience of other projects that had to make similar development decisions". No mention is made of trying to understand the organisation, its context or its politics.

SSADM (Goodland and Slater, 1995, p. 10), on the other hand, uses its first stage to investigate the current environment, resulting in a large number of situation-related deliverables such as a project scope, activity network, current context diagram, user catalogue and requirements catalogue, all of which are intended to reflect current aspects of the organisation. No mention is made of understanding political aspects of the organisation. The MSF appears to have some level of concern with gathering information about the current organisation and situation, by suggesting the use of highly interactive techniques such as shadowing, interviewing, focus groups, surveys and user instruction (Microsoft, 2003, p. 40). The information is then represented with high-level Use Cases and Usage Scenarios (Microsoft, 2003, p. 50).

While some of the methodologies analysed (SSADM, the RUP and the MSF) do appear to take some level of organisational context into account, some (XP,FDD and OOAD) do not and it may be argued that none of the methodologies analysed take this context into account to a particularly high level, or are particularly effective at this aspect of NIMSAD. Jayaratna (1994, pp. 206-210) believes that methodologies like Checkland and Scholes' Soft Systems Methodology, as they have a concept of *Weltanschauungen*, or 'world images' of those impacted by the methodology, consider this aspect very well.

Mental Construct

McLeod and Roeleveld (2002, p. 3) state that, as the RUP (and the UML) were created by more than one author and are therefore an amalgam of different mindsets and ways of thinking, it poses additional learning difficulties for new methodology users. The same can be said for XP (Beck, 1999; Jeffries *et al.*, 2000; and Wake, 2000 provide three of many XP references), FDD (created by Peter Coad and Jeff de Luca[6] and has since had input from Palmer and Felsing, 2002, amongst others), SSADM (an initiative of British government departments and an open standard[7]) and the MSF. Only OOAD (Booch, 1994) is the product of a single author. Secondly, McLeod and Roeleveld (2002, p. 3) suggest that, as UML specifications are large and explain the language using meta-metamodels, cognitative problems can be caused as meta concepts are usually only understood after prior instance modeling experience.

According to McLeod and Roeleveld (2002, p. 3), the UML is also a complex method with over 233 concepts for the new user to grasp. The RUP, FDD and MSF all suggest or mandate the use of UML (Palmer and Felsing, 2002, p. 22; IBM, 2004: *Best Practice: Model Visually*; Microsoft, 2003, p. 57). OOAD uses Booch's OO notation, which was a forerunner to the UML (Booch, 1994, p. 171; Alhir, 1998, p. 10) and can thus be said to at least share some level of the RUP's complexity. SSADM mandates the use of non-UML diagrams, which could be said to add even more complexity as UML is probably the most widely-used notation at present (Alhir, 1998, p. 8).

Only XP does not specify the use of a notation, and has a Coach project role whose purpose is to drive the application of the methodology (Beck, 1999, p. 8). XP also recommends the use of a metaphor to guide all development with a simple shared story of how the whole system works (Beck 1999, p. 46), in an attempt to simplify the problem and system domain. Finally, XP is, according to Beck (1999, p. 6), intended to be made up of common sense principles and practices that have been used prior to XP's introduction. As a result of this, it appears that XP has the most easily understood mental construct, while SSADM could be said to have the least easily-understood mental construct.

Construct Desirability

McLeod and Roeleveld (2002, p. 3) state that there needs to be understanding between the method users, and the method users and the client whose problem is to be solved, and that while UML "facilitates easy communication between users" of the methodology, the ability to use UML models to

6 Palmer and Felsing, 2002, p. xxiii.
7 Maybank, 1999.

communicate with clients is "questionable". While the same could be said for the MSF, due to its use of UML, this is mitigated slightly by FDD, which specifies the role of Domain Experts on the project team (Palmer and Felsing, 2002, p. 30) and thus facilitates a higher level of communication between the methodology users and the client (some of whom also become methodology users). FDD, given its use of both UML and clients on the project team (no other methodology provides both of these attributes) is probably the most effective methodology in terms of construct desirability. XP follows a similar track to FDD, in that it specifies the role of the Onsite Client, which is intended to create a high level of communication, normally verbal, between them and rest of the team (Beck, 1999, p. 52).

OOAD, on the other hand, uses the creation and review of prototypes to communicate with the client (Booch, 1994, p. 250), almost motivating for the project team to exist in a client-less vacuum until the prototypes are complete and communication can commence. While SSADM's Stage 1 attempts to introduce clients and possible future system users to SSADM techniques (Goodland and Slater, 1995, p. 10), the client is not very involved in the SSADM process and thus SSADM is probably the least effective methodology in terms of construct desirability.

Understanding the Situation of Concern

According to McLeod and Roeleveld (2002, p. 3), the RUP recommends using business use cases for scoping purposes, as well as identifying the situation of concern, as these have the advantage of being easy to use, and easy for clients to understand. OOAD's *Domain Analysis* activity (Booch, 1994, p. 253) centres around understanding the particular problem domain and states that "[b]efore setting out to implement an entirely new system, it is often wise to study existing ones". As mentioned earlier, OOAD focuses more on the system being constructed, than the reasons why the system should be constructed.

A similar approach is followed by both XP and FDD, which do not attempt to understand the reasons for the new system, but scope the proposed system instead. XP's Planning Game practice has an Exploration phase which is concerned with determining the functionality of the new system and representing it using User Stories (Beck, 1999, p. 73), while FDD uses the creation of Feature Lists to represent the functionality and possible scope of the system (Palmer and Felsing, 2002, p. 57). Of the methodologies analysed, XP, FDD and OOAD are probably the least effective at understanding the situation of concern, as they are more concerned with the required system than the current environment.

SSADM (Goodland and Slater, 1995, p. 10) takes the most detailed, and thus most effective, view of the situation of concern. Its Stage 1 (Investigation of the Current Environment) involves the investigation of the current system in order to learn its terminology and functions, data requirements and to set the boundaries of project. The result of this is a logical view of the current system, detailed in Data Flow, Process and Entity-Relationship diagrams. Requirements are also approached in the Investigate and Define Requirements activity (Goodland and Slater, 1995, p. 10). The MSF also approaches the issues that the business hopes to address, albeit in a "short narrative ... relat[ing] primarily to the current state of business activities" as part of the Vision Statement, which is one of the

final deliverables of the MSF's Envisioning phase (Microsoft, 2003, p. 91). Arguably, SSADM and the MSF are the most effective of the methodologies analysed at understanding the situation of concern, while OOAD, FDD and XP make no real attempt to do this.

Performing the Diagnosis

Neither OOAD, SSADM nor the MSF perform any kind of diagnosis or attempt to challenge any requirements which are, in the case of the MSF, almost entirely derived from clients, and consistent with McLeod and Roeleveld's view of the RUP (2002, p. 2). XP attempts to allow the project team to challenge the client on scope and requirements through the use of the Onsite Client, who must be empowered to make decisions regarding scope and requirements, and the YAGNI ("You ain't gonna need it") principle (Beck, 1999, p. 52; Jeffries *et al.*, 2000, p. 219). This makes it the most effective in terms of performing the diagnosis, as it is the only methodology analysed that challenges the clients in terms of their requirements. FDD allows the definition of scope and prioritisation of requirements through the *Build a Features List* and *Plan By Feature* processes (Palmer and Felsing, 2002, pp. 65-68) thereby allowing the project team, including the domain experts and the client to clarify the requirements of the features of the proposed system and then, based on the time available and time/cost requirements of the features, to sequence the construction of the system features.

Defining the Prognosis Outline

According to McLeod and Roeleveld (2002, p. 4), in terms of the RUP, the client's expectations are gathered in the requirements stage of the Inception phase. XP's prognosis outline definition involves the capturing of requirements onto User Story cards, described above, which are very short and low in detail and describe the future system requirements (Beck, 1999, p. 73). FDD's Build a Feature List process (Palmer and Felsing, 2002, p. 57) has also been described above, and serves the same purpose: to define the client's expectations for a system. OOAD's Conceptualisation phase "seeks to establish the core requirements of the system" (Booch, 1994, p. 250), specifically around a "new piece of software" such as "a new business venture, a new complimentary product in an existing product line, or perhaps a new set of features for an existing software system".

SSADM's Stage 2 (*Business Systems Options*) is used to reflect different ways in which a system can be organised to meet requirements and involves using creative thinking to determine ways in which a new system could meet the organisation's objectives (Goodland and Slater, 1995, p. 136). These options may include non-information system-based suggestions (Goodland and Slater, 1995, p. 137). The MSF, on the other hand, provides a Project Scope (defined using use case diagram) and a solution concept in the Vision Statement document (Microsoft, 2003, pp. 95-101, 127) which also includes high-level requirements and high-level system scenarios to describe an information system-based solution to the business requirements. All methodologies analysed, including the RUP (McLeod and Roeleveld, 2002, p. 4), but with the exception of SSADM, make a base assumption that an information system is the appropriate type of solution and do not attempt to explore alternative approaches. As a result, SSADM is probably the most thorough and thus effective methodology in terms of defining the prognosis outline.

Defining Problems

According to McLeod and Roeleveld (2002, p. 4), the process of identifying what problems are "holding the organisation back from achieving its 'desired situation' may be helpful in designing the solution". These problems are usually identified using some form of gap analysis (McLeod and Roeleveld, 2002, p. 4), which the RUP is "largely incapable of ... because it does not collect any information on process effectiveness or on resource inefficiencies". The same can be said of XP, FDD, OOAD and the MSF, as gap analysis-type activities are not offered by these methodologies. The only methodology that does offer sufficient insight into current activities, and their problems, is SSADM, which includes investigations of current processing (process step 130) and data (step 140) which result in a derived logical view of current services (150). Accordingly, SSADM is the most effective of the methodologies analysed in terms of defining problems with the current situation.

Deriving Notional Systems

Jayaratna (1994, p. 88) states that notional systems are "those systems which need to be developed if the client organisation is to overcome the previously defined 'problems', thereby helping to transform the 'current state' to the 'desired state'". The RUP, according to McLeod and Roeleveld (2002, p. 4), states that the process of devising systems that attempt to satisfy the agreed requirements is catered for by the Analysis and Design activity during the Elaboration phase and that the RUP emphasises an easily updateable, robust design should requirements change. Commitment phase of the Planning Game allows the client to be involved in choosing the scope of the system, based on the User Stories (Beck, 1999, p. 27). However, this appears to be relatively limited, as the notational system is, in essence, a set of prioritised user requirements, given that XP's focus on design is intended to be as simple as possible, easily changed or refactored over short iterations (Beck, 1999, p. 27).

FDD's notational systems are essentially the same as XP's, in that they are prioritised features in a feature list (Palmer and Felsing, 2002, p. 57) but cover analysis and design in more detail than XP, which involves very little, or no, analysis. The MSF, on the other hand, attempts to provide a number of solution concepts after analysis has been performed during the Envisioning phase (Microsoft, 2003, p. 127). This is in the same vein as SSADM, which aims to provide business and technical solution options to the client, as part of stages 2 and 4 (Goodland and Slater, 1995, pp.10, 11, 12). Secondly, in stage 3, SSADM (Goodland and Slater, 1995, p. 11) provides a detailed specification of the required system, where there is an attempt to take a fresh view of requirements, without taking into account any current systems. OOAD uses prototyping, storyboarded scenarios and object diagrams to describe possible options, which are sometimes assembled into Requirements Analysis Document (Booch, 1994, pp. 253, 256). Again, SSADM appears to be the most thorough and effective of the methodologies analysed at deriving notional systems, with XP and FDD being the least thorough, if they can even be considered to attempt to derive specific notional systems at all.

Performing the Logical/Conceptual and Physical Design

Beck (1999, pp.6, 8) states that the design goal of XP is that of simplicity, or a simple design that is easily changed over short iterations, with no need to perform design beyond the current iteration. The basic tenet of XP design is "the simplest thing that could possibly work" (Beck, 1999, p. 6), and is, as such, not divided between logical and physical design. SSADM, on the other hand, has large, separate Logical Design and Physical Design stages, comprising 140 pages in Goodland and Slater (1995). In terms of logical design, the RUP, according to McLeod and Roeleveld (2002, p. 5) recommends splitting the software system into subsystems and grouping classes into *packages,* designing interfaces (using Component diagrams) and modeling classes collaboration in realising Use Cases (using Collaboration Diagrams).

The RUP's physical design is predicated on using Component and Deployment diagrams to represent linking with source files, binaries, and executables and, as McLeod and Roeleveld (2002, p. 5) state, it does this "clearly". McLeod and Roeleveld (2002, p. 5) note that the RUP, and thus FDD and the MSF, because of their use of UML, have a good foundation for rigorous architectural analysis because the methodology user creates models from many different perspectives.

The MSF applies an almost iterative approach to conceptual, logical and physical design, and separates logical and conceptual design into separate activities (Microsoft, 2003, p. 128). The Functional System Specification (Microsoft, 2003, pp. 134, 135) is concerned with representing much of the design (conceptual, logical and physical) and comprises artefacts such as task and task-sequence models, logical object and service models, conceptual models of the solution, a logical database model and a system architecture for conceptual and logical design, all of which are represented in UML and ORM notation. In terms of the MSF's physical design activities, the Functional System Specification (Microsoft, 2003, p. 135) contains physical design artefacts such as component packaging and distribution topology, infrastructure architecture and design, a physical database model and class, sequence, component and deployment diagrams. This is followed by a Technical Specification which contains an architecture overview, object model, interfaces, code flow, error codes, error logging, configuration information, any supporting documents, and a list of documented issues (Microsoft, 2003, pp. 381, 382).

In terms of FDD, an overall logical model is created very early in the methodology process as part of the *Define an Overall Model* phase and refined throughout the life of the project (Palmer and Felsing, 2002, pp.57, 69-70), while physical design is performed as part of the *Design By Feature* phase where detailed sequence diagrams and class and method prologues are created, prior to implementation. Finally, OOAD's Design phase (which is part of the OOAD *Macro* process) includes activities to describe the system architecture and tactical policies (Booch, 1994, pp.254, 255). The analysis phase includes creating a data dictionary with all identified classes and objects (Booch, 1994, p. 255). In terms of physical design, the *Micro* process specifies the identification of classes and objects and their semantics and relationships, which comprises a detailed physical design for the system (Booch, 1994, p. 235). The MSF, as it uses UML and approaches design from three, as

opposed to two, perspectives, in an iterative fashion is probably the most effective of the methodologies, followed by SSADM, which specifies very detailed design processes, and the RUP, through its use of UML and detailed design processes.

Implementing the Design

In terms of implementation, XP uses an iterative model, with the Onsite Customer providing input and feedback (Beck, 1999, pp. 6, 52). XP also relies on continuous integration with an attempt to integrate unit and user-based testing as much as possible (Beck, 1999, p. 6). This is paradigmatically similar to FDD, in which the Build by Feature phase involves the implementation of necessary classes, unit testing, integration and code inspection on an iterative basis (Palmer and Felsing, 2002, p. 58). The RUP links the UML packages, or groups of classes, to implementation subsystems and undertakes construction in an iterative manner, using unit and component testing, and only considers the system complete when all client requirements have been fulfilled (McLeod and Roeleveld, 2002, p. 6).

The MSF also takes an iterative approach to implementation, albeit using a combination of the iterative and waterfall approaches, employing the Microsoft *.net* architecture along with both coverage testing (including unit testing, regression testing, configuration tests) by the project team and usage testing by users (Microsoft, 2003, pp. 4, 398, 399). The final phase of OOAD's Micro process (Booch, 1994, pp.246, 247) is involved with the implementation of the classes and objects identified in the design phase, through iterative and incremental "spins" of the Micro process, which ensures that physical design and implementation are very closely linked.

This is as a result of the phases of the Micro process being part of the Evolution Macro process phase, which itself thus contains both physical design and implementation tasks (Booch, 1994, p. 258). SSADM has no implementation focus, as it does not address the construction phase of the SDLC (Goodland and Slater, 1995, p. 7). Consequently, SSADM is clearly the least effective of the methodologies analysed in terms of implementing the design. However, it would be difficult to select a most effective method in terms of implementing the design, as they all appear to cover the necessary ground, with the following exceptions:

- OOAD does not cover environment preparation, changeover and management and control, all of which are activities that Jayaratna (1994, pp. 105, 106) defines as necessary for the implementation phase.
- XP attempts to perform change management (or changeover) through providing small releases of the implemented system on a near-continual basis (Beck, 1999, p. 120) as well as using the Planning Game to cover Jayaratna's (1994, p. 105) strategy and planning activity, which is concerned with sequencing activities and setting completion dates.
- XP does not cover the environmental preparation or management and control activities specified by Jayaratna (1994, pp. 105, 106).
- The MSF does not cover the changeover or strategy and planning activities, but does cover environmental preparation (Microsoft, 2003, pp. 416, 417) and management and control

activities through artefacts such as the Project Structure document (Microsoft, 2003, pp. 104, 105).

- FDD handles the changeover activity, albeit in a high-level, general manner (Palmer and Felsing, 2002, pp. 250-253) and strategy and planning, as part of the *Plan By Feature* phase (Palmer and Felsing, 2002, p. 145), but does not cover the environment preparation or management and control activities.
- The RUP does not approach change management in any detail, but does cover the management and control (IBM, 2004:*Workflow Detail: Monitor & Control Project*), strategy and planning activities (IBM, 2004:*Guidelines: Iteration Plan*) and environment preparation (IBM, 2004:Artefact*: Deployment Model*). Accordingly, the RUP may be the most effective methodology in terms of implementing the design.

Conclusions

While no absolute ranking can be given to the analysed methodologies (as the method of analysis does not lend itself to this type of result), it can be seen from the individual element results that some methodologies are substantially better than others in some criteria, and significantly limited in others. For example, XP, FDD and OOAD are limited in their inability to take into account problem context-related aspects, while SSADM is limited, especially in terms of this type of analysis, by its lack of focus on the implementation aspects of systems. Perhaps one aspect that is worth taking into account is how the different methodologies handle design activities. The design activities specified by SSADM, RUP and MSF are of a high level of detail, while the methodologies based on the agile paradigm, XP and FDD, use much simpler, lighter design activities. While this research is not concerned with evaluating, or being able to evaluate, whether more or less detail is universally more effective in terms of design activities, it is suggested that this is dependent on the individuals making up the project team.

This measure used for this criteria is of debatable validity for a number of reasons, specifically:

- It does not provide a way of ranking methodologies in terms of their effectiveness.
- It requires a high level of understanding of the NIMSAD framework, which is time-consuming and relatively complex to gain a good understanding of.
- It requires a high degree of effort, and time-investment, to perform the analysis.

Other possible measures for methodology effectiveness would require empirical data, experiment or test cases which, while valuable and possibly more valid than this measure, are even more time-consuming and fraught with unforeseeable issues. As such, given the type of analysis required by this framework, methodology effectiveness could be considered an optional criteria, based on whether the user of the framework is willing to provide subjective rankings based on the NIMSAD elements.

Validity

A number of different approaches to measuring validity have been suggested. Caddell and Linssen (1985, p. 203) believe that the best approach to measuring the validity of a methodology is to use and observe the methodology in a slightly different situation to the one it was intended for. In their research, they determined the validity of a methodology concerned primarily with the "creation of original software", or focused on the "pure development perspective", by using it to port System V Unix to the Motorola 68 processor. El Emam *et al.* (1993, p. 269) believe that methodology validity should be measured using two criteria, namely content validity and construct validity, based on quantitative analysis. Content validity is analogous to completeness, and refers to the degree to which hypotheses capture all significant relationships to achieve the purpose, while construct validity is analogous to consistency and can be measured using factor analysis techniques (El Emam *et al.*, 1993, pp. 279, 281).

Van Belle (2003, p. 212) suggests that content and construct validity are best assessed in terms of statistical and experimental measures. In addition, he recommends the use of face validity as a criterion for validity, as this index can be measured by acceptance in the field and this can influence, or be influenced by, the perceived authority of the author(s) of the methodology. Accordingly, Van Belle (2003, p. 213) recommends the following as potential measures of validity:

- The number of lead author citations, based on a citation search engine such as the Citeseer search engine (http://citeseer.ist.psu.edu) can be used to determine the degree of authority of the methodology reference material author(s). The results of this investigation are shown in the table below, as Authoritative Validity.
- An informed estimate of methodology use.
- Book sales rankings (specifically Amazon.com), which reflect methodology purchase, as the majority of the methodologies evaluated are available in book format.

Authoritative Validity

The results of the investigation into methodology author validity are discussed below (*Table 30*):

Methodology	*Lead Author*	*Citations*	*Ranking*
FDD	Stephen. R. Palmer, John M. Felsing	1	7
MSF	Microsoft	**> 10000**	**1**
OOAD	Grady Booch	**862**	**2**
RUP	IBM	**> 10000**	**1**
SSADM	M Goodland	22	4
XP	Kent Beck	331	3
	Ron Jeffries	19	5
	William (C.) Wake	7	6

Table 30: Authoritative Validity Results

In terms of authoritative validity, IBM and Microsoft, being large commercial entities, have substantially more citations than any of the other authors. The majority of the other authors, with the possible exceptions of Grady Booch (Object-Oriented methodology) and Kent Beck (Extreme Programming), would appear to have quite a low level of authoritative validity. Both Wake (Extreme Programming Explored) and Palmer and Felsing (A Practical Guide to Feature-Driven Development) have less that 10 citations, and Goodland (SSADM Version 4) and Jeffries have slightly more. It is not unexpected that Booch and Beck are the highest-ranked of the named authors, being consistent with their reputations as the main figures and most influential advocates within the OO and XP methodologies (Capretz, 2003, p. 12; Grossman *et al.*, 2004, p. 245).

Estimated Usage

In a survey of agile methodology adoption, Ambler (2006[2], p. 2) surveyed over 4200 software development professionals with the following results of interest:

- 1886, or 46% of respondents (163 did not respond) believed that they had average, extensive or very extensive knowledge of agile methodologies.
- 23.4% of respondents stated that their organisation had adopted XP.
- 12.3% of respondents stated that their organisation had adopted FDD.
- 4.7% of respondents stated that their organisation had adopted the MSF, or at least the agile portion thereof.
- 59% of respondents stated that their organisation had not adopted an agile methodology.

The results of this survey indicate that XP is the most popular of the agile methodologies. One problem with this criterion is that comparable usage data for all methodologies surveyed should be available and, in this case, no usage data was available for the RUP, OOAD and SSADM. In Ambler's survey (2006[2]) the adoption of agile methodologies is not necessarily exclusive, as respondents were able to indicate that their organisations had adopted more than one agile methodology. Very few, or no, estimates were available for the other methodologies analysed, however, a number of points can be taken into account. In terms of SSADM, Fitzgerald (1998, pp. 321, 322) determined that, in a survey of 162 organisations on methodology usage, SSADM was the most popular methodology represented in the study, more popular than Information Engineering and Oracle*Case and Andersen Consulting's Foundation/Method 1. Secondly, Fitzgerald (1998, pp. 322, 323) found that, based on commercial methodologies, SSADM was the most popular basis for methodologies developed internally.

IBM (2004, p. 2) quotes one of their customers in stating that one of their reasons for selecting the RUP was that the RUP was a standard used by many of their "customers and business partners all over the world" and that "everyone knows RUP and its terminology". While very little other information is available regarding the usage of the RUP, it would appear that it enjoys significant usage, as open-source competitors such as OpenUP have been based on it. In terms of OOAD, it appears that while it may still be used, there is very little evidence for its use as a complete methodology.

OOAD is, however, being taught in many university-level courses on computer science and software engineering, and there are substantial amounts of literature relating to teaching it (i.e. Kolling, Koch and Rosenberg, 1995; Kolling, and Rosenberg, 1996; Kolling and Rosenberg, 2001; Zhou and Zhu, 2003). OOAD was however, according to Alhir (1998, p. 11), one of the predecessors of the RUP, as it was incorporated into Objectory, which later became the RUP. From this it could be inferred that, with the RUP's usage increasing, the usage of OOAD has decreased.

Book Sales Rankings

To extend Van Belle's (2003, p. 213) suggestion of using Amazon.com book sales rankings, both BarnesAndNoble.com and Amazon.co.uk book sales rankings have been added. Amazon.co.uk books sales rankings were introduced in order to investigate whether there was a 'cultural divide', specifically in terms of SSADM (SSADM is of British origin, while the other methodologies analysed are of American origin). The results are shown in the table below, and can be explained as follows:

- The Relative Ranking is a number (1 = highest, 5 = lowest) assigned to each reference material based on their SalesRank (it is important to note that SalesRank figures are provided in a descending order thus, for example, Amazon.com's highest selling book would have a SalesRank of 1).
- Only a single XP reference was used, namely Extreme Programming Explained 2nd Edition (Beck, 2004).
- A cumulative relative ranking (***Book Sales Ranking***) value was assigned to each reference material analysed, being the average of their combined relative rankings.

In terms of books sales rankings (see *Table 31* for full results), Beck's Extreme Programming Explained (2nd Edition, 2004) is ranked highest in terms of all online booksellers SalesRank figures and thus has the lowest CRR. It was followed by the MSF reference material and Booch's Object-Oriented Analysis and Design, which was, at least in one instance, the second-lowest relative ranking. A very interesting point to note is that Goodland and Slater's SSADM Version 4 is by far the lowest ranked (as the only book with a SalesRank of over a million) by Amazon.com and is, in fact, not even stocked by BarnesandNoble.com. However it is ranked third in terms of book sales rankings for the reference materials analysed by Amazon.co.uk, and ranked higher than the second-highest ranked reference by Amazon.com.

This may allow a conclusion to be drawn that SSADM is considered more valid, at least in terms of face validity, in the United Kingdom where it originated as an initiative of the British Government. It thus appears to be considered less valid or is more possibly less known in the United States of America. As the Rational Unified Process was the only methodology not made available in book format, it could not be ranked against the other methodologies.

Book Seller	FDD	MSF	OOAD	SSADM	XP
Amazon.com	314044	46384	176439	2300305	***24794***
Amazon.co.uk	256730	162223	151409	158314	**2775**
BarnesAndNoble.com	162784	156141	178446	Not Stocked	**57398**
Book Sales Ranking	4	2	3	5	***1***

Table 31: Book Sales Rankings

Please note that the references used for the basis of this analysis are those described in *Table 7*.

Conclusion

These measures appear to be a fairly good representation of methodology validity, with one caveat – measuring methodology usage is very difficult and the best results one can hope for is, in many instances, an informed estimate drawn from a number of unrelated sources. Methodology usage and book sales are continually changing, but the advantage of using book sales as a measure is that is easily available, which is not the case for methodology usage.

Availability and Cost

Kelley and Rogers (1992, p. 177) states that cost should comprise one of the main criteria for methodology evaluation and that the cost criteria should include both the cost of purchasing the methodology reference material and, if necessary, the associated methodology software, as well as the cost of using the methodology and associated training, infrastructure and changing business processes as a result of introducing a (new) methodology and supporting the methodology. As many of these factors are largely dependent on the organisation into which the methodology is being introduced, they will not be assessed in this criterion which will instead focus directly on the aspects of the methodology that must be purchased in order to use the methodology.

In most cases (7 out of 8 methodologies), the bare minimum required to use the methodology is the reference material in book format. Accordingly, neither cost nor availability appears to be a huge issue because the cost for this reference material varies between around $30 and $70 dollars, and all (bar the RUP) are available on the major internet bookselling sites. The only methodology available only in electronic format, the Rational Unified Process, is made available freely as a trial download, but the Rational Method Composer, or non-free aspect, is priced at $395 per seat, substantially more than any of the methodologies available in book format. The results for this criterion are shown in the following table (*Table 32*). In the case of this criterion, it seems that methodology acquisition, at least in terms of the methodologies analysed in this research is a relatively small effort and cost compared to the other aspects of methodology implementation such as training and change management. As such, the availability and cost criterion might not differentiate significantly between methodologies. However, more expensive commercial methodologies do exist, many of which, such as those associated with consulting firms, are coupled with consulting fees, and cost may thus become an important selection criterion when faced with the possibility of purchasing one of these methodologies.

Conclusion

The measures for this criteria, and the criteria itself, are largely descriptive, but it could be argued that they provide a useful addition to the decision making tools offered by this research. As with the measures for the previous criteria, it is important to take into account that methodology availability and cost are continually changing, but that information regarding this is easily available.

Methodology	Reference	Availability (as of 15 October 2006)	Size (Physical)	Costs	Process Model Capture time (days)	Available in (legal) digital format?
FDD	A Practical Guide to Feature-Driven Development (Palmer and Felsing)	Book	304 pages	$33.59	2	No
MSF	Analyzing Requirements and Defining Microsoft .NET Solution Architectures (Microsoft)	Book and CD-Rom	495 pages	$44.09	2	Yes
OOAD	Object-Oriented Analysis and Design with Applications (2nd Ed) (Booch)	Book	608 pages	$59.49	4	No
RUP	Rational Unified Process Trial Edition/Rational Method Composer	Electronic	12mb zip file	Free; $395/ seat licence	3	Yes
SSADM	SSADM Version 4 (Goodland and Slater)	Book	534 pages	$77.03	3	No
XP	Extreme Programming Explained (Beck)	Book	256 pages	$37.99	3.5	No
	Extreme Programming Installed (Jeffries)	Book	288 pages	$34.99	-	No
	Extreme Programming Explored (Wake)	Book	144 pages	$29.99	-	No

Table 32: Methodology Availability and Cost Results

Currency and Maturity

Kitchenham (1996, p. 13) states that the maturity of a methodology indicates the extent to which there is likely to be information about it readily available and believes that this aspect can be assessed in terms of three categories:

1. Not in use on commercial projects (or still being developed).
2. Used in a few state-of-the-art products or projects.

3. In widespread use.

Kelley and Rogers (1992, p. 177) state that the maturity of a methodology is predicated on its experience base, trained personnel, progress over time, and stability and believes that a methodology that "has a trained user base provides both available, knowledgeable personnel and a known success rate". A further argument that methodology maturity is beneficial is that, according to Kelley and Rogers (1992, p. 177), if a methodology has matured over time without significant disruptions, "it is more likely to be well engineered than one that has undergone significant modification to accommodate changes". Finally, Kelley and Rogers (1992, p. 177) define a mature methodology as a methodology that has "accommodated change yet remained stable over a period of time".

Accordingly, a methodology maturity and currency scale can be defined based on the following criteria:

- Date of methodology introduction.
- Last update to methodology reference material.
- Frequency of updates.
- Methodology version, if a standard versioning system or notation is used.
- Estimated methodology user base (using a truncated scale of small, medium and large). For more information on the derivation of this value see the Validity section above. Note, however that:
  - A large estimated user base is one where anecdotal evidence leads one to believe that the methodology is well known and used.
  - A medium-sized estimated user base is one where anecdotal evidence leads one to believe that the methodology is relatively well known and enjoys usage beyond what can be considered niche usage.
  - A small estimated user base is one where anecdotal evidence leads one to believe that the methodology is largely unknown and/or enjoys mainly niche usage.

The following scale of methodology maturity can be adapted from Van Belle (2003, p. 232) and suggests the classification of methodologies, subjectively, as follows:

- *Fossils*, or methodologies not updated in the last five years.
- *Rising Stars*, or methodologies less than five years old but with an already large or quickly growing user base. Rising stars may often be available as a commercial product.
- *Immature*, or methodologies less than five years old with a small but possibly growing user base.
- *Mature*, or methodologies between five and ten years old, regularly updated, with an established estimated user base of small or medium.

- *Stars*, or mature methodologies that are regularly or very regularly updated, have a large user base, are older than five years old and are often available as a commercial product.
- *Waning Stars*, or mature methodologies that are more than five years old, have a large user base, are often available as a commercial product but are not updated regularly or at all.

The analysis results, and subjective classifications, can be presented as follows (*Table 33*):

	Year Introduced	Last update	Update Frequency	Version	Estimated User Base	Tentative Classification
XP	1999	2006	Very regular	n/a	Large	*Star*
FDD	1997	2002	Unknown, presumed low	1.3	Medium	*Mature*
RUP	1998	2005	Regular	1.2	Large	*Star*
SSADM	1981	1995	Not updated	4+	Large	*Waning star*
MSF	1994	2006	Regular	3	Small	*Mature*
OOAD	1992	1994	Not updated	n/a	Small	*Fossil*

Table 33: Methodology Currency/Maturity Results

The analysis results describe a good spread of methodology maturity, from *Fossil* to *Star.* As expected, XP and the RUP are *Stars*, given their large estimated user bases and regular updates. FDD can be classified *Mature*, but not a *Star*, as it has been updated within the last five years and has an estimated medium-sized user base. According to Ambler's survey of agile methodology adoption and usage (2006[2]), FDD had a user base rate of around 12% of the 4300 individuals surveyed organisations; or the second highest after XP (Ambler, 2006[2], p. 2). The MSF can also be considered *Mature*, as it is one of the older methodologies, and regularly updated, but has a small user base. SSADM can, despite its large user base, be classified as a *Waning Star*, because of its lack of updates and age (it is the oldest of the methodologies by 11 years, followed by OOAD which is classified a *Fossil* for reasons explained later). SSADM is, however, the most mature of the methodologies analysed as a result of its long lifetime. During this time (14 years) it underwent numerous updates, moving from Version 1 to Version 4+.

The MSF could also be classified as *Mature* as it has managed to capture a small user base (around 5%) since its introduction (Ambler, 2006[2], p. 2). This can be compared to XP's user base of around 23% and FDD's user base of around 12%, based on Ambler's survey into agile methodology usage and adoption (2006[2]). Secondly, as the MSF is a commercial product closely linked to the Microsoft *.net* platform, its usage can be expected to grow if the .net platform's usage grows. OOAD, given its age, lack of updates, and small user base (which cannot be confused with users of the UML, or users of the Object-Oriented paradigm which are both very widely used) can probably be considered a *Fossil*.

Support

Another criterion closely linked to methodology usage and currency/maturity is methodology support. In fact, Kitchenham's definition of maturity (1996, p. 13) which defines maturity as the extent to which there is likely to be information about the methodology readily available can, in many ways, be as applicable to support as it is to maturity.

In these terms, the following aspects of support have been investigated:

- *Tool support*, the tools made available to support the use (and customisation) of the methodology.
- *Vendor support*, the support made available by the vendor to aid in the use of the methodology. This support is often concerned with training prospective users of the methodology, or providing consulting services around the implementation and use of the methodology at an organisation.
- *Additional support*, third-party or community-based support for the use of the methodology. This aspect of support relates largely to the informal, easily-accessed support for the use of the methodology through methods such as online forums, user groups, and conferences, and are not necessarily vendor sponsored, supported or authorised.

The results are shown in *Table 34*. From the results, it is obvious that supporting tools are available for all of the methodologies analysed, ranging from open source tools intended to assist in the use of aspects of a methodology, such as Feature Trackers for FDD, to proprietary methodology tailoring tools such as the Rational Workbench (which is available from IBM, the suppliers of the RUP, for USD395 per seat) and Microsoft's Visual Studio Team System (available from Microsoft, supplier of the MSF, at USD10,939 per implementation). One particularly interesting point to note is the number of third-party tools available to support the use of SSADM. This may be due to the fact that while the *de facto* owner of SSADM is the United Kingdom government, SSADM was made an open standard which allowed third party vendors to create supporting tools, such as SelectSSADM (http://www.selectbs.com/products/select-ssadm.htm), and to sell them (Maybank, 1999).

Another interesting point to note is the open source and free software support for the methodologies based on the agile paradigm. There are numerous tools available for free download on websites such as xprogramming.com (which is administrated by Ron Jeffries, one of the authors of Extreme Programming Installed) and featuredrivendevelopment.com, which often has the input of Jeff de Luca, one of the creators of FDD (Palmer and Felsing, 2002, p. xxiii). De Luca's company, Nebulon, also publishes updates to FDD and offers accredited training in it (Nebulon, 2002).

Methodology	Tool support	Vendor support	Additional Support
FDD	Open source, free and proprietary support tools available	Training and consulting made available by vendor and third parties.	At least one online forum exists. Updated documentation and processes freely available at Jeff de Luca's site: http://www.nebulon.com
MSF	Proprietary tools available (such as Microsoft Visual Studio)	Training and consulting made available by vendor.	Vendor-supported forum and FAQs at http://msdn.microsoft.com
OOAD	Proprietary, free and open source tools available	Training made available through third parties	Some forums and blogs through third party websites such as http://www.ooad.org
RUP	Proprietary tools available (such as Rational Workbench)	Training and consulting made available by vendor and third parties.	User groups, technical resources and online forums made available to customers by vendor at http://www.ibm.com
SSADM	Proprietary, third-party tools available (such as SelectSSADM)	Training and consulting made available through third parties	Unknown
XP	Open source, free and proprietary support tools available	Training and consulting made available through third parties	Many user groups, online forums and conferences at http://www.extremeprogramming.com, http://www.xprogramming.com

Table 34: Methodology Support Results

In conclusion, this chapter has described the final aspect of the framework proposed for the analysis and evaluation of software development methodologies, the pragmatic aspect. This chapter has provided the following:

- A brief introduction to the criteria that make up the pragmatic aspect of the proposed framework.
- A discussion of the measures used for each criterion, and analysis and evaluation results for the methodologies pertaining to each of the criteria that make up the pragmatic aspect of the proposed framework.

The final chapter will present conclusions regarding the research and will propose ideas for future research.

Chapter 8: Conclusions

This section concludes the research, presenting some aspects around the overall ranking of the methodologies analysed and evaluated, as well as some conclusions about the suggested framework and the use thereof. Finally, some ideas for future research will also be presented.

Comparative Methodology Evaluation

Until this point, the present research has not attempted to rate the methodologies analysed in a relativistic manner for the most applicable criteria. However, when considering the compilation of a holistic ranking, there is other information that should be kept in mind. Despite each methodology having had hundreds, and probably thousands, of man-hours invested in creating and refining them, they are only as valuable to an organisation attempting to implement it as the team implementing it and will be constrained by other aspects, many of which have been discussed in this research.

Some aspects that have not been ranked or looked at from a relativistic viewpoint in this research include *Paradigm*, *Effectiveness* and *Situationality*, and these criteria are obviously of great importance when selecting a methodology for use. The circumstances that can lead to one methodology being a tremendous success on a project may very well be the same circumstances that doom another to failure. Brooks (1987) has famously, and probably accurately, asserted that no methodology can be a silver bullet. This is true for all of the methodologies analysed and evaluated in this research, and should probably hold true for most methodologies.

It is also necessary at this point to take a step away from the relativist viewpoint and argue that in many ways all software development methodologies are comparable in aspects of their basis and their effectiveness. They should all relate to the development of software, they should all promise 'a better way of doing things' and they should all have evolved from experience. As such, no final, cumulative rankings of the methodologies evaluated have been included in this research. While this would be a trivial exercise to do, the sections discussing the limitations of the framework and the revised framework should provide insight into the decision to leave out a cumulative ranking.

Conclusions on the Proposed Framework, and the Use Thereof

It is important to note that, while this research has focused on the reference material made available for these methodologies, there is a substantial amount of knowledge and empirical information about the application of the methodologies in the public and, especially, the private domains.

Summary: Formal Criteria

The following formal criteria were proposed to analyse and measure the structural aspects of the methodology process models, which were derived by mapping the methodology processes, as described in the methodology reference material, against a methodology process metamodel:

- *Process Model Size* is measured against the number of elements in the methodology process.

- *Complexity* is measured using aspects such as fan-out and element coupling between the methodology process model elements.
- *Modularity and Structuredness* uses inheritance tree measures on methodology process models to determine their modularity and degree of structure.
- *Life Cycle Coverage/Granularity* measures the number of activities, process roles and artefacts per phase.
- *Formal Process Communication* uses the roles defined in the methodology process models for each methodology to identify and count the roles, both inter-project team and extra-project team, and the communication relationships between them.

Summary: Intrinsic Criteria

The following intrinsic criteria were proposed to analyse and measure the methodologies as described in the methodology reference material:

- *Paradigm, Process and Notation* are described in order to provide insight into the basis of the methodology.
- *Completeness* refers to the methodology's scope of coverage of the traditional software development life cycle and Zachman's enterprise architecture framework (Zachman, 1987).
- *Best Practice Coverage* refers to how the best practices suggested by the methodologies analysed compare to a set of best practices derived from industry research.
- *Support for Requirements Change* measures how changes to requirements and design are identified and actioned during the methodology process.
- *Verification and Validation Techniques* relate to the tools and techniques made available for verification and validation of their artefacts.
- *Management Support* measures the level to which project management activities are supported.
- *Documentation* measures quality and understandability of the methodology reference materials.

Summary: Pragmatic Criteria

The following pragmatic criteria were proposed to analyse and measure aspects of the methodologies which cannot be assessed purely on the information contained within the reference material:

- *Situationality* is concerned with how applicable a methodology is to various types of organisations, problems and types and sizes of organisations and projects.
- *Flexibility/Expandability/Enhanceability* is concerned with measuring the extent to which a methodology allows change and adaptations.
- *Effectiveness* is concerned with a methodology's ability to increase productivity and manage

life cycle phases, activities, process roles and products, and is measured using Jayaratna's NIMSAD framework (1994).

- *Validity* is measured using face validity (which includes estimated methodology usage) and authoritative validity.
- *Availability/Price* are concerned with the medium and state of availability of the methodology and how it is made available.
- *Currency/Maturity* relate to how well-updated the methodology is and how often changes and updates are made.
- *Support* relates to product or tool support, vendor support and user base.

Conclusions on the use of Van Belle's Framework (2003)

The first conclusion to be drawn in the present study is that the use of Van Belle's framework (2003) as a structuring method is largely effective for methodology comparison, based on the following observations:

- The field of method engineering, especially relating to the evaluation of methodologies, is appropriate for and applicable to the field of enterprise models. This is due to the "valuable criteria (often including various metamodeling measures) that can be applied to the evaluation of enterprise models" (Van Belle, 2003, p. 54). These criteria have influenced and informed the development of Van Belle's framework, which in turn informed the development of the framework proposed in this research, thereby constituting a circular aspect to the research.
- The framework proposed by Van Belle and the framework proposed by the present research 'tread very similar ground' in that they reference many of the same tools and frameworks, an example being Zachman's enterprise architecture framework (1987), which Van Belle believed could be a useful framework in the context of research in this area (2003, p. 239). The Zachman framework was not used for the evaluation of enterprise models as it was considered "more useful as a reference when constructing a model rather than measuring the completeness of the model ex post" (Van Belle, 2003, p. 239). However it can be used as an evaluation tool for measuring methodology completeness.

The second conclusion drawn is that methodologies appear to have a degree of isomorphism with enterprise models, albeit not a particularly high one. Van Belle asserts that his framework could be "applied, *mutatis mutandis*, with almost equal success" to other research areas where there is a high level of isomorphism with enterprise models (2003, p. 260). This is obviously not the case with methodologies, as both different criteria have been suggested to make up the framework and, when similar criteria have been used, different measures have been used for the criteria.

A final conclusion is that the framework is more suitable for relative/comparative evaluation, and the removal of Absolute/Relative axes was beneficial to the framework. However, some criteria, such as *Paradigm*, *Effectiveness* and *Situationality,* were evaluated from an absolute perspective in this

research, but given a specific context for a methodology, a relative approach could be used. As a side note, the investigation into relative measures for absolute criteria such as *Effectiveness*, especially in the context of this research, may provide the basis of valuable and rewarding research.

Limitations of the Proposed Framework

In terms of the findings in this research and the use of the proposed framework, a number of limitations were identified.

- In the cases of some criteria, the measures used do not take into account the format of the reference material evaluated. For example, physical methodology size cannot be evaluated where the methodology reference materials are of different formats.
- Adequate measures for some criteria, such as *Modularity* and *Validity*, could not be found in the reference material, prompting the use of measures in an isomorphic setting. These measures were not always effective.
- Many of the measures require explicit statement of supporting aspects. An example of this is the criteria relating to *Support for Best Practices* which, for evaluation purposes, requires the methodology reference material to explicitly state sets of best practices.
- Differences in documentation type may affect criteria such as *Documentation Quality*. For example, methodology reference manuals may be written in a more formal style than methodology textbooks resulting in a lower Flesch Reading Ease Scale value. While this was not found to be the case in this research, it is entirely conceivable that it may be in other scenarios.
- The framework does not have a substantial quantitative aspect. Accordingly, it could be argued that the framework is limited in that it cannot properly quantify aspects such as methodology cost, *Situationality* and *Effectiveness*, criteria which could be evaluated through quantitative measures.
- As not all criteria were considered meaningful, nor all measures adequate, an overall ranking of the methodologies evaluated was not included. Another factor that contributes to this limitation is that not all criteria, and the measures thereof, allow the ranking of methodologies (such as *Paradigm, Process and Notation).*

These limitations are discussed in more detail in the following section.

A Revised Framework

Based on the limitations described above, the proposed framework and some of the measures used can be refined in a number of ways. In terms of the Formal criteria, the following points should be noted:

- *Size* is a meaningful criteria and the metrics proposed to measure this are suitable. However, there is no adequate way to compare physical methodology size where the methodology reference materials are of different formats. Where this is the case, and given the proposed

relationship between methodology physical size and methodology model size, this measure should be considered optional.

- While relative Formal measures of *Complexity* are adequate, this should also be measured from an Intrinsic perspective, potentially focusing on aspects such as concept complexity and ease of learning.
- The measure proposed for *Modularity* is not adequate and, based on the lack of adequate measures in the literature surveyed, this criteria has been removed from the final framework.
- *Life Cycle Coverage/Granularity* and *Formal Process Communication* are meaningful criteria and the metrics used appear to be an effective measure of these criteria.

The following conclusions can be made about the Intrinsic criteria:

- The criteria covering *Paradigm, Process and Notation* is not intended to be used as a measure but should continue to be included from a descriptive perspective.
- Despite being a meaningful criteria, it could be argued that, as the measures proposed for *Completeness* cover only two aspects, specifically artifact completeness (through coverage of the Zachman Framework for Information Systems Architecture) and coverage of the SDLC, these measures are not effective when taking agile methodologies, which value "working software over comprehensive documentation" (Boehm, 2006, p. 19) and thus may limit documentation efforts, into account.
- While a valuable criteria, *Best Practice Coverage* should be regarded as an optional criteria as it is only useful when comparing methodologies that explicitly suggest sets of best practices. In the case of this research, only half of the methodologies evaluated did this, thus meaning that the comparison was limited and incomplete.
- Both *Support for Requirements Change* and *Verification and Validation Techniques* are meaningful criteria, but the measures used are potentially not satisfactory in that they do not focus on the detail of the criteria, merely the presence or absence thereof. Accordingly, more qualitative measures may provide better insight.
- *Project Management Support* is a valuable criteria and while the measures suggested are satisfactory, it could be argued that additional measures, such as methodology dependence on external project management methodologies and the inclusion of progress tracking techniques could be included for completeness.
- *Documentation Quality* is a useful criteria, and while most of the measures used are satisfactory, the Flesch-Kincaid Grade Level measure may not be of sufficient value to justify its use as a metric.

Finally, a number of potential conclusions can be drawn about, and revisions made to, the Pragmatic criteria and measures:

- Measured as they were in this research, *Situationality* and *Effectiveness* should be included as Intrinsic criteria. However, should different, more qualitative metrics be used then these criteria would remain Pragmatic criteria. In fact, it could be argued that to gain a true picture of methodology support for these criteria, an approach based around an initial qualitative evaluation, followed by a quantitative evaluation be followed, which would allow both Intrinsic and Pragmatic aspects of the criteria to be evaluated. In addition, it could be suggested that the NIMSAD framework is too 'heavy' for use in terms of the *Effectiveness* criteria and that a more qualitative approach would allow better evaluation.
- *Flexibility/Expandability/Enhanceability* is a valuable criteria where the use of both Intrinsic and Pragmatic metrics provides a balanced and complete measure.
- Unless a more accurate measure for *Validity* can be found, it is recommended that this criteria is removed, as the measures used are not an accurate view of a criteria that may be considered subjective.
- While *Availability/Cost* provides an additional decision making tool for users of the framework, it does not attempt to quantify the true cost of methodology adoption, which is largely situational and based on the time and effort spent on adopting the methodology including aspects such as training and change management. In this case it is recommended that the *Cost* criteria be included within the *Support* criteria, where costs can be quantified against support types and providers, and the *Availability* criteria and metrics be incorporated within *Currency/Maturity* criteria, as reference material availability is more closely linked to currency than to cost.

In conclusion, the revised framework is as follows:

	Formal	***Intrinsic***	***Pragmatic***
Criteria (Mandatory)	Process Model Size and Complexity Life Cycle Coverage/ Granularity Process Roles and Inter-role Communication	Paradigm, Process and Notation Intrinsic Complexity Completeness Support for Requirements Change Verification and Validation Tools/Techniques Project Management Support Documentation quality Best Practice Coverage (Optional)	Situationality Flexibility/Expandability/ Enhanceability Effectiveness Support/Cost Availability/Currency/ Maturity

Table 35: Revised Framework

Heuristics for the Use of the Proposed Framework

As the framework described in this research was not proposed purely for the academic domain, it is hoped that it could provide a useful tool in the public domain. As such, an initial heuristic is that the

proposed framework should be seen as a decision-making tool, within a larger decision making process, as opposed to being the entire decision making process. Some of the more valuable criteria are difficult to use as decision-making tools unless the context in which the methodology will be used has been fully explored. As such, it is imperative to thoroughly understand the organisational context in which the methodology will be used. It is also important to fully understand the requirements for the methodology, since different projects and project contexts will favour different paradigms and methodologies. Finally, the cumulative results above are not weighted. Where necessary and appropriate, weightings should be assigned to criteria that are considered more important in terms of the organisational and project context in which the methodology will be used.

Ideas for Future Research

One aspect of methodologies that has not been explored in this research, and could form the basis of productive and relevant research, is related to the intangibles that make a methodology succeed in a specific project environment, but fail in only a marginally different one. It could be argued that organisational culture and specific attributes of the project team play crucial roles in this. Agile methodologies seem to thrive in environments that are unstructured, enthusiastic and full of experienced, but not jaded, technologists. More structured methodologies, on the other hand, may be more appropriate in environments containing intelligent, but not necessarily experienced, methodology users. It would appear that the combination of enthusiasm and experience can make up for the lack of structure associated with agile methodologies, while the combination of intelligence and ambition can make up for the possible frustrations at the inherent structure and delays in methodologies such as the RUP.

Another important aspect are the abilities of the project manager: motivational skills, likeability and a modicum of organisational skills may be 'all' that are required to make an excellent project manager. Indeed, an effective project manager can go a long way towards making up for shortcomings on the side of the project team or methodology employed. And while it may seem that this is not an extravagant mix of attributes, the high occurrence of failed projects raises questions about the personal qualities necessary for capable and successful project management. Maybe there is an "x-factor" that has not been identified. The role and functions of project management, specifically targeting this "x-factor" would make an interesting and highly relevant research study.

Chapter 9: References

Abrahamsson, P., Salo, O., Ronkainen, J., Warsta. J. (2002). *Agile Software Development Methods: Review and Analysis*. VTT Publication 478.

Abrahamsson, P., Warsta, J., Siponen, M. T. & Ronkainen, J. (2003). New Directions on Agile Methods: A Comparative Analysis. *Proceedings of the 25th International Conference on Software Engineering*. Portland, Oregon, USA. pp. 244-254.

Alhir, S. (1998). *UML in a Nutshell: A Desktop Quick Reference*. Sebastopol: O'Reilly Publishing.

Ambler, S. (2005). *The Enterprise Unified Process Best Practices*. Retrieved 19 October 2005 from http://www.enterpriseunifiedprocess.com/essays/bestPractices.html.

Ambler, S. (2006[1]). *History of the Unified Process.* [Online] Retrieved 20 December 2006 from http://www.enterpriseunifiedprocess.com/essays/history.html.

Ambler, S. (2006[2]). *Agile Adoption Rates Summary.* [Online] Retrieved 18 December 2006 from http://www.surveymonkey.com/DisplaySummary.asp?SID=1870875&Rnd=0.4730387.

Arni-Bloch, N. (2005). Towards a CAME Tool for Situational Method Engineering. *Proceedings of First Conference on Interoperability of Enterprise Software and Applications*. Geneva, Switzerland.

Atwood, M., Burns, B., Gairing. D., Girgensohn, A., Lee, A., Turner, T., Alteras-Webb, S. & Zimmermann, B. (1995). Facilitating Communication in Software Development. *Proceedings of the Conference on Designing Interactive Systems: Processes, Practices, Methods and Techniques*. Ann Arbor, Michigan, USA. pp. 65-73.

Avison, D., Fitzgerald, G. (1995). I*nformation Systems Development: Methodologies, Techniques and Tools, Second Edition*. London: McGraw-Hill.

Beck, K. (1999). *Extreme Programming Installed: Embrace Change*. Boston: Addison-Wesley.

Berki, E. & Georgiadou, E. (1998). A Comparison of Qualitative Frameworks for Information Systems Development Methodologies. *Proceedings of the Twelfth International Conference of the Israel Society for Quality.* Jerusalem, Israel.

Bjorn-Anderson, N. (1984). Challenge to Certainty. In *Beyond Productivity: Information Systems Development for Organisational Effectiveness*. Bemelmans, T. (ed). Amsterdam: Elsevier Science Publishers B. V., North-Holland Press. pp. 1-8.

Boehm, B. (2006). A View of 20th and 21st Century Software Engineering. *Proceedings of the 28th International Conference on Software Engineering*. Shanghai, China. pp. 12-29.

Boehm, B. & Turner, R. (2004). Balancing Agility and Discipline: Evaluating and Integrating Agile and Plan-Driven Methods. *Proceedings of the 26th International Conference on Software Engineering.* Edinburgh, Scotland. pp. 718-719.

Booch, G. (1994). *Object-Oriented Analysis and Design With Applications, Second Edition*. Redwood City: The Benjamin/Cummings Publishing Company.

Boggs, R. (2004). The SDLC and Six Sigma: An Essay on Which is Which and Why?. *Issues in Information Systems*, 5 (1), pp. 37-42.

Brooks, F. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. In *IEEE Computer,* 20 (4), pp. 10-19.

Bruegge, B., Dutoit, A. (1997). Communication Metrics for Software Development. *Proceedings of the 19th International Conference on Software Engineering*. Boston, Massachusetts, USA. pp. 271-281.

Burns, R. & Dennis, A. (1985). Selecting the Appropriate Application Development Methodology. In *Database,* 17(1), pp. 19-23.

Caddell, L. & Linssen, D. (1985). Application of a Software Development Methodology During the Port of Unix System V to the M68000™ Family of Microprocessors. *Proceedings of the 1985 ACM SIGSMALL Symposium on Small Systems*. pp. 203-205.

Capretz, L. (2003). A Brief History of the Object-Oriented Approach. *ACMSIGSOFT Software Engineering Notes*, 28(2), pp.6-16.

Castellani, X. (1998). Evaluation of Models Defined with Chart of Concepts: Application to the UML Model. *Proceedings of the Third CAISE/IFIP 8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design.* Pisa, Italy.

Catchpole, C. (1987). *Information Systems Design for the Community Health Services*. PhD Thesis, Aston University, Birmingham, United Kingdom.

Cockburn, A. (2001). *Agile Software Development.* Boston: Addison-Wesley.

Connors, D. (1992). Software Development Methodologies and Traditional and Modern Information Systems. *ACM SIGSOFT Software Engineering Notes*, 17(2), pp. 43-49.

Cortada, E. (1998). *Best Practices in Information Technology.* Saddle River: Prentice Hall.

De Champeaux, D., Lea, D. & Faure, P. (1993). *Object-Oriented System Development.* Boston: Addison-Wesley.

De Luca, J. (2002). *Feature-Driven Development.* Retrieved 22 December 2006 from: http://www.nebulon.com/fdd/index.html.

De Villiers, D. (2001). Using the Zachman Framework to Assess the Rational Unified Process. Retrieved 21 July 2006 from http://www.therationaledge.com/content/mar_01/t_zachman_dv.html.

Dufty, D., McNamara, D., Louwerse, M., Cai, Z. & Graesser, A. (2004). Automatic Evaluation of Aspects of Document Quality. *Proceedings of the 22nd Annual International Conference on Design of Communication: The engineering of Quality Documentation*, Memphis, Tennessee, USA. pp. 14-16.

El Emam, K., Moukheiber, N. & Madhavji, N. (1993). An Empirical Evaluation of the G/Q/M Method. *Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research: Software Engineering - Volume 1.* Toronto, Canada. pp. 265-289.

Enger, N. (1981). Classical and Structured Systems Life Cycle Phases and Documentation. *System Analysis and Design: A Foundation for the 1980s.* Cotterman, W.M., Couger, J.D., Enger, N. and Harold, F. [Eds] Amsterdam: Elsevier Science Publishers B. V., North-Holland Press. pp. 1-24.

Fitzgerald, B. (1996). Formalised Systems Development Methodologies: A Critical Perspective. *The Information Systems Journal,* 6(1), pp. 3-23.

Fitzgerald, B. (1997). The Use of Systems Development Methodologies in Practice: A Field Study. *The Information Systems Journal*, 7(3), pp. 201-212.

Fitzgerald, B. (1998). An Empirical Investigation into the Adoption of Systems Development Methodologies. *Information & Management*, 34, pp. 317-328.

Freeman, P., Wasserman, A. & Houghton, R. (1984). Comparing Software Development Methodologies for Ada: A Study Plan. *ACM SIGSOFT Software Engineering Notes,* 9(4),pp. 22-55.

Garland, J. & Anthony, R. (2003). *Large-Scale Software Architecture: A Practical Guide Using UML*. Chichester: John Wiley and Sons, Ltd.

Goodland, M. & Slater, C. (1995). *SSADM Version 4: A Practical Approach*. London: McGraw-Hill.

Gopal, A., Mukhopadhyay, T. & Krishnan, M. (2002). The Role of Software Processes and Communication in Offshore Software Development. *Communications of the ACM*, 45(4), pp.193-200.

Gnatz, M., Marschall, F., Popp, G., Rausch, A. & Schwerin, W. (2001). Towards a Living Software Development Process Based on Process Patterns. *Lecture Notes in Computer Science 2077, 8th European Workshop on Software Process Technology EWSPT,* Witten, Germany. pp. 182-202.

Griffiths, S. (1978). Design methodologies - a comparison, *Infotech State of the Art Report*, 41(2), pp. 133-166.

Grossman, F., Bergin, J., Leip, D., Merritt, S. & Gotel, O. (2004). One XP Experience: Introducing Agile (XP) Software Development into a Culture that is Willing but not Ready. *Proceedings of the 2004 Conference of the Centre for Advanced Studies on Collaborative Research*. Ontario, Canada. pp. 242-255.

Hackathorn, R. & Karimi, J. (1988). A Framework for Comparing Information Engineering Methods. *MIS Quarterly*, June 1988, pp. 202-220.

Hargis, G. (2000). Readability and Computer Documentation. *ACM Journal of Computer Documentation*, 24(3), pp. 122-131.

Harrison, W. (2004). Best Practices: Who Says? *IEEE Software*, January/February 2004, pp. 8-9.

Heil, M. (1999). Preparing Technical Communicators for Future Workplaces: A Model that Integrates Teaming, Professional Communication Skills, and a Software Development Process. I*Proceedings of the 17th Annual International Conference on Computer Documentation*. New Orleans, Louisiana. pp.110-119.

Henderson-Sellers, B., Cossentino, M., Gaglio, S. & Seidita, V. (2006). A Metamodeling-based Approach for Method Fragment Comparison. Presentation at *Eleventh International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'06)*. Luxembourg.

Heylighen, F. (1996). What is Complexity?. Retrieved 29 January 2007 from: http://pespmc1.vub.ac.be/complexi.html.

Highsmith, J. (2002). *Agile Software Development Ecosystems*. Boston: Addison-Wesley.

Hong, S., van den Goor, G. & Brinkkemper, S. (1993). A Formal Approach to the Comparison of Object-Oriented Analysis and Design Methodologies. *The Proceedings of the 26th Hawaii International Conference on System Sciences.* Hawaii. pp. 689-698.

Humphrey, W. (1995). *A Discipline for Software Engineering*. Boston: Addison-Wesley.

Humphries, S. (1997). *Modular Concept*. Retrieved 19 October 2006 from: http://www.life.uiuc.edu/plantbio/wimovac/modular.htm.

International Business Machines (IBM). (2004). *The Rational Unified Process*. Retrieved 25 February 2005 from: http://www.ibm.com

International Business Machines (IBM). (2004 [2]). *Swedbank Improves Consistency, Efficiency and Business Alignment with IBM Rational Unified Process and IBM Rational Development Tools*. Retrieved 21 December 2006 from: http://www3.software.ibm.com/ibmdl/pub/software/rational/web/success/swedbank.pdf.

Jacobson, I. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach, Fourth Edition*. Boston: Addison-Wesley.

Javed, T., Maqsood, M. & Durrani, Q. (2004). A Survey to Examine the Effect of Team Communication on Job Satisfaction in Software Industry. *ACM SIGSOFT Software Engineering Notes*, 29(2), pp.6-13.

Jayaratna, N. (1994). *Understanding and Evaluating Methodologies. NIMSAD: A Systematic Framework*. London: McGraw-Hill.

Jeffries, R., Anderson, A. & Hendrickson, C. (2000). *Extreme Programming Installed*. Boston: Addison-Wesley Professional.

Johnson, R. (2000). The Ups and Downs of Object-Oriented Software Development. *Communications of the ACM*, 43(10), pp. 69-73.

Keenan, F. (2004). Agile Process Tailoring and probLem analYsis (APTLY). *Proceedings of the 26th International Conference on Software Engineering*. Edinburgh, Scotland. pp. 45-47.

Kelly, J. (1987). A Comparison of Four Design Methods for Real-Time Systems. *Proceedings of the 9th International Conference on Software Engineering*. Monterey, California, USA. pp. 238-252.

Kelley, M. & Rogers, K. (1992). Software Engineering Environment Framework Evaluation Method: Three Examples. *Proceedings of the Ninth Washington Ada Symposium on Ada: Empowering Software Users and Developers*. Washingon D.C., Washinton, USA. pp. 176-180.

Kiely, G. & Fitzgerald, B. (2003). An Investigation of the Use of Methods Within Information Systems Development Projects. *Proceedings of Information Systems Perspectives and Challenges in the Context of Globalisation*. Athens, Greece.

Kitchenham, B. (1996). Evaluating Software Engineering Methods and Tool. Part 1: The Evaluation Context and Methods. *ACM SIGSOFT Software Engineering Notes*, 21(1), pp. 11-15.

Kolling, M., Koch, B., & Rosenberg, J. (1995). Requirements For a First Year Object-Oriented Teaching Language. *Proceedings of the Twenty-sixth SIGCSE Technical Symposium on Computer Science Education*. Nashville, Tennessee, USA. pp. 173-177.

Kolling, M. & Rosenberg, J. (1996). Blue – A Language for Teaching Object-Oriented Programming. *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education*. Philadelphia, Pennsylvania, USA. pp. 190-194.

Kolling, M. & Rosenberg, J. (2001). Guidelines for Teaching Object-Orientation with Java. *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education*. Canterbury, United Kingdom. pp. 33-36.

Krogstie, J. (2000). The UML as a Basis for the Development of Models of High Quality. *Proceedings of the Fifth CAISE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design* Stockholm, Sweden.

Krutchen, P. (1999). *The Rational Unified Process: An Introduction*. Boston: Addison-Wesley/Longman.

Land, F. (1976). Evaluation of Systems Goals in Determining a Design Strategy for a Computer-Based Information System. *Computer Journal*, 19(4), pp. 290-294.

Ledeczi, A., Nordstrom, G., Karsai, G., Volgyesi, P. & Maroti, M. (2001). On Metamodel Composition. *Proceedings of the IEEE Conference on Control Applications*. Mexico City, Mexico.

ManagementUpdate.Info. (2007). *Best Practice – Organisation Development*. Retrieved 29 January 2007 from: http://www.managementupdate.info/organization-development/best-practice/.

Mannino, P. (1987). A Presentation and Comparison of Four Information Systems Development Methodologies. *ACM Sigsoft Software Engineering Notes,* 12(2), pp. 26-33.

Matinlassi, M. (2004). Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA. *Proceedings of the 26th International Conference on Software Engineering*. Edinburgh, Scotland. pp. 127-136.

Maybank, S. (1999). *History of SSADM*. Retrieved 23 December 2006 from: http://www.dcs.bbk.ac.uk/~steve/1/tsld005.htm.

McCabe, T., Butler, C. (1989). Design Complexity Measurement and Testing. *Communications of the ACM,* 32(12), pp. 1415-1425.

McLeod, G. (1992). *Managing Methods Creatively*. Retrieved 5 September 2005 from: http://www.inspired.org

McLeod, G. (1997). Method Points: Towards a Metric for Method Complexity. *Proceedings of the Second CAISE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*. Barcelona, Spain.

McLeod, G. & Roeleveld, D. (2002). Method Evaluation in Practice: UML/RUP & Inspired Method. *Proceedings of the 7th CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*. Toronto, Canada.

McNurlin, B. & Sprague, R. (1993). Information Systems Management in Practice. Saddle River: Prentice Hall International Inc.

Microsoft. (2003). *Analysing Requirements and Defining Microsoft .Net Solution Architectures*. Redmond: Microsoft Press.

Mikulovic, V. & Heiss, M. (2006). "How do I know what I have to do?": The Role of the Inquiry Culture in Requirements Communication for Distributed Software Development Projects. *Proceedings of the 28th International Conference on Software Engineering*. Shanghai, China. pp. 921-925.

Miller, G. (2001). The Characteristics of Agile Software Processes. *The 39th International Conference of Object-Oriented Languages and Systems (TOOLS, 39)*. Santa Barbara, California, USA.

Misic, V. (2006). Perceptions of Extreme Programming: An Exploratory Study. *ACM SIGSOFT Software Engineering Notes,* 31(2), pp. 4-11.

Moody, D. (2000). *Dealing with Complexity: A Practical Method for Representing Large Entity Relationship Models*. PhD Thesis. University of Melbourne, Australia.

Moody, D., Sindre, G., Brasethvik, T. & Sølvberg, A. (2001). Evaluating the Quality of Information Models: Empirical Testing of a Conceptual Model Framework. *Proceedings of the 25th International Conference on Software Engineering.* Portland, Oregon, USA. pp. 295-305.

Nance, R. & Arthur, J. (1998). The Methodology Roles in the Realization of a Model Development Environment. *Proceedings of the 1998 Winter Simulation Conference*. Washington D.C., Washington, USA. pp. 220-225.

Nebulon. (2002). *Nebulon website*. Retrieved 24 December 2006 from: http://www.nebulon.com.

Noble, J., Marshall, S., Marshall, S., & Biddle, R. (2004). Less Extreme Programming. *Proceedings of the Sixth Conference on Australasian Computing Education*. Dunedin, New Zealand. pp. 217-224.

Object Management Group (OMG). (2005). *Software Process Engineering Metamodel Specification.* Retrieved 7 October 2006 from http://www.omg.org. Document number: 05-01-06.

Olsen, D. & Wood. S. (2004). Fan-out: Measuring Human Control of Multiple Robots. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Vienna, Austria. pp. 231-238.

Palmer, S. (2002). Feature Driven Development and Extreme Programming. *The Coad Letter: Modeling and Design.* Vol 70.

Palmer, S., Felsing, J. (2002). *A Practical Guide to Feature-Driven Development*. New Jersey: Prentice-Hall.

Perks, M. (2003). *Best Practices for Software Development Projects*. Retrieved 18 October 2005 from: http://www-128.ibm.com/developerworks/websphere/library/techarticles/0306_perks/perks2.html.

Pierce, R. (2005). Leveraging Technology Affinity: Applying a Common Set of Tools and Practices to Information Development. *Proceedings of the 23rd International Conference on Design of Communication (SIGDOC).* Coventry, UK. pp. 123-130.

Pressman, R. (2005). *Software Engineering: A Practitioner's Approach, Sixth Edition*. London: McGraw-Hill.

Probert, S. (2001). Modeling using IS Methodologies: Some Guidelines Based on Authenticity and Contemporary Epistemology. *Informing Science*, June 2001, pp. 437-443.

Purao, S. & Vaishnavi, V. (2003). Product Metrics for Object-Oriented Systems. *ACM Computing Surveys,* 35(2), pp. 191–221.

Ramesh, B., Cao, L., Mohan, K. & Xu, P. (2006). Can Distributed Software Development be Agile? *Communications of the ACM*, 49(10), pp. 41-46.

Ramos, I. & Roode, D. No Date. *Best Practices Considered Harmful*. Unpublished journal paper.

Rolland, C. (1997). A Primer for Method Engineering. *Proceedings of the INFORSID Conference 1997: CREWS Report Series 97-06*. Toulouse, France.

Rozman, I., Gyorkos, J. & Rizman, K. (1992). Understandability of the Software Engineering Method as an Important Factor in Selecting a CASE Tool. *ACM SIGSOFT Software Engineering Notes*, 17(3), pp.43-46.

Shami, S., Bos, N., Wright, Z., Hoch, S., Kuan, K., Olson, J. & Olson, G. (2004). An Experimental Simulation of Multi-Site Software Development. *Proceedings of the 2004 Conference of the Centre for Advanced Studies on Collaborative Research*. Markham, Ontario. pp. 255-266.

Singh, S. & Kotze, P. (2003). An Overview of Systems Design and Development Methodologies with Regard to the Involvement of Users and Other Stakeholders. *Proceedings of the 2003 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement through Technology (SAICSIT '03)*. Johannesburg, South Africa. pp. 37-47.

Smart, K. (2002). Assessing Quality Documents. *ACM Journal of Computer Documents*, 26(3), pp. 130-140.

Sol, H. (1983). A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations. *Information Systems Design Methodologies: A Comparative Review*. Olle, W., Sol, H., Tully, C. (eds), Amsterdam: North-Holland Publishers.

Solms, F. (2004). *Business Analysis Using UML*. Johannesburg: Solms Training, Consulting and Development.

Sommerville, I. (2004). *Software Engineering, 7th Edition*. Boston: Pearson/Addison-Wesley.

Sorenson, R. (1995). *A Comparison of Software Development Methodologies.* Retrieved 8 September 2005 from: http://www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1995/01/Comparis.asp.

StatSoft Inc (2004). *Statistica 6.1 Elecronic Manual*.

Stephens, M. (2004). *The Case Against Extreme Programming: A Self-Referential Safety Net*. Retrieved 10 August 2004 from: http://www.softwarereality.com/lifecycle/xp/safety_net.jsp.

Sullivan, K., Griswold, W., Cai, Y. & Hallen, B.(2001). The Structure and Value of Modularity in Software Design. In *Proceedings of the 8th European Software Engineering Conference*. Vienna, Austria. pp. 99-108.

Thayer, R. (2002). Software Engineering Project Management. *Software Engineering, 2nd Edition: Volume 2 -- The Supporting Processes,* Richard H. Thayer and Mark Christensen (eds), Washington: IEEE Computer Society Press, pp. 227-267.

The Object Agency. (1995). *A Comparison of Object-Oriented Methodologies*. Retrieved 1 September 2005 from: http://www.toa.org.

Upshall, M. (ed). (1992). Hutchinson Concise Encyclopedic Dictionary. London: Helicon Publishing.

Van Belle, J. (2002). Towards a Syntactic Signature for Domain Models: Proposed Descriptive Metrics for Visualizing the Entity Fan-Out Frequency Distribution. *Proceedings of the 2002 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement through Technology (SAICSIT '02)*. Port Elizabeth, South Africa. pp. 210-215.

Van Belle, J. (2003). *A Framework for the Analysis and Evaluation of Enterprise Models*. PhD Thesis. University of Cape Town, South Africa.

Van De Weerd, I. (2005). *WEM: A Design Method for CMS-based Web Implementations*. Technical Report UU-CS-2005-043, Institute of Information and Computing Sciences, Utrecht University.

Van den Bosch, F., Ellis, J., Freeman, P., Johnson, L., McClure, C., Robinson, D., Scacchi, W., Scheff, B., von Staa, A. & Tripp, L. (1982). Evaluation of Software Development Life Cycle: Methodology Implementation. *ACM SIGSOFT Software Engineering Notes*, 7(1), pp. 45-60.

Wagner, S. (2003). *NIMSAD Evaluation of the Rational Unified Process*. Retrieved 6 August 2006 from http://www.stefan-wagner.info/index.php.

Withers, D. (2000). Software Engineering Best Practices Applied to the Modeling Process. *Proceedings of the 2000 Winter Simulation Conference*. Orlando, Florida, USA. pp. 432-439.

Zachman, J. (1987). A Framework for Information Systems Architecture. *IBM Systems Journal*, 26(3), pp.276-292.

Zhou, M. & Zhu, H. (2003). Methodology First and Language Second: A Way to Teach Object-Oriented Programming. *Companion of the 18th annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*. Anaheim, California. pp. 140-147.

Appendix A: Methodology to Zachman Framework Mappings

SSADM	Motivation (Why)	People (Who)	Function (How)	Data (What)	Time (When)	Network (Where)
Scope View		Organisational Structure	Requirements Catalog			
Owners View		User Roles	Level 1 Data Flow Diagram	Logical Data Structure		
Designers View		User Catalog	Level 2 Data Flow Diagram	Logical Data Model	Entity Life History	
Builders View	Update/Enquiry Process Models	Application Style Guide	Function Definition	Physical Data Design		
Detailed View		User Dialogues	Update/Enquiry Process Models	Data Catalog	Timing Definitions	

Table 36: SSADM/Zachman Framework Mapping

RUP	Motivation (Why)	People (Who)	Function (How)	Data (What)	Time (When)	Network (Where)
Scope View	Vision: Needs	Vision. Stakeholders	Vision: Features	Business Entities	Business Workflows	
Owners View	Business Rules	Actors	Use Cases	Business Object Model	Use case flow of events	Network Configurations
Designers View	Constraints, Multiplicities, workflow activities	Boundary Classes	Use Case Realisations	Persistent Classes	Interaction diagrams	Deployment model
Builders View		End User Support Material	Components	Data Model	Process Model	Process-to-node mapping
Detailed View		UI Design Classes	Design Classes	Columns, types, keys, indices	State machines	

Table 37: RUP/Zachman Framework Mapping (from de Villiers, 2001, p. 8)

MSF	Motivation (Why)	People (Who)	Function (How)	Data (What)	Time (When)	Network (Where)
Scope View	Vision Statement: Project Goals	Vision: User Profiles	Vision Statement: Design Goals/Scope	Vision Statement: Solution Concept	Vision Statement: Design Goals	Vision Statement: Design Goals
Owners View	Business Rules Catalog	Actors Catalog	Functional Specification: Conceptual Design Summary (Use Cases)	Logical Object Model	Conceptual Design: Use Cases	Network Topology
Designers View	Conceptual Design: User Requirements	User Services	Application Architecture	Data Schema	Logical Design: Sequence Diagrams	Component and Data Topology
Builders View	Physical Design: Usage Scenario	Technical Specification: User Interface	Components	Tables and Columns; Implemented Relationships	Usage Scenario/Physical Design: Sequence Diagrams	Deployment model
Detailed View			IDL Definitions of Published Interfaces	Class Model	Logical Design: State Diagrams	

Table 38: MSF/Zachman Framework Mapping

FDD	Motivation (Why)	People (Who)	Function (How)	Data (What)	Time (When)	Network (Where)
Scope View		Develop an Overall Model: Form the Modelling Team	Main Features		Develop an Overall Model: Sequence Diagram	
Owners View		Object Model: Role	List of Feature Sets per Main Feature	Develop an Overall Model Overall Object Model		
Designers View		Sequence Diagram/Refined Class Model Roles	List of Features per Feature Set	Class Diagrams with Operations and Attributes	Design By Feature: Sequence Diagrams	
Builders View		Sequence Diagram/Refined Class Model Roles		Refined Object Model		
Detailed View				Class and Method Prologues		

Table 39: FDD/Zachman Framework Mapping

XP	Motivation (Why)	People (Who)	Function (How)	Data (What)	Time (When)	Network (Where)
Scope View			Metaphor			
Owners View						
Designers View	Functional Tests		Story Cards			
Builders View	Unit Tests					
Detailed View						

Table 40: XP/Zachman Framework Mapping

OOAD	Motivation (Why)	People (Who)	Function (How)	Data (What)	Time (When)	Network (Where)
Scope View						
Owners View			Primary and Secondary Scenarios	Class Model		
Designers View			Function Partitions	Object Model	Interaction diagrams	
Builders View			Abstractions with Set of Responsibilities	Module Diagrams	State Transition Diagrams	Process Diagram
Detailed View			Implemented Classes	Data Dictionary		

Table 41: OOAD/Zachman Framework Mapping

Appendix B: Best Practice Sets

The following sets of best practices were used in the Best Practice Coverage analysis.

The Rational Unified Process [Set 1]

IBM (2004:*Best Practices*) describes the six best practices for the Rational Unified Process. These best practices involve using iterative development in order to divide the work in manageable and tangible sections, managing the requirements for the system, using visual techniques (specifically the Unified Modeling Language) to model the proposed system, the use of a component-based architecture such as Object-Orientated architectures (which are supported by the Unified Modeling Language), continuously verifying the quality of the system through a dedicated and planned testing process, and the managing of system and requirements change.

The Enterprise Unified Process [Set 2]

In a similar vein, Ambler (2005) proposes ten best practices based on the Enterprise Unified Process (EUP), which is an extension of the Rational Unified Process (RUP). These best practices include developing the project in iterations, managing requirements through their identification, gathering, documenting and maintaining them and identifying and using a proven (although possibly not component-based) architecture, often through prototyping practices.

Ambler (2005) also states that the system should be modelled (perhaps not visually as the RUP suggests), that quality should be continuously verified through testing, that change should be managed continuously, as well as collaborative development, be it through pair programming, active stakeholder participation or other means. The final best practices suggested by Ambler (2005) are looking beyond development to system operation and support (including standards and guidelines), incrementally delivering working software to the users of the system on a regular basis, and managing risk through identification and mitigation.

Feature-Driven Development [Set 3]

Palmer and Felsing (2002, pp.36-54) list eight best practices for Feature-Driven Development, namely domain object modeling, or the building of class diagrams to represent the system within it's problem domain; developing by feature, or using feature sets to describe the functionality of the system; individual class (or code) ownership, where people are given responsibility, or ownership, for the contents of a specific class or piece of program code; the use of feature teams, or teams created dynamically for specific features, based on the classes used in those features and the owners of said classes; using inspections to ensure the quality of the system being developed; the use of a regular build schedule to incrementally deliver software to the client or users; the use of configuration management, although theoretically only to identify the source code for the classes that have been completed up to a certain date and maintain a history of changes to the code; and reporting or visibility of results, in order to allow for planning and scheduling as well as the tracking of progress.

Perks (2003) [Set 4]

Perks (2003) defines 16 best practices for software development projects. These processes include selecting the correct development process for the type of project being undertaken, gathering and agreeing on functional and non-functional requirements, choosing the correct architecture for the intended application, and using the correct system design approach, possibly using Object-Oriented Analysis and Design or the Rational Unified Process. Other best practices suggested by Perks (2003) include the use of a defined process for system construction, the use of peer reviews of project work and the planning of testing.

Perks also suggests the use of performance testing in order to minimise performance and application defects, configuration management, quality and defects management, possibly using quality priorities and release criteria, deployment planning, possibly using a deployment checklist, planning for systems operation and support, planning for and correctly resourcing data migrations, using project management best practices, and measuring the project success, possibly using the Capability Maturity Model. Finally, Perks (2003) suggests the best practice of designing applications specifically for IBM's WebSphere application server, but this best practice will be disregarded, as WebSphere is not a universally used product and may well not be used in many software development projects.

The Airlie Software Council (ASC) [Set 5]

The ASC (Withers, 2000, pp. 432-435) summarises a number of best practices, as well as three categories, based on the best practices identified by the United States' Department of Defence-sponsored Airlie Software Council (ASC). These best practices, originally identified as a list of nine in 1994, were later modified and expanded to sixteen, at which they were categorised by the Software Program Managers Network in 1999. The first category is project integrity, where the best practices relate to project management. These best practices include the use of formal risk management, the estimation of cost and empirical scheduling, the use of metric-based scheduling and management, the tracking of earned value as the project progresses, tracking defects against quality targets and treating people as the most important resource on a project.

The second category suggested by the ASC (Withers, 2000, p. 434) involves construction integrity and is concerned with suggested best practices for design and development activities. These best practices include the use of configuration management (managing and controlling all changes to shared project information), the management and traceability of requirements, using the practice of system-based software design, ensuring that the data and database are interoperable, the definition and control of interfaces to other systems, the principle of designing twice and developing once, and the assessment of the risks and costs of artefact reuse.

The third category, product stability and integrity, is concerned with best practices for ensuring the quality of the resulting product and includes three individual best practices (Withers, 2000, pp. 434, 435). These best practices are the use of formal requirements and design inspections, the activity of managing testing as a continuous process, and the principle of smoke testing frequently, involving regression testing.

Extreme Programming [Set 6]

According to Beck (1999, p. 46) and Jeffries, Anderson and Hendrickson (2000, p. 5), Extreme Programming has twelve key, or best practices, which can be broken down into three key categories, as described in *Table 42:*

Activity	Practice
Programming	Simple design, testing, refactoring, coding standards
Team practices	Collective ownership, continuous integration, metaphor, coding standards, 40-hour week, pair programming, small releases
Processes	On-site customer, testing, small releases, planning game

Table 42: XP Categories and Practices (from Beck, 1999, p. 46, Jeffries et al., 2000, p. 5)

Beck (1999, p. 46) and Jeffries *et al.* (2000, pp. 5 – 7) describes the practices as follows:

- Simple design, or designing the system as simply as possible at any given moment and removing extra complexity as soon as it is discovered;
- Testing, through unit and system tests;
- Refactoring, or restructuring the system without changing its behaviour to remove duplication, improve communication, simplify, or add flexibility;
- Coding standards, or writing all code in accordance with rules emphasizing communication;
- Collective ownership, or making any programmers on the team able to change any code anywhere in the system at any time;
- Continuous integration, or integrating and building the system many times a day or every time a task is completed;
- Metaphor, or guiding all development by a simple shared story of how the whole system works;
- 40-hour week, or encouraging team members not to work more than 40 hours a week as a rule;
- Pair programming, or mandating that all production code must be written with two programmers at one computer;
- Small releases, or ensuring that a simple system should be put into production quickly, with new versions released on a very short cycle;
- Onsite Customer, or including a real user of the proposed system on the team, available full-time to answer questions; and
- The Planning Game, or quickly determining the scope of the next release by combining business priorities and technical estimates and updating the plan if needed.

Appendix C: Methodology Reference Material Readability Scores

	Reference Material	Type	FRE	FKGL
FDD	A Practical Guide to Feature-Driven Development (Palmer and Felsing, 2003)	Book	44.5	12
MSF	Analysing Requirements and Defining Microsoft .Net Solution Architectures (Microsoft, 2003)	Book	32	12
OOAD	Object-Oriented Analysis and Design with Applications (Booch, 1994)	Book	27.2	12
RUP	The Rational Unified Process (IBM, 2004)	Electronic Resource	36.6	12
SSADM	SSADM Version 4 – A Practical Approach (Goodland and Slater, 1995)	Book	33.6	12
XP	**Average**		**61.2**	**8.53**
	Extreme Programming Explained (Beck, 1999)	Book	59.7	8.7
	Extreme Programming Installed (Jeffries *et al*, 2000)	Book	66.5	8
	Extreme Programming Explored (Wake, 2000)	Book	57.4	8.9

Table 43: Methodology Reference Material Readability Scores