

**A SELECTION FRAMEWORK FOR AGILE METHODOLOGY
PRACTICES: A Family of Methodologies Approach**

Ernest Mnkandla

A thesis submitted to the Faculty of Engineering and
the Built Environment, University of The
Witwatersrand, Johannesburg, in fulfillment of the
requirements for the degree of Doctor of Philosophy.

Johannesburg, 2008

Declaration

I declare that this thesis is my own, unaided work. It is being submitted for the degree of Doctor of Philosophy in the University of The Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

(Signature of candidate)

8th day of January 2008

Abstract

The challenges of selecting appropriate software development methodologies for a given project, and tailoring the methodologies to a specific human culture have been dealt with since the establishment of software development as a discipline. The general trend in software development methodologies since the early 1990s has been to change from plan-driven approaches to more iterative incremental development approaches. This change has led to the birth of a group of methodologies called “agile methodologies”, whose values and principles are becoming more and more prevalent in the software development industry. The practice in this industry has since 2004 shifted from an initial focus on the selection of specific agile methodologies to selecting the most appropriate practices from the agile family. Previous work on methodology selection has centered on either choosing between agile and plan-driven approaches or choosing specific methodologies from these two groups. This research aims at developing a framework for selecting the most appropriate set of agile practices for a given project instead of selecting a specific agile methodology.

The results of this research contribute towards greater understanding of agile software development issues and should be useful to developer firms that want to adopt agile methodologies as a generic development culture without worrying about specific agile methodologies.

To

My wife Sindi, my son Awande Kevin and my daughter Omuhle

Acknowledgements

I would like to thank my supervisor Professor Barry Dwolatzky for his untiring assistance in guiding me into the world of scientific writing. Many thanks also go to Monash South Africa for financial assistance and providing state-of-the-art facilities. Thanks to Sifiso Mlotshwa (Ngiyabonga mzala) for openly discussing details of the software development activities in his IT consultancy organization. Many thanks go to Fazel Mayet and Barry Myburgh for their valued contributions to my study. I thank my colleague Judy Backhouse and my friend Joseph Bhekizwe Dlodlo for taking the trouble to read my thesis and make valuable comments. I would not forget to thank Jan Meyer those valuable discussions on research methodology. I thank Total Services Solutions Education South Africa for giving me practical experience in a real world project. Space will not allow the list of fellow professionals and practitioners who provided valuable information through formal interviews, emails and informal discussions at conferences and seminars.

Table of Contents

1	Introduction	1
1.1	Background to the Area of Work	1
1.1.1	Early Software Development Methodologies	1
1.1.2	Agile Methodologies	2
1.1.3	Methodology Jungle	3
1.2	Thesis Outline	4
1.3	Conclusion	5
2	Research Questions and Approach	6
2.1	Problem Definition	6
2.1.1	Problem statement	6
2.1.2	Objectives of the study	6
2.2	Hypothesis	7
2.3	Justification of the Research	9
2.4	Research Methodology	11
3	Background and Foundational Analysis	13
3.1	Software Crisis	14
3.2	Software Engineering	17
3.2.1	The Software Development Problem	19
3.3	Software Engineering Process	22
3.3.1	Management of the Software Engineering Process	23
3.4	Software Development Projects	24
3.4.1	Project Classification	25
3.4.2	Project Size	25
3.4.3	Taxonomy of Software Development Projects	26
3.5	Software Development Methodologies	27
3.5.1	Rigorous Methodologies	28
3.5.2	Iterative Incremental Development Methodologies	29
3.5.3	Representation of Software Development Methodologies	30
3.5.4	Representation of Agile Methodologies	37
3.6	Agile Methodologies	41
3.6.1	Adaptive Software Development (ASD)	48
3.6.2	Crystal	51
3.6.3	Feature Driven Development (FDD)	54
3.6.4	Agile Model Driven Development (AMDD)	58
3.6.5	ICONIX	59
3.6.6	Scrum	62
3.6.7	Dynamic System Development Methodology (DSDM)	65
3.6.8	Lean Development (LD)	68
3.6.9	Extreme Programming (XP)	70
3.6.10	Notes on Adopting Agile Methodologies	73
3.7	Methodology Selection Approaches	74
3.7.1	Methodology Selection in South Africa	75
3.8	Conclusions	77

4	Methodology Selection and Tailoring Framework	78
4.1	Background	79
4.2	Architecture of the Framework	80
4.2.1	Philosophy of the Framework	81
4.2.2	Process of the Framework	81
4.3	Stage one: Project Classification Matrix	83
4.3.1	Requirements Stability	85
4.3.2	Project Size	87
4.3.3	Project Development Timeframe	88
4.3.4	Project Complexity and Criticality	88
4.3.5	Project Risk	91
4.3.6	Customer's Priorities/Availability	91
4.3.7	Final Choice	92
4.4	Stage two: Methodology Selection Matrix	92
4.4.1	Methodology Philosophy	98
4.4.2	Methodology Process	98
4.4.3	Methodology Techniques and Tools	98
4.4.4	Methodology Scope	99
4.4.5	Methodology Outputs	99
4.4.6	Adoption and Experience	100
4.4.7	Methodology Product	100
4.4.8	Roles and Responsibilities	100
4.4.9	Support for Distributed Teams	100
4.4.10	Partial Agile Methodologies	101
4.4.11	The Role of the Human Expert	101
4.5	Stage Three: Tailoring Agile Methodology Practices	102
4.5.1	A Family of Methodologies Model	103
4.5.2	The Generic Agile Methodologies (GAM) Model	104
4.6	Conclusions	107
5	Implementation of the Framework	108
5.1	Implementation of Project Classification Matrix	109
5.2	Implementation of Agile Methodologies Selection Matrix	110
5.3	Implementation of the GAM Model	119
5.4	Methodology Tailoring in a South African Context	121
5.5	Framework Reasoning Process and Knowledge Contribution	123
5.6	Conclusions	124
6	Research Methodology and Validation	126
6.1	Qualitative Research Method	126
6.2	The Research Approach	127
6.3	Rationale for Choice of Approach	130
6.3.1	Choice of Methodology Experts	132
6.3.2	Data Gathering	132
6.3.3	Results	134
6.3.4	Analysis	142
6.4	Validation Approach	144

6.5	Results and Analysis	146
6.6	Discussion	148
6.7	Evaluation	149
6.8	Conclusion	150
7	Summary and Future Work.....	151
7.1	Summary	151
7.2	Knowledge Contributions	153
7.3	Future Work	154
7.4	Limitations	155
7.5	Lessons Learnt	155
8	References	158
9	Appendices.....	179
9.1	Appendix A Agile Manifesto for Agile Software Development	179
9.1.1	Principles behind the Agile Manifesto.....	179
9.2	Appendix C The Questionnaire and Interviews.....	183
9.2.1	Software Development Questionnaire	183
9.3	Appendix D Personal Correspondence	189
9.3.1	Interview Data.....	189
9.4	Appendix E Framework Validation Document.....	191
Testers		200
	develop test cases.....	200
	test	200
	report on bugs.....	200
Interaction designer / Analyst		200
	develop stories.....	200
	keep model up to date	200
Architects		200
	look for and execute large scale refactorings	200
	system level tests that stress architecture.....	200
	Partitioning.....	200
Project manager		200
	communication inside team	200
	communication outside team.....	200
	keeps metrics of progress	200
Technical writer		200
	create closer relationship with users	200
	write manual	200
Users		200
	pick stories.....	200
	domain decisions.....	200
Programmers		200
	break stories into tasks.....	200
	estimate tasks	200
	write tests.....	200

write code	200
implement features.....	200
automate tedious development processes.....	200
improve design of system	200
development infrastructure	200
Planning game	202
Whole team	202
Informative workspace.....	202
Weekly cycle	202
Ten minute build	202
Continuous integration.....	202
Test first programming.....	202
Incremental design	202
Real customer involvement	202
Shared code	202
Daily deployment.....	202
9.5 Appendix F Screen Dumps for the Generic Agile Methodologies Toolbox	
204	
9.6 Appendix G Methodology Description Tables	217

List of Figures

Figure 2-1: Summary of Research Objectives	7
Figure 3-1: Literature Review Overview	14
Figure 3-2: Definition of Agility.....	43
Figure 3-3: An Overview of Adaptive Software Development	48
Figure 3-4: Crystal Overview.....	51
Figure 3-5: Overview of Feature Driven Development	55
Figure 3-6: AMDD Graphical Overview	58
Figure 3-7: ICONIX Graphical Overview	60
Figure 3-8: Scrum Graphical Overview.....	62
Figure 3-9: DSDM Graphical Overview	65
Figure 3-10: LD Overview.....	68
Figure 3-11: XP Overview	70
Figure 4-1 Agile Methodologies Selection Framework.....	83
Figure 4-2: Summary of Stage Two.....	95
Figure 4-3 Generic Agile Methodologies (GAM) Model.....	105
Figure 5-1: Framework Reasoning Process Graph	124
Figure 6-1: Graphical Overview of the Research Process	129

List of Tables

Table 3-1: Phases of the Software Development Lifecycle.....	22
Table 3-2: Agile Values Principles and Practices	44
Table 4-1: Stage One: Determine Between the Two Major Paradigms.....	84
Table 4-2: Stage Two: Methodology Description.....	96
Table 4-3: Stage Two: Methodology Selection Matrix.....	97
Table 4-4: Methodology Practices	102
Table 4-5: Values of the Agile Manifesto.....	105
Table 5-1: Project Classification Matrix for SS Technologies Project.....	110
Table 5-2: Methodology Description for Extreme Programming.....	113
Table 5-3: Agile Methodologies Selection Matrix for SS Technology Project.....	115
Table 5-4: Agile Practices per Methodology	117
Table 5-5: Customer Values/ Priorities and Practices for WCR Case.....	119
Table 6-1: List of Methodology Experts Contacted.....	135
Table 6-2: List of IT Professionals and Practitioners Interviewed	137
Table 6-3: Framework Evaluation Results.....	138
Table 6-4: Changes Made to the Initial Version of the GAM Model	143
Table 6-5: Top-level Summary of the Framework	144
Table 6-6: IT Professionals Interviewed	145
Table 9-1: List IT Professionals Interviewed.....	189
Table 9-2: Themes Identified in the Data	190
Table 9-3: Methodology Description for Extreme Programming.....	217
Table 9-4: Methodology Description for Adaptive Software Development.....	218
Table 9-5: Methodology Description Table for Crystal.....	220
Table 9-6: Methodology Description Table for Feature Driven Development.....	222
Table 9-7: Methodology Description Table for Agile Modeling.....	225
Table 9-8: Methodology Description Table for ICONIX	226
Table 9-9: Methodology Description Table for Scrum.....	228
Table 9-10: Methodology Description Table for Dynamic System Development Methodology	231
Table 9-11: Methodology Description Table for Lean Development.....	235

1 Introduction

This thesis presents a Generic Agile Methodologies (GAM) model designed for the selection of the most appropriate agile practices from the agile family. This chapter motivates the need for continued research on current approaches in methodology selection, and thus provides a background to the problems surrounding issues of methodology selection and tuning. The last part of the chapter provides the outline of the thesis.

1.1 Background to the Area of Work

The work presented here belongs to the broad field of software engineering and specifically falls within the area of software development methodologies. A review of the literature shows a progression from early software development methodologies through current research in agile methodologies.

1.1.1 Early Software Development Methodologies

Since the early days of computing the demand for software applications has increased from simple applications that replaced the tasks of the human computer (Parr, 1982) to the complex military applications for controlling weapons such as patriot and stinger missiles to name but a few. The initial demand for software increased at such a rate that it became obvious that a more formalized approach to the development of software had become inevitable. Surprisingly even in the twenty-first century the demand for more software still outstrips the supply. During this era of the history of computing, software development problems led to what became known as the software crisis (Olerup, 1991; Yeh, 1999; Friedman, 1989; van Vliet, 2003; Pressman, 2000; Brooks, 1987; Naur, 1996; Glass, 1998; and Shemer, 1987). The software crisis (discussed in detail in Chapter Three) not only refers to problems associated with approaches to developing software but also includes the broader aspects of how to maintain software. Efforts towards solving these problems led to

the birth of a new discipline called software engineering ascribed to the 1968 NATO Software Engineering Conference held in Garmisch, Germany.

The limitations of the classical software engineering paradigm are rooted in a focus on sequential thinking which focuses on the activities presumed to be necessary to produce a result and assumes that one has nothing to learn so that each activity can be completed in sequence (Larman, 2004 and Hoffer, George and Valacich, 2004). To try and solve some of these problems many methodologies emerged from the early 1970s up to the present with the addition of agile methodologies over the last ten to fifteen years.

1.1.2 Agile Methodologies

From a practical perspective agile methodologies¹ emerged from a common discovery among practitioners that their practice had slowly drifted away from the traditional heavy document and process centered development approaches to more people-centered and less document-centered approaches, (Boehm and Turner, 2004; Highsmith, 2002b; Fowler, 2002).

Agile methodologies were originally created for small collocated teams developing software in an iterative, incremental approach, hence use of the term *lightweight methodologies* before agile was adopted, (Fowler, 2002). However, with the use of proper modeling techniques agile methodologies can be applied to large projects, (Ambler, 2002b and Eckstein, 2004). The birth of agile approaches resulted in the development of several methodologies that are based on one or more of the four agile values and the agile principles as declared in the Agile Manifesto², (Agile Alliance, 2001).

¹ See section 3.6 for the definition of agile methodologies

² See appendix A for details on the Agile Manifesto.

The increase in agile methodologies has led to a new challenge of finding the most appropriate agile methodology for a given situation, which is not different from the selection problems found in the rest of the methodologies usually called the methodology jungle problem discussed in the next subsection.

1.1.3 Methodology Jungle

The proliferation of systems development methodologies led to confusion and difficulty in objectively selecting specific methodologies for specific uses, (Avison and Fitzgerald, 2002; Glass and Vessey, 1995; Siau and Rossi, 1998; and Hofstede and Weide, 1993). Cockburn (2004a) classifies software developers into three levels of practice. The three levels describe the stages of learning for a software development practitioner. A level one person is just starting and would not be able to learn many methodologies at once. This is a stage where people look for one methodology that works and they learn it and follow it and expect it to work always. A level two practitioner is at a stage where he or she realizes that the technique will not serve in all situations so he or she looks for rules about when the procedure breaks. At level three developers do not pay much attention to the methodology formula. They pay attention to what is happening on the project and make process adjustments as they work. They understand the desired end effect and apply whatever works to get to that end (Cockburn, 2004a). The three levels of practice therefore add to the subjectivity of methodology selection rather than providing a solution.

It is useful to clarify the meaning of ‘methodology jungle’ in this thesis. In the systems development methodologies context the term methodology jungle refers to the difficulty of selecting the appropriate methodology for a given project.

While the emergence of agile methodologies contributed to the solution of the problems caused by sequential thinking (as practiced in traditional methodologies) a new problem of selecting the most appropriate agile methodology for a given project

resulted. Some methodology selection frameworks have been suggested based on risk (see, e.g. Boehm and Turner, 2004), based on project size (see, e.g. Cockburn, 2000), and based on project criticality (see, e.g. Cockburn, 2004). Research data from field studies suggest that methodology selection frameworks do indeed help in matching a given project to the appropriate process thereby improving the chances of project success (see empirical information in Boehm and Turner, 2004).

A noticeable feature of the published research in agile methodology selection frameworks is the lack of information on selecting a specific agile methodology. The published frameworks tend to provide guidance up to a general level of going agile or traditional.

The author believes that it is implausible to assume that choosing a group of agile methodologies will improve project success. The author also believes that different software development projects need different sets of principles and practices from the agile group in order to form an appropriate process. Chapter two presents the research question.

1.2 Thesis Outline

The following paragraph describes the layout of the thesis.

Chapter one is an introduction. It defines a brief history of the software development problems that led to software development becoming a profession. The issues that led to the emergence of agile methodologies and the problems they intend to solve are also given. It finally gives an outline of the entire thesis. The second chapter gives a summary of the research questions addressed by the thesis and also outlines the research methodology used to embark on the research. Chapter Three gives the summary of literature reviewed, in order to establish the theoretical underpinning of the problem under consideration. Chapter Four is the main part of the thesis where a solution to the identified problem is proposed. Chapter Five describes the validation

process of the proposed solution. Chapter Six outlines the findings, conclusions, and recommendations for further work in the area of software project taxonomy. This is followed by references and the last section contains appendices with some details of certain concepts described in the body of the thesis.

1.3 Conclusion

The chapter gave an introductory background to software engineering problems that led to the development of software development methodologies and the problems that led to the introduction of more methodologies. The problem of methodology selection is introduced in particular the selection issues relating to the agile methodologies group.

The next chapter follows up on this one by giving more detail on the research problems and questions.

2 Research Questions and Approach

This chapter focuses on the problem area of the research by detailing the question dealt with in this research and is followed by an outline of the research approaches adopted in proposing a solution.

2.1 Problem Definition

This section comprises two aspects: a problem statement for the thesis, and an outline of the objectives of the research.

2.1.1 Problem statement

The problem statement is:

“To develop a thinking framework for achieving three things namely: 1) characterization of software development projects in order to determine the suitability of using agile software development methodologies in a given project, 2) selecting the most appropriate set of practices from the family of agile methodologies, 3) tuning the selected set of practices into a feasible way of working ”

Based on the problem statement the following section explains the process and route to be followed for the successful implementation of the proposed solution.

2.1.2 Objectives of the study

This thesis will:

- Determine and define the existing techniques of software project characterization;
- Determine and define the current methods of agile methodology selection;
- Determine and define the parameters that can be used to represent an agile software development methodology;
- Develop a thinking framework, which implements methodology selection in three stages, the project characterization matrix, the methodology selection matrix and the methodology–tuning model; and

- Determine the suitability of the framework in the South African context.

The research objectives are summarized in figure 2.1.

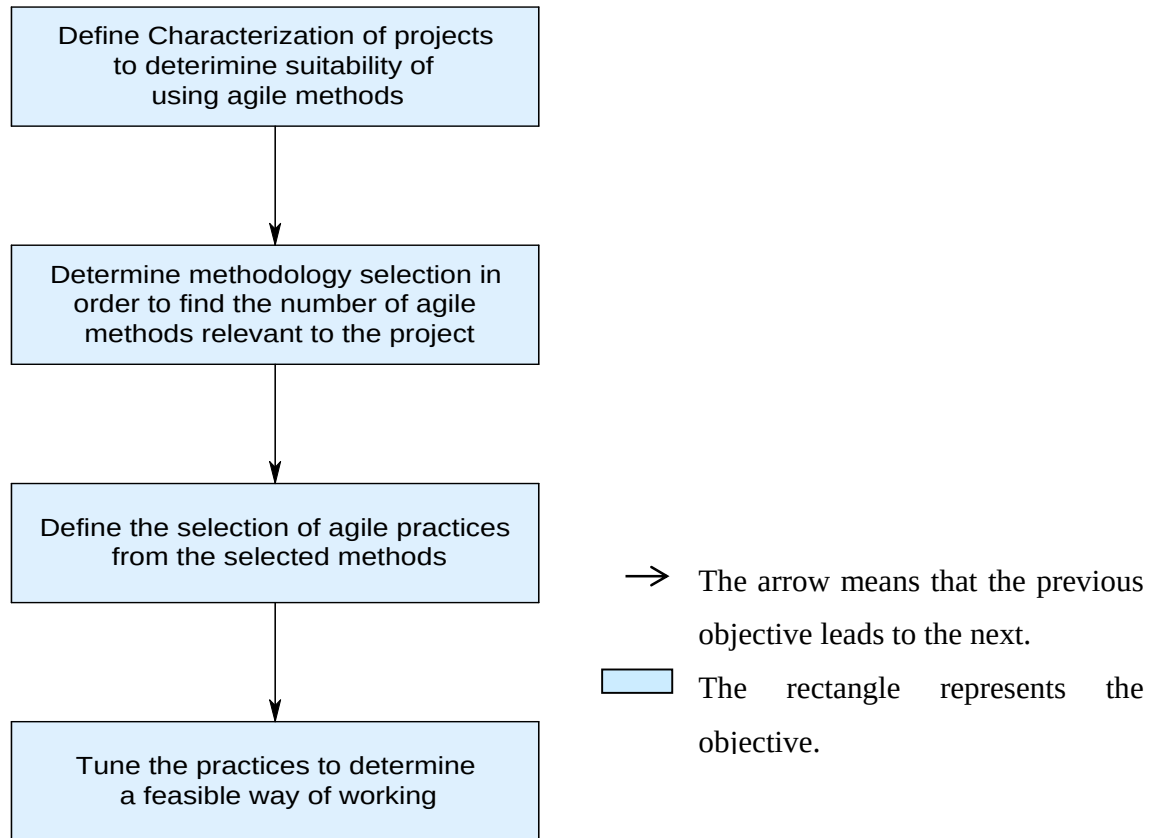


Figure 2-1: Summary of Research Objectives

2.2 Hypothesis

Statement of the hypothesis:

“The success of software development projects can be enhanced by the correct choice of a set of agile methodology practices and the proper adaptation of the selected agile practices to a given project environment”

Success can be viewed in various ways. In this thesis however, success means developing a software product that meets the intended user requirements within time

and budget constraints that are agreeable to the sponsor of the project. The issues of budget overruns and time overruns in software projects are fundamental to this view of success. In the traditional waterfall-based development approaches meeting the budget and time constraints proved to be difficult as witnessed by a high project failure rate which has been decreasing over the years because of various new technologies in software development, (Standish Group, 1994). In the agile methodologies arena the story is different. Success as defined here has been achievable because of the higher level of customer involvement, change-embracing approaches, and iterative incremental development approaches, (Highsmith, 2002a; Eckstein, 2004). However, some have questioned the authenticity of the 1994 Standish report: in particular the article by Robert Glass, in the Loyal Opposition column³, even challenges the Standish Group to explain their views or refute his criticism of their reports (Glass, 2005).

*Agile Methodology Practices*⁴: agile methodologies are made up of values, principles and practices. While agile practices may differ a little according to specific agile methodologies there exist fundamental agile practices that are based on the four agile values and twelve principles and are common to all agile methodologies. Values give purpose to software development, complement each other, and are aligned with life goals (for example, putting more value on development of working software instead of writing comprehensive documentation). Principles are more general and they may clash (for example, cost versus quality, the principle is to maintain low cost and high quality). *Practices* are less flexible, they bring accountability, and they take the purposes depicted in the values to real practice (for example, pair programming improves the level of individual interactions within a team), (Beck and Andres, 2004). From this perspective practices would be easier to adopt as long as the values and principles are understood.

³ Loyal Opposition is a regular column in the IEEE Software journal.

⁴ Agile methodology practices are presented against the corresponding agile values in table 3.2.

The major research questions are:

How does a software development professional select a methodology to be used in a given project?

In as much as there can be a strong desire to follow methodology in the development of software there is a general lack of discipline on the part of the practitioners in using the methodology. The most likely reason for such behavior is that the main purpose of a methodology is planning, but the mission of a software development project is to deliver a quality working software product. Hence the success of a software development project is measured by delivering the software not conforming to a plan, (Highsmith, 2002c). So what it means is that as you move towards meeting the stated goals there may be a general drift from strict use of the methodology. The methodology of choice should therefore allow for such tailoring.

What techniques can be used to match a set of agile methodology practices to a project?

Since the focus is not on the methodology but on the software product, it is important that the approaches to methodology selection be sufficiently generic by not specifying a particular agile methodology. The approach would then be to select relevant agile practices not for the entire project for that matter but for each increment. Meaning that at the beginning of each increment an analysis of the specific requirements and environment for that increment is done followed by the relevant selection process of the matching agile practices.

The next section provides the justification for the research and state the novel aspects.

2.3 Justification of the Research

Published research in the selection of agile methodologies raises three concerns:

- The selection process generally points to a group of methodologies such as traditional or agile (Boehm and Turner, 2004).
- The selection process points to a single methodology as in Cockburn (2000).

- The selection process points to a particular type of agile methodology as in Cockburn (2004).

A further analysis of these frameworks reveals that they are helpful in making a choice between using agile methodologies or traditional methodologies based on risk (Boehm and Turner, 2004); they are also helpful in choosing a particular methodology based on project size (Cockburn, 2000); and they are helpful in choosing a particular agile methodology based on criticality (Cockburn, 2004). However, the mere choice of a group of agile methodologies or a particular agile methodology is not sufficient. It is always necessary to tailor the process into a given environment. Many agile methodologies do not specify how the tailoring should be done (Abrahamsson et al, 2003).

The specific contribution of this study is to take these proven facts two steps further. The first step is to select the principles and practices of agile methodologies that are relevant to a given project, thus providing further help than simply choosing a methodology. In a case study at a company in Finland Pikkarainen and Passoja (2005) discovered that using an agile assessment approach that identified relevant practices from Extreme Programming and Scrum, incremental development monitoring and traceability of requirements improved. The second step is to take the choice of practices to a more feasible level by specifying how to tune the practices to a given environment. The tuning is necessary because agile methodologies provide general guidelines and research shows that for some of the agile methodologies the guidelines are rather abstract (Abrahamsson et al, 2003). The rationale is that according to Nandhakumar and Avison (1999), a useful and efficient method does not refer to abstract principles. Abrahamsson et al (2003) defines abstract principles as principles not supported with concrete guidance. Concrete guidance refers to practices, activities, and work products that characterize and provide guidance on how to execute a specific task, (Abrahamsson et al, 2003). This thesis therefore, proposes a thinking framework that will provide guidance on how to select practices and execute specific tasks.

In the earlier versions of some agile methodologies for example Extreme Programming, (see section 3.6.9) people were encouraged to do all of the methodology in order for it to work (Beck, 1999; Beck and Fowler, 2000; Highsmith, 2002a; and Cockburn, 2002). As experience reports on improvement of these methodologies came, more success stories involved the adoption of principles and practices across different agile methodologies, (Beck and Andres, 2004; Beck, 2005; Highsmith, 2002a; Reifer, 2003; Rasmusson, 2003; Boehm and Turner, 2005). This change in the adoption approach of agile methodologies resulted from a deeper understanding of agile methodologies through which reveals limitations of agile methodologies. Some of the limitations that are noticeable are: a lack of process in some methods, a complex process in others, and a focus on a specific area (for example project management) for others, (Abrahamsson et al, 2003)

The next section summarizes the research methodology followed in solving these problems.

2.4 Research Methodology

This section outlines the research approach followed in this thesis. The research starts by building theory through a literature survey of existing methods of project characterization in order to determine:

- The purpose of project characterization
- Ascertain the existing methods of agile methodology selection
- Establish the need for a methodology selection technique for matching a given project to a set of agile methodology practices.

As part of the theory building stage the author also participated in a project at a software development company. Knowledge gaps were identified and research questions described in this chapter were formulated.

It is then followed by the building of a proposed solution to the identified problems and a partial validation of the solution. The approach is essentially qualitative research based on the underlying philosophies of positivist and critical theories and implemented through grounded method and the instruments of interviews and questionnaires.

The next chapter summarizes the literature review on the evolution of software development methodologies and reveals the arguments leading to the establishment of agile methodologies and the need for a structured framework for the matching of agile methodology practices to a project.

3 Background and Foundational Analysis

This chapter gives a summary of the literature survey that was conducted in the area of software engineering. The focus is on revealing the fundamental problems that software development has always faced. The survey will lead to the review of technologies that have partially solved some of the problems over the years. The work that the South African industry has contributed in the field of software development will also be reviewed. A more detailed review of agile methodologies is given because the proposed methodology selection framework is based on agile methodologies. The chapter concludes by describing the justification for the choice of the proposed framework. Key technical terms will be defined as they appear in this thesis. Figure 3.1 gives an overview of the perspective from which the literature review was conducted.

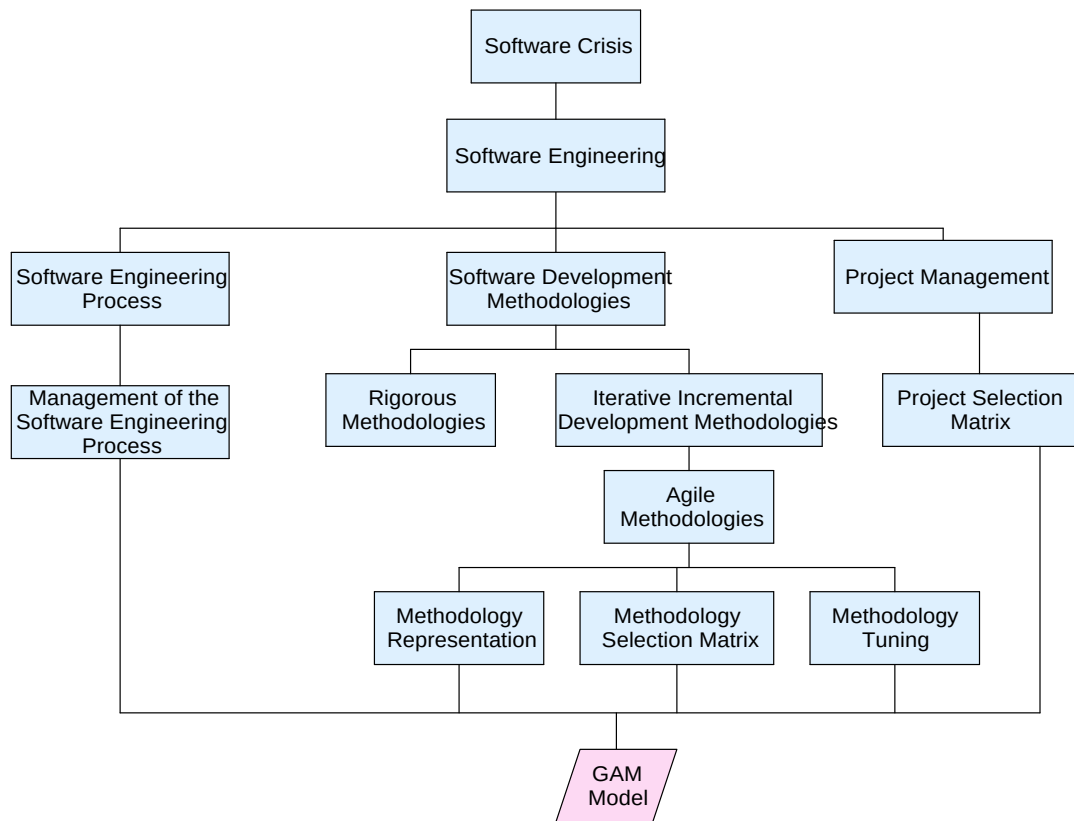


Figure 3-1: Literature Review Overview

3.1 Software Crisis

This term refers to the problems in the early history of software development when programs were developed following unsystematic and random methods, (Olerup, 1991 and Yeh, 1991). As the programs became more complex more programmers would be added and the results were: late delivery of software, software not meeting the intended needs, programs not adaptable to changed circumstances, and many errors were detected after product delivery, which led to difficulties in maintaining software. These problems were referred to as the software crisis, (Friedman, 1989; van Vliet, 2003; Olerup, 1991; Yeh, 1991; Pressman, 2000; Brooks, 1987; Naur, 1996; Glass, 1998 and Shemer, 1987). The software crisis spanned over three decades from the 1960s to the 1980s during which some projects resulted in huge

overspends, wasted money, unacceptably long development timeframe, caused property damage and a few projects caused loss of life, (Glass, 1998; Bryant, 2000 and Brooks, 1987). Some used the term software crisis to refer to their inability to hire qualified personnel. The software crisis was originally defined with regard to productivity, but evolved to emphasize quality, (Glass, 1998, Bryant, 2000 and Brooks, 1987). The software crisis not only refers to problems associated with approaches to developing software but also includes the broader aspects of how to maintain software. Efforts towards solving these problems led to the birth of a new discipline called Software Engineering ascribed to the 1968 NATO Software Engineering Conference held in Garmisch, Germany. The rationale behind the engineering solution was to develop a set of standards that would regulate the approach to developing software. A methodology called the Waterfall Model, also commonly known as the Classical Software Engineering Paradigm, SDLC (Software Development Life Cycle) was developed. According to Boehm and Turner (2004) the strengths of the SDLC lie in concepts such as *process improvement*, *process capability*, *organizational maturity*, *process group*, and *risk management* each of which will be defined below:

- *Process improvement* –a set of activities designed to improve the performance and maturity of the organization's processes.
- *Process capability* –the natural ability of a process to produce planned results. Process capability leads to process predictability and measurability, which results in effective elimination of causes of poor quality and productivity. Process predictability is a highly debatable concept because some in the methodology fray consider it a limitation instead of a point of strength. Schwaber and Beedle (2002) argue that defined and repeatable processes only work for tackling defined and repeatable problems with defined and repeatable people in defined and repeatable environments, which is not possible in today's fast changing business environments. The essence of the problem is that traditional software engineering is based on repeatability that

was derived from the other engineering disciplines such as civil, electrical and mechanical where repeatable processes are not only possible but are a virtue. However, issues of software development are different in two ways; 1) aesthetically –in the traditional engineering fields for example a draughtsman may design and draw the diagrams for a bridge but he/she will not necessarily build the bridge. Yet in software development it does not work well if the designer is not the programmer (Beck, 1999). 2) application area –the requirements for building a bridge would not normally change, but the business environment for which software is built are always changing.

- *Organizational maturity* –refers to the organization’s ability to apply common standard processes across the organization.
- *Process group* –a collection of specialists that facilitate the definition, maintenance, and improvement of the processes used by an organization.
- *Risk management* –an organized, analytic process to identify uncertainties that might cause harm or loss.
- *Verification and validation* –verification is concerned with technical details of whether the product meets the specification as detailed in the functional and non-functional requirements, in other words the product has been built the right way. Validation is less technical and it aims at ensuring that the product does what the customer wants it to do, in other words the right product has been built (Sommerville, 2004). These two checking and analysis processes generally reveal product defects.

The limitations of the classical software engineering paradigm are rooted in sequential thinking which focuses on the activities presumed to be necessary to produce a result and assumes that one has nothing to learn so that each activity can be completed in sequence, (Larman, 2004 and Hoffer, George and Valacich, 2004). The paradigm therefore tends to be predictive and assumes that business requirements can be accurately predicted. Therefore some techniques have been developed since the early 1980s to try and predict requirements and control their stability, (Davis, 1980). To try and solve some of these problems many methodologies emerged from the

early 1970s up to this age. The emergence of methodologies led to a new challenge of finding the best methodology for a given situation, which is usually called the methodology jungle.

Despite the tremendous growth in the discipline of software engineering, the irony is that the problems that led to the software crisis still remain though in relatively smaller percentages as confirmed in the CHAOS report of the Standish Group International (1994) and the subsequent reports. Details of problems that led to the software crisis are well documented in (Pressman, 2000; Brooks, 1987; Naur, 1996 and Shemer, 1987).

This section summarized literature on the software crisis. The general tone of published research on the software crisis is that the software crisis problems found in the nineteenth century are still the same as those of the twenty-first century. There is in fact a fundamental difference because over the past two decades or so (1980s to 2000s) a lot of maturity in terms of planning and management has been introduced into software engineering from both teaching/training and practice. This kind of growth has resulted in the delivery of high quality software which is also very complex. Therefore maintenance problems of the 1980s are different from maintenance problems of the 2000s hence the original definition of software crisis may not be relevant to the situation of the 2000s. Innovations to the existing software technologies (such the one proposed in this thesis) continue to improve on the quality of software engineering.

3.2 Software Engineering

In general software engineering involves the design, development, maintenance and documentation of software systems. Since the establishment of software engineering as a discipline more than thirty-five years ago, there have been different opinions

about the exact definition of software engineering. These are some of the definitions found in various literature:

1) The Software Engineering Institute at Carnegie Mellon University in their report on Undergraduate Software Engineering Education gives the following definition:

Software engineering is that form of engineering that applies the principles of computer science and mathematics to achieving cost-effective solutions to software problems, (Ford, 1990).

2) The IEEE Computer Society in a joint effort with the ACM agreed on the following definition:

The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software, (IEEE, 1990).

3) A definition that combines these two definitions is given by Pfleeger and Atlee (2005) as follows:

Software engineering can be viewed as an approach that combines computer science (which is made up of theories and computer functions) and the customer (who provides the problem to be solved) to develop a set of tools and techniques to solve problems.

There are a few other views on what the definition for software engineering should or should not include. The three definitions listed here have been chosen because they provide a more balanced view in accordance with the purposes of this research.

The introduction of software engineering as an engineering discipline has with certainty led to formal methodologies for developing software applications and has encouraged logical thinking in programming. However, software engineering has not met levels of predictability that classical engineering disciplines have achieved. This challenge contributes to the continued criticism about calling software engineering an

engineering discipline instead of a branch of computer science or a craft (McBreen, 2001).

3.2.1 The Software Development Problem

This section runs through a brief history of software engineering in order to build a background to the formulation of the questions addressed by this research work. Research shows that the history of software has its beginnings between the early 1950s and the early 1960s (van Vliet, 2003; Cortada, 2002) when software was considered women's job and men preferred the higher prestige hardware engineering roles. Improvement of the efficiency of practitioners through improved technologies and practices remains the fundamental goal. It is believed that the 1968 NATO conference marked the official start of software engineering as a profession, (Mahony, 1998).

The progress in solving software development problems led to the rise of technologies and practices such as *structured programming*, which was followed by *object-oriented programming*, *Computer Aided Software Engineering (CASE) tools*, and *Unified Modeling Language (UML)*.

The current status (2000s) in the software development industry can be classified into three focus areas:

Programming environments: Programming environments have changed from the times of machine language programming which involved the coding of binary digits into the computer, this was done by scientists and researchers. This era was followed by the introduction of assemblers which made programming much easier than the machine language, then came Third Generation Languages (3GLs) such as FORTRAN, COBOL, C et cetera some of which are still in use in the 2000s, and they are concurrently used with Fourth Generation Languages (4GLs) such as SQL, DBase, VB, etc. The future promises to be different because of the emergence of web

services, wireless applications, and distributed computing. Component programming is the future in programming practices. The major development environments that are fighting for supremacy are the Java and Microsoft's .NET environments. Since many applications have been developed in each of the environments, they are most likely going to coexist for some time.

*Software project management*⁵: there is a lot of work being done to master the control and management of software projects through formal project management structures. The need to control projects has given rise to bodies and standards such as; the Project Management Body of Knowledge (PMBok), Project Management Institute (PMI), Capability Maturity Model Integration (CMMI), ISO9000, and Six Sigma among others.

*Software development methodologies*⁶: the fray on software development methodologies is as old as the software engineering profession itself. The major debate of the 2000s is between plan-driven and agile methodologies. The rise of agile methodologies was marked by the establishment of the Agile Alliance in 2001. Agile methodologies are based on four values that define their philosophy. The Agile Alliance (see Appendix A for details) values are summed up in a philosophy that esteems the following: individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan. Agile methodologies have solved to a great extent the software development problems associated with these two issues: 1) Putting value to the way individuals interact in order to deliver the right product. 2) The importance of involving a customer in the development team in order to develop a product that meets the changing needs of the customer. The second point can be considered as a best practice that improves on the quality of the product.

⁵ See definition in section 3.4

⁶ See definition in section 3.5

Software industries in some developing countries focus mainly on acquiring software development methodologies that will lead the organizations to accredited maturity levels with whichever certifying body. The financial expenses associated with acquisition of software development methodologies can be too high for most small to medium enterprises (Enzenhofer and Chroust, 2001; Feldmann and Pizka, 2002) not to mention micro and entrepreneurial organizations that form the majority of the South African software industry. The expenses would include training, hardware upgrades and maintenance.

To conclude this section, the 2000s era is marked by an increase in the number of software development methodologies with agile methodologies being a relatively new addition to the list. Those who apply agile methodologies claim to have realized a marked improvement in the quality of the software product, and reduced time and budget overruns through more formal management systems that take into consideration the importance of people without assuming people to be replaceable mechanical entities of the software development process. The wisdom in leading the software development industry is therefore determined by the ability of an organization to embrace change rather than resist it as more and more software engineering problems are solved by the emerging technologies such as agile.

The software development problem is composed of a complex set of sub-problems. In general engineers and scientists as problem solvers will agree that in order to solve a complex problem, the problem needs to be broken down into smaller problems. Then each sub-problem is solved as a small chunk and this thesis proposes a solution to a smaller part of the software development problem.

The next sections 3.3 to 3.6 give more detail on the major aspects of the software engineering problem as introduced in sections 3.1 to 3.2.

3.3 Software Engineering Process

The notion of a process in developing software refers to a description of activities that must be carried out to realize the intended product. The software engineering process is better defined as a lifecycle and is sometimes called the software development lifecycle, software development process, or the software process. It is characterized by the following phases as described in Table 3.1:

Table 3-1: Phases of the Software Development Lifecycle

	Description
Requirements definition	Get a complete description of the problem to be solved and the requirements posed by and on the environment in which the system will function, (van Vliet, 2003).
System and software design	The problem is broken down into modules and the interfaces between the modules are precisely defined. The focus during design is on what is to be done. The software design level includes data structure definitions, file format definitions, and important algorithm descriptions, (Pressman, 2000).
Implementation and Unit Testing	This phase deals with coding, testing, and debugging of each module designed in the design specification. Code walkthroughs are done to ensure quality and reliability in the implementation, (Pressman, 2000).
Integration and system testing	Testing each module built during coding, then integrating the modules into a single program structure. The whole program is then tested to make sure the modules fit together and perform as designed, (Sommerville, 2004).
Operation and maintenance	Involves fixing any bugs or problems found by the users of the released version. The maintenance team is responsible for releasing minor upgrade versions of the software to accommodate bug fixes, (van Vliet, 2003).

Various methodologies of software development usually define a process that covers in one way or another these five steps. The plan-driven methodologies follow these phases sequentially. The iterative incremental development methodologies implement them concurrently.

3.3.1 Management of the Software Engineering Process

The major purpose of managing the software process is to achieve acceptable levels of risk management in the software development process. It is however important to appreciate that “software development organizations exist to develop software rather than processes” (Fayad, 1997). As part of an effort to improve the quality of software engineering and consequently that of the products being developed a lot of work has been done since the early days of software engineering to improve the software engineering process. To that effect certification standards such the Capability Maturity Model (CMM), ISO 9000, Software Process Improvement Capability dEtermination (SPICE) also called ISO 15504, were established and the software development industry is ever struggling to attain the certifications.

The main challenges as far the certifications are concerned are the inhibitive costs that small companies may not be able to meet. The balance between implementation of these standards and the actual delivery of a quality product remains a mystery that some have tried to solve through theories such as software evolution (Lehman and Belady, 1985 and Lehman and Ramil, 2002) and innovative software technologies such as agile software development technologies (McMunn and Nielsen, 2005). According to Pfleeger and Atlee (2005) IBM started the work on process maturity and many organizations then followed. Of greater value is the development of the CMM by the Software Engineering Institute (SEI) at Carnegie Mellon University. The CMM was followed by other process assessment methods such SPICE.

The trends in issues surrounding the management of the software engineering process have been towards less prescriptive processes such as the ones promoted by the agile movement. Could it be that the highly prescriptive and heavy process assessment methods are on their way out? While some software engineering experts find that there are situations where it would more comfortable to use heavy processes than agile (Highsmith, 2005; Boehm and Turner, 2004), other more pious “agilists” claim that they have not encountered situations where traditional methodologies outperform agile (Poppendieck, 2005). Many success stories where agile development rescued failing projects that were originally developed using the traditional plan-driven methodologies can be found in (Highsmith, 2002a; Cockburn, 2002; Hansson, Dittrich and Randall, 2004). Another very interesting overview of agile successes is found in a report on the experiences of agile development by representatives from many European Union companies (Moonen and van Deursen, 2003).

An interesting observation is that agile methodologies look at the software engineering process and its management from a different perspective to the traditional approach. The principle in agile methods is not to follow the software engineering process sequentially but to follow an iterative incremental approach. As empirical evidence on the success stories of agile development continue to be published we can only but add to the innovations that improve on agile quality.

3.4 Software Development Projects

This section looks at basic definitions of terms that are normally associated with the characterization of software development projects. The position of this section in this thesis is to provide a fundamental understanding of what constitutes a software development project according to this research.

A software development project is a temporary endeavor that is undertaken to create a unique product or service. A software development project in this case can be

defined as a temporary effort to create a unique software product. In this context the term product encompasses any software applications including computer-based systems, embedded software, and problem solving programs that are to be built (Pressman, 2000). According to Schwalbe (2004), a project is characterized by five attributes:

Unique purpose –means that every project should have a well defined objective.

Temporary –a project has a clear-cut beginning and an explicit end.

Requires resources –a project requires resources often from various areas. The resources required include people, hardware, software or other assets.

Primary customer or sponsor –although most projects might have many interested parties or stakeholders, someone must take the primary role of sponsorship.

Involves uncertainty –every project is unique which brings uncertainty in defining the project's objectives, estimate how long it will take to complete, or how much it will cost.

3.4.1 Project Classification

In this thesis characterization of software projects involves the analysis of projects according to certain parameters (project size, development timeframe, complexity, criticality, and customer priorities) found to be sufficient for the purpose of representing a project.

3.4.2 Project Size

According to Eckstein (2004), there are many dimensions that can be used to measure the size of a project. However, the dimensions are interrelated. They belong to two groups. First order dimensions that are a result of requirements and constraints and second order dimensions that are a result of growth of other dimensions. The first order dimensions are *scope*, *time*, and *number of people*. Second order dimensions are *money*, and *risk*. Cockburn (2000) uses a sweeping statement when he defines project size. He defines it simply as the number of people needing to be coordinated on the

project. In this thesis (see section 4.3) the size of a software development project means more than just the number of people and will be defined according to (Eckstein, 2004).

3.4.3 Taxonomy of Software Development Projects

The subject of classification of software development projects usually gets mixed up with the classification of computer programs. It is therefore pertinent to this research work that the difference between these two areas is spelt out. However, the scope of this thesis does not allow a detailed treatment of the taxonomy of software projects. Literature on taxonomy of computer programs shows that computer programs are generally classified according to their functions, complexity, and criticality. Landwehr et al, (1994) provide a classification of computer programs based on security flaws. This classification essentially considers how flaws are introduced into the system, when they entered the system and where they are manifested in the system. Glass and Vessey (1995) argues that in order to match a methodology to an application it is necessary to be able to classify both the methodology and the application. The classification approach given in (Glass and Vessey, 1995) essentially follows qualitative and quantitative approaches. Qualitative being classification by kinds (e.g. oysters or grapes) and quantitative being classification by common dimensions (cows, donkeys, alligators, classified according to number of feet). Lehman and Ramil (2002) describe another program classification technique that groups programs into Static, Practical, and Evolutionary (SPE) types. In this case programs are classified according to the way they are used and they change as the environment in which they operate changes.

The classification of software projects in this research is a more abstract taxonomy that considers the system at a general level without getting to the details of the program. However, for issues such as complexity and criticality of the project it is difficult to classify those without looking at the quantitative nature of the program. In

order to determine a comprehensive set of project parameters that can be used for the taxonomy of software development projects a survey was conducted as reported in section 4.3.

3.5 Software Development Methodologies

There is usually some confusion about the use of the words “method” and “methodology”. Sometimes the words are used interchangeably depending on the subject under discussion. In this research a method is a set of steps that are followed in the software development process. A methodology is a study of a group of related methods for software development. For a more detailed comparative analysis of method and methodology refer to (Cronholm and Ågerfalk, 1999).

The use of methodologies in software development is analyzed by Cockburn (2004a), who in fact divides software development practitioners into three levels and argues that level-one practitioners need a methodology. At level one the practitioners copy the methodology and learn it and they measure their success by determining if the methodology works and how well they can carry it out. Cockburn goes on to say that the level two practitioners would be more concerned with issues of methodology limitations and how to deal with them. At level three Cockburn argues that there is no need for a methodology. At level three the practitioner is assumed to know what works and what does not work. However, Humphrey (1990) argues that there will always be a need for methodologies because even the companies that hire the best graduates from the best computer science schools still face the same problems as everyone else. It is therefore important for everyone concerned about the issues of software development methodologies to understand the need for methodology. Whatever innovations are done around the subject of methodologies should be aimed at developing effective techniques to manage the software development process.

Software development methodologies can be classified into two major groups the *plan-driven group*, which follows a sequential development process. The second group is the *iterative development group*, which breaks down the problem into smaller chunks and develops the system through iterations of each chunk. Cockburn (1997) defines a methodology as the way an organization repeatedly produces and delivers systems. In defining software development methodologies another term of interest is the size of a methodology, which is defined by Cockburn as the number of control elements in the methodology, including deliverables, standards, activities, quality measures, et cetera (Cockburn, 2000). A parameter that is related to the size is the density of a methodology. It is the detail and consistency required in the methodology elements. Greater density corresponds to tighter controls or higher ceremony. Methodology "weight" is its size multiplied by the density, (Cockburn, 2000). To explain further the concept of methodology density let us consider a project whose main aim is to develop a life critical system⁷. In such a system there would be need to ensure that each control element of the methodology is detailed enough to reduce or eliminate the risks that may lead to the failure of the system. For example, if there are any calculations that must be performed by the system they must follow sound scientific techniques that have been proven to work. The need for such detail calls for more resources and time and increases the density hence the size of the methodology.

In this thesis a methodology is defined as all the systematic steps that an organization regularly follows from the inception of a project to the end of a project in order to successfully deliver a system.

3.5.1 Rigorous Methodologies

These are generally considered the traditional way to develop software. They are also known by many other names like plan-driven, (Boehm and Turner, 2004),

⁷ A life critical system refers to a system whose failure may result in the death of people, an example is software used for the control of aircrafts.

documentation driven, heavyweight methodologies, and big design upfront, (Boehm, 2002). These methodologies follow the generic engineering paradigm of requirements, design, build, and maintain. They are based on the notion that the mentioned phases occur sequentially with the input to a phase being the output of the previous phase. These methodologies are also called waterfall-based taking from the classical software development paradigm. The fundamental argument in this research work is that while there is still many software developer companies out there that still use traditional methodologies, the trend is now toward the adoption of iterative incremental development strategies. In fact Poppendieck (2005) stresses that there is less and less reason to use traditional methodologies in software development.

Published research shows that we still have systems that are developed the plan-driven way with success and that there are situations where agile and plan-driven are combined (Boehm and Turner, 2004; Pikkarainen and Passoja, 2005)

3.5.2 Iterative Incremental Development Methodologies

According to Larman (2004) and Martin (1999), iterative and incremental development (IID) methodologies represent software development methodologies that are based on developing the system in small and manageable pieces. Iterative development is an approach to building software in which the overall lifecycle is composed of several iterations that are developed in sequence. The principle behind iterative development is to develop each iteration as a full realization of the system based on an initial set of requirements and follow a software development process until an increment is released. The next set of requirements can then be added to form the next increment and another set of iterations begins. The iterations can be developed concurrently.

From this description of IID it follows that agile methodologies form a subset of IID together with methodologies like the Unified Process (UP), Evo, Pragmatic

programming, and Open Source Programming among others. There are many professionals who critique agile methodologies as unsuitable for the development of large complex software, (Turk, France and Rumpe, 2002 and Maurer and Martel, 2002). Larman and Basili (2003) provide a detailed review of the history of IID showing that IID methodologies form the conceptual basis for agile methodologies. Starting from the 1930s to the 1950s when IID concepts were used to improve quality in non-software projects. Their review shows that the first application of IID concepts to software development was in the early 1960s to a NASA project, (Larman and Basili, 2003). The value of Larman and Basili's review is that IID concepts have been successfully applied to software and other projects.

The focus of this research is on agile methodologies and will therefore give more review of the agile group in the following sections.

3.5.3 Representation of Software Development Methodologies

This section considers the representation of software development methodologies. In the context of this thesis, methodology representation is the classification or characterization of software development methodologies. To represent a methodology there is need to analyze the details of what constitutes a methodology and come up with parameters that can define it in detail. Literature shows that the characterization of methodologies is a rather subjective concept that depends on the purpose for the characterization.

Of interest to this research are the representation frameworks that were developed by Catchpole (1987), Cockburn (2002), Avison and Fitzgerald (2002), Abrahamsson et al (2002), Boehm and Turner (2004), and Cohen et al (2003), in which the subjectivity of methodology characterization is explained in detail.

It is therefore difficult to agree on the actual elements that can be used to represent software development methodologies. Avison and Fitzgerald (2002) emphasize that in fact the choice of the elements is subjective as evidenced by the variations in the number of elements used in different approaches for example Karam and Casselman (1993) have twenty-one elements grouped into three categories; technical elements (has fourteen elements), usage elements (has five elements), and managerial elements (has two elements). The elements that were used in the proposed framework were adapted from Avison and Fitzgerald (2002), Cockburn (2002), Abrahamsson et al (2002), and Cohen, Lindvall, and Costa (2003).

The continued proliferation of software development methodologies has led some to call the problem “methodology wars” and others to call it the “methodology jungle” problem. Since methodologies will continue to grow in numbers because of the need to address various problems in the software development process the selection of the most appropriate set of agile methodology practices for a given project can be considered a nontrivial task, hence the use of terms such as methodology jungle.

In the paragraphs that follow, a review of the various frameworks is given. Cockburn (2002) says that if the development team has experienced and talented people they do not need a methodology in order to deliver a quality system. It can be argued that in order for a team to reach a high level of experience and skill they will have worked with a methodology sometime in their career development. Various organizations and individuals consider methodologies differently, but what can be considered common among various individuals and organizations is the inevitability of a methodology, which means that those who develop nontrivial systems and have no methodology are bound to fail, (Highsmith, 2002a). Avison and Fitzgerald (2002) provide three main categories of justification: a better end product, a better development process and a standardized process.

Cockburn (2002) gives a popular treatment without sufficient referencing on the topic of methodology characterization. The referencing given is biased towards agile methodologies literature. Based on his practical experience he makes sweeping assumptions on issues such as the rationale for a methodology, methodology metrics and elements of a methodology. While his methodology characterization approach is practically biased he assumes that the best way to understand a methodology for the purposes of selection is to view it from the perspective of the methodology designer. The problem with such an approach is that the intended user will be lost in the quagmire of methodology designs as if they wanted to design a methodology when in fact they just want to select one. Cockburn might be given the benefit of the doubt for making sweeping assumptions because having authored many software development methodologies over a long period of time he is definitely an authority in the field. However, there is need for his interpretations to be based on empirical scientific evidence. A review of Cockburn's work shows that he developed a methodology selection framework based on two methodology elements: system criticality and methodology size. The framework is easy to understand, but for practical reasons users may find the approach too simplified to implement in specific situations, for example if cost and development time frame are the main constraints the framework would be difficult to use.

Avison and Fitzgerald (2002) give a scholarly treatment with a lot of referencing. Avison and Fitzgerald provide a detailed review of the history of information systems development methodologies, the trends and future. The approach is informative and academic. They also give an informed review of some literature on various approaches to methodology characterization. They then propose a framework for comparing methodologies based on a list of methodology elements. The focus in the final chapter of the second edition of their book (Avison and Fitzgerald, 1995) is to provide a more general comparison framework for information systems development methodologies.

Based on facts from this approach this thesis aims to develop a framework for the matching of a set of agile software development methodologies practices to a given project. What then becomes of interest is the difference between methodologies for developing an information system (defined below) and those for the specific development of software.

The difference between methodologies for developing information systems and methodologies for developing software does not seem to be something of specific concern to most researchers and practitioners in information systems and software engineering. However, for the purposes of this research it is important to clarify the difference so as to provide a detailed critical analysis of the existing methodologies. To analyze the differences there is need to start by defining an information system and assume the definition for software engineering as given in section 3.2 of this thesis.

An information system includes software, which means that developing an information system will always result in development of software or acquisition of a Commercial–Off–The Shelf (COTS) package of some sort. According to Avison and Fitzgerald (2002), the ‘system’ part of ‘information system’ represents a way of seeing the set of interacting components, such as:

People (for example, analysts and business users)

Objects (for example, computers and office equipment)

Procedures (for example, those suggested in an information systems methodology)

The interaction must take place within a boundary separating those components relevant to the system from those concerned with the environment around the system. Kotonya and Sommerville (2002) define information systems as systems that are primarily concerned with processing information held in a database. Information systems are usually implemented using standard computer hardware (for example,

mainframe computers, workstations, PCs) and are built on top of commercial operating systems.

The Tech Encyclopedia defines an information system as a business application of the computer. It is made up of the database, application programs, manual and machine procedures and encompasses the computer systems that do the processing. The Tech Encyclopedia (Techweb, 2006) further gives the structure of an information system and the functions of each component of the structure as follows:

Database: defines the data structures.

Application programs: data entry updating, querying, and reporting.

Procedures: define data flow.

Methodologies for developing information systems can be viewed from two perspectives, enterprise resource planning, and software development.

The enterprise resource planning approach looks at the entire business system in an organization and aims to provide a complete environment for developing and implementing the information system. In fact enterprise frameworks provide packaged software, rather than proprietary software.

The software development approach looks at the software requirements of an information system and aims to provide the software applications either by implementing pre-packaged applications or developing from scratch. Research shows that these two approaches have been dealt with as separate fields. Literature on information systems development methodologies deals with software development methodologies, which is what Avison and Fitzgerald (2002) are addressing. For a more comprehensive definition of information systems methodologies and a historical review of enterprise resource planning systems see, (Fitzgerald, 1996 and 2002).

Iivari et al (1999) propose a dynamic classification approach that assumes that not all features of an information systems development methodology are equally central. The proposed classification approach is aimed at both academics and practitioners. The fundamental principle of the classification approach is to identify essential methodology features, where essential methodology features are those features that are necessarily abstract and hiding detail and the accidentals are features that are specific to the methodology and may not be generalized to include other methodologies. The proposed framework in Iivari et al (1999) provides concepts for migrating from a discussion of information systems development methodologies as principal units of analysis to a more abstract level. The key feature of the framework is the flexibility of the levels of abstraction described through an algorithmic approach.

The main limitation of this approach is that since it was proposed in 1999 people have continued to teach and research in information systems development methodologies without applying the approach. The reason is that the framework is divorced from the practicalities of applying development methodologies in the industry. It does not answer questions to do with use of specific methodologies and issues of training in certain methodologies. An organization would still have to buy specific methodologies and helping them to identify a group of methodologies may not necessarily solve the “methodology jungle” problem.

Song and Osterweil (1992a) consider the issue of software development methodologies from a design perspective. Reading through Xiping Song’s six papers (Song and Osterweil, 1992a; Song, 1995; Song and Osterweil, 1994 and Song and Osterweil, 1992b, c, 1996) on the subject of methodology comparisons reveals that what they call software design methodologies is in fact software development methodologies. Their comparison approach is based upon formal models of the methodologies and the classifications of methodology parts as types. The approach uses a systematic analysis of the methodologies and is not based on using examples

of software development methodologies. The basis for this comparison approach is derived from the approaches used in other fields of science like biology where the analysis and comparison of animals is based on their organs and their inter-organ relations. Applying such an approach to methodologies involves classifying the parts according to their functions and characterized by their structures, (Song and Osterweil, 1992b, 1992c). This methodology comparison approach is implemented in five steps:

- Building a process model
- Classifying components
- Selecting comparison topics
- Making comparison, and
- Developing process code

The approach is validated by using it to compare many methodologies among them Jackson Systems Development method and Booch Object–Oriented Design method.

The methodology comparison framework proposed by Song and Osterweil is more detailed than any other similar approaches available in literature. The limitation though is that the framework is beneficial mainly to the academic and research communities and leaves out the practitioners. The other limitation of this framework is that it is very outdated because it uses some methodologies such as the Jackson Systems Development method which are no longer in use. When it comes to the software development industry, issues of greater concern to practitioners can be listed as:

- The relevance of a methodology to the project at hand
- How easily a methodology can be tailored to the organization's software development culture
- Costs related to the acquisition of the methodology
- Would the present development team in the organization have the skills necessary to implement the methodology?

- What training requirements would be associated with the methodology?

The following subsection considers the representation approaches that are more specific to agile methodologies in order to give a detailed review of agile methodology representation issues. It is under agile methodology representation issues that some gaps exist, the solution of which is provided in Chapter Four.

3.5.4 Representation of Agile Methodologies

Schuh (2004) summarizes agile development by stating that in order to understand what characterizes agile methodologies it is important to realize that the agile practices are not new, what is different and original about the agile approach is that the Agile Alliance has published these practices, fused them with core values about people and project environments and stated the way to build software better. Such a view assumes that people behind agile practices have provided proof that these practices are better than the traditional practices. However, the lack of this proof is one of the major limitations of agile development cited by other authors (Cohen, Lindvall and Costa, 2003; Abrahamsson et al, 2002; Lindvall et al, 2002; Turk, France and Rumpe, 2002 and Maurer and Martel, 2002) all show that most of the evidence available is only anecdotal. It is however, important to note that more recent research work (2004 to 2005) reveals more and more empirical evidence confirming success in the use agile practices, (Eckstein, 2004; Highsmith, 2004, and Cockburn, 2004a).

Armour (2004) represents agile methodologies using a combination of critical attributes and defining attributes as follows:

Change in requirements is welcome and accepted as a norm and used to improve the quality of the product according to the needs of the customer.

Feedback management in essence defines the feedback from each activity in the development process as a learning process implemented through the application of some methods of determining the correctness of the work done.

Iterative development is done in understandable short units and not in a monolithic unit. This approach results in the system being developed in a flexible way with respect to design, schedule, and functionality. The approach also allows for refactoring⁸, which is a window into the embracing of new knowledge.

Development is a human activity; this means that the development process becomes a humanistic process and not a mechanistic process.

Customer centric, the customer is considered to be part of the development team instead of consulting the customer as an outsider who injects unwanted changes into the system.

Event-driven, development is more driven by what an individual learns as opposed to what an individual is expected to learn.

Boehm and Turner (2004) conducted extensive research on the issues of effectively mixing agile and plan-driven methodologies and using risk analysis to provide the balance between these two groups of methodologies. In their book (Boehm and Turner, 2004) they give many cases of projects that have successfully combined these groups of methodologies with success. Two cases are given to illustrate the use practices from both agile and plan-driven methodologies. The details of the cases are originally published in (Elssamadisy and Schalliol, 2002; and Royce, 1998).

After proving that agile and plan-driven methodologies can be and have been successfully mixed Boehm and Turner (2004) argues that more details of how the mixing can be done are required and they describe a method that is based on risk analysis to do the mixing. They use case studies again to illustrate the effectiveness of their technique, see (Anthes, 2003) for details of the cases.

⁸ Refactoring is the process of changing a software system in a way that does not alter the external behavior of the code, yet improves its internal structure (Fowler, 1999).

Boehm and Turner (2004) conclude by describing a detailed framework to represent and compare agile methodologies using the following framework:

Each methodology is represented by three profile ratings namely: *levels of concern*, *lifecycle coverage*, and *sources of constraint*.

This framework is in fact based on a representation model in (Abrahamsson et al, 2002). Each of the profile ratings can be described as follows:

Levels of concern –represents the levels of concern that each methodology raises about the business enterprise, business system, multi-team project, and individuals.

Lifecycle coverage –defines which phase of the lifecycle (concept development, requirements, design, development, and maintenance) the methodology covers.

Sources of constraint –represents what constraint each methodology puts on the management process, technical practices, risk/opportunity, measurement practices, and customer interface.

The conclusion that can be broadly derived from the analysis by Boehm and Turner (2004) is that agile processes are more relevant in the development of systems that are not critical and not complex. Such a conclusion was highly criticized by most agile proponents who participated in the first electronic workshop on agile methodologies see Lindvall et al (2002). The counter argument against this conclusion was that with proper modeling as described in the Agile Modeling methodology it would be possible to use agile methods on critical and complex projects.

Abrahamsson et al (2002) gives a detailed report on the research work conducted at the VTT Technical Research centre on the application of agile methodologies in embedded systems. The report focuses on the review of literature on agile methodologies and describes a technique for classifying agile methodologies. They start by giving a working definition of agile methodologies followed by a detailed description of some of the most common agile methodologies then (i.e. in 2002). The

methodologies are described according to a set of elements. It is not clear how the elements are arrived at though the choice of the elements seems feasible when compared with the elements used in the methodology selection framework by Avison and Fitzgerald (1995).

Each of the elements is described in Abrahamsson et al (2002) as follows:

Process –means the lifecycle, roles and responsibilities of the team members.

Role and Responsibilities –the roles assumed by different members of the development team and the specific duties that they have to do.

Practices –means specific activities and work products of the methodology.

Adoption and experience –defines existing reports of use by practitioners and how to adopt the methodology.

Scope of use –defines limitations of the methodology, and

Current research –defines the current research work on the methodology.

In general the analysis based on this framework revealed a few gaps in agile methodologies such a lack of empirical evidence on the benefits of using agile processes and a lack of comprehensive representation frameworks for agile processes. Abrahamsson et al (2002)'s representation structure constitutes a baseline used in most literature on characterization of agile methodologies. This approach will be adapted in chapter four to build a methodology selection framework. Abrahamsson et al (2002)'s set of elements were compared with Avison and Fitzgerald (1995)'s set of elements and a more comprehensive set was built.

Cohen, Lindvall, and Costa (2003) developed a methodology comparison structure based on Abrahamsson et al (2002). This structure compares a few of the agile methodologies based on the following characteristics:

Team size –represents the number of people, which the methodology can handle in a project.

Iteration length –represents the time it takes to deliver the first usable bit of the system.

Distributed support –how the methodology supports teams in various geographical locations.

System criticality –defines the methodology’s support for mission critical systems.

The next section delves into more detail on agile methodologies and specifically defines and analyzes some of the most commonly used agile methodologies. The purpose of this section is to give an overview of some of the most commonly used agile methodologies. Such an analysis will give the necessary background for the selection matrixes used in Chapter Four.

3.6 Agile Methodologies

This section starts by defining agile methodologies and gives an overview of some of the most commonly used agile methodologies in order to reveal the philosophy and fundamental approach that has been introduced by agile practices.

Definition

The term “agile” carries with it connotations of flexibility, nimbleness, readiness for motion, activity, dexterity in motion and adjustability, (Abrahamsson et al, 2002). According to Beck (1999) agile methodologies are a lightweight, efficient, low-risk, flexible, predictable, scientific and fun way to develop software. Lightweight implies minimizing everything that has to be done in the development process, e.g. documentation, requirements, et cetera. Efficient implies doing only that work that will deliver the desired product with as little overhead as practically possible. Low-risk implies trading on the practical lines and leaving the unknown until it is known. Predictable implies that agile methodologies are based on what practitioners do all the time. Scientific means that the methodologies are based on sound scientific principles.

This group of software development methodologies was given the name ‘agile’ when a group of software development practitioners met and formed the Agile Alliance in February 2001. The agile movement could mark the emergence of a new engineering discipline that has shifted the values of the software development process from the mechanistic to the organic. Mnkandla and Dwolatzky (2004a, and b) call it Agile Software Engineering (ASE). Calling it “Agile Engineering” without the software part could confuse many as the term ‘agile’ has become a buzzword even in other disciplines that are outside software development. Schuh (2004) defines agile development as a counter movement to thirty years of increasingly heavy-handed process meant to refashion computer programming into software engineering, rendering it as manageable and predictable as any other engineering discipline.

Presently there is work going on in an effort to fully define what agile methodologies are, hence various researchers look at agile methodologies in various ways. In this thesis the term ‘Agile Methodologies’ will follow a definition agreed upon by Agilists at the first eWorkshop on agile methodologies in June 2002. Lindvall et al (2002) summarized the working definition of agile methodologies as a group of software development processes that are iterative, incremental, self-organizing and emergent:

Iterative, i.e. delivers a full system at the very beginning and then changes the functionality of each subsystem with each new release.

Incremental, i.e. the system as specified in the requirements is partitioned into small subsystems by functionality and a new functionality is added with each new release.

Self-organizing, i.e. the team has the autonomy to organize itself to best complete the work items.

Emergent, i.e. technology and requirements are “allowed” to emerge through the product development cycle. The details of the Agile Manifesto and the principles of

agile software development are given in Appendix A. Figure 3.2 sums up the definition of agile methodologies.

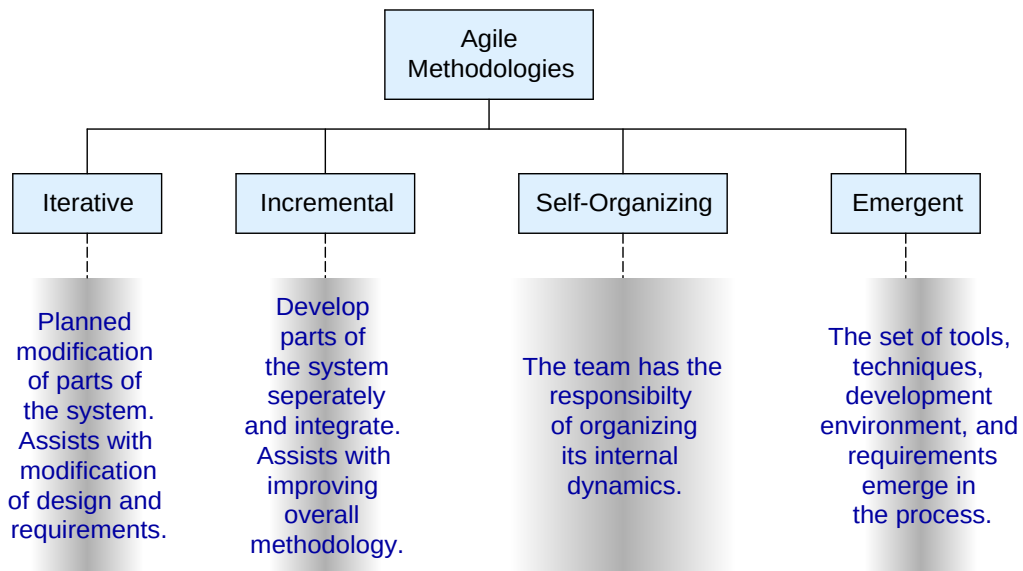


Figure 3-2: Definition of Agility

On a practical perspective agile methodologies emerged from a common discovery among practitioners that their practice had slowly drifted away from the traditional heavy document and process centered development approaches to more people-centered and less document-driven approaches, (Boehm and Turner, 2004; Highsmith 2002b and Fowler, 2002). There is a general misconception that there is no planning or there is little planning in agile processes. This is because the Agile Manifesto lists as one of its four values the preference for responding to change over following a plan, (Agile Alliance, 2001). In fact planning in agile projects could be more precise than in traditional processes, it is done rigorously for each increment and from a project planning perspective agile methodologies provide a risk mitigation approach where the most important principle of agile planning is feedback. Collins-Cope (2002a) lists the potential risks as: risks of misunderstandings in functional requirements; risks of a deeply flawed architecture; risks of an unacceptable user

interface; risks of wrong analysis and design models; risks of the team not understanding the chosen technology et cetera. Feedback is obtained by creating a working version of the system at regular intervals or per increment according to the earlier planning effort, (Collins–Cope, 2002b).

Besides dealing with the most pertinent risks of software development through incremental development, agile methodologies attack the premise that plans, designs, architectures, and requirements are predictable and can therefore be stabilized. Agile methodologies also attack the premise that processes are repeatable (Highsmith, 2001 and Schwaber and Beedle, 2002). These two premises are part of the fundamental principles on which traditional methodologies are built, and they also happen to be the main limitations of the traditional methodologies.

Boehm and Turner (2004) view agile methodologies as a challenge to the mainstream software development community that presents a counter–culture movement which addresses change from a radically different perspective. All agile methodologies follow the four values and twelve principles as outlined in the Agile Manifesto. Table 3.2 maps principles and practices to corresponding agile values.

Table 3-2: Agile Values Principles and Practices

	Values	Principles	Practices
1	Individuals and interactions over processes and tools	Build projects around motivated individuals. Best results emerge from self–organizing teams. Team reflects regularly where and how to improve.	Tacit knowledge. Risk lists.
2	Working software over comprehensive documentation	Measure success only through working software. Continuous attention to technical excellence	Incremental development. Iterative development. Running software.

		and fine design. Simplicity is essential.	
3	Customer collaboration over contract negotiation	Business people and developers work together daily throughout the project. Place emphasis on face-to-face communication.	Customer collaboration. Face-to-face meetings.
4	Responding to change over following a plan	Keep delivery cycles short. Satisfy customer through early and frequent delivery. Welcome changing requirements even late in the project. Promote sustainable development pace.	Configuration management. Frequent releases.

The next few sub-sections give a brief analysis of some of the most commonly used agile methodologies namely:

- Adaptive Software Development
- Crystal
- Feature Driven Development
- Agile Model Driven Development
- ICONIX Process
- Scrum
- Dynamic System Development Methodology
- Lean Development
- Extreme Programming

The analysis is based on eight elements adapted from (Abrahamsson et al, 2002; Avison and Fitzgerald, 2002; Cockburn, 2002 and Cohen, Lindvall, and Costa, 2003). An understanding of these elements should give adequate knowledge about a

methodology. An analysis of these elements in comparison with the elements that define a software development project should result in the selection of the most appropriate methodology for a given project. However, this section is not intended to give an exhaustive description of agile methodologies, a more exhaustive understanding of agile methodologies can be found in (Mnkandla and Dwolatzky, 2004a and b; Highsmith, 2002a and c; Cockburn, 2002; Boehm and Turner, 2004; Abrahamsson et al, 2002 and Rosenberg, Collins–Cope, and Stephens, 2005). The elements are briefly defined as follows, (Avison and Fitzgerald, 2002):

Methodology philosophy: This is a set of principles underlying the methodology. This includes the practices of the methodology that spell out issues such as paradigm – which means a specific way of reasoning about problems, objectives –a statement of what problem area the methodology designer intended it to solve. For example, development of a computerized information system or ascertaining the need for an information system.

Model: How the methodology views reality. The model may be reflected in the lifecycle or development process of the methodology, (Abrahamsson, et al 2002). The model provides a means of communications, captures the essence of a problem or a design, and gives insight into the problem area.

Techniques and tools: Identification of the techniques and tools used by the methodology. It should be noted that a methodology might not necessarily specify techniques and tools.

Methodology scope: The level of detail to which the methodology's development framework is spelt out. Of particular importance are the limitations of the methodology, (Abrahamsson et al, 2002). According to Avison and Fitzgerald (2002) the scope is defined in the life cycle of the methodology. In this research the scope will be understood in the context of spelling out the limitations of a methodology.

Methodology outputs: Deliverables to be expected from each specified phase of the methodology. This usually provides some guidance that analysts can use to evaluate their progress.

Adoption and experience: This includes a clear description of the origins of the methodology whether it was developed from practice –commercial background or from theory –academic background. Another point about adoption and experience is an indication of the number of organizations that are using the methodology i.e. user base, very difficult to determine. The types of users of the methodology would also fall under this element, i.e. is it used by developers or management? Associated with this notion are the skill levels that are required to use the methodology, (Abrahamsson et al, 2002).

Methodology product: this element describes what somebody gets if they buy the methodology. This usually ranges from software to documents such as manuals, books, online access to all kinds of help and training. The product normally changes because of the changes in technology and available tools.

Roles and responsibilities: refer to the allocation of specific roles through which the software production in a development team is carried out. This element essentially outlines who does what in the project team, (Abrahamsson et al, 2002).

3.6.1 Adaptive Software Development (ASD)

The structure of ASD is illustrated in figure 3.3.

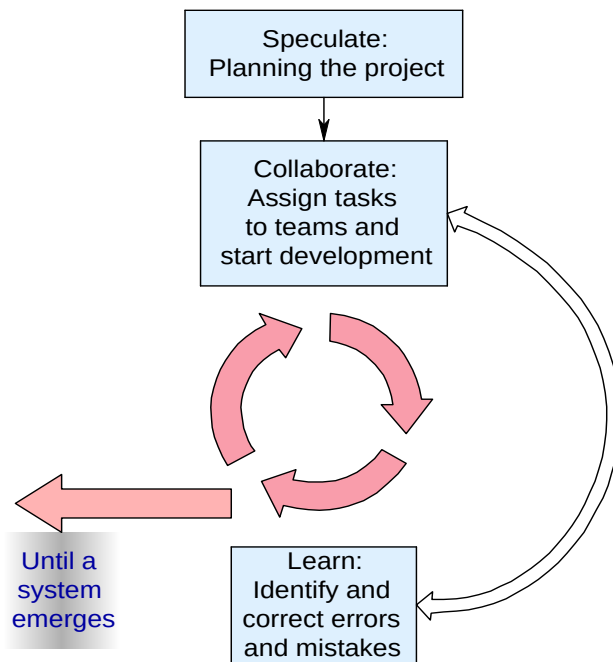


Figure 3-3: An Overview of Adaptive Software Development

Philosophy: the ASD philosophy is rooted in five primary goals of software management as stated in (Highsmith, 2000c and 2004): *adaptive culture*, which views organizations as complex adaptive systems, and creates emergent order out of a team of individuals. The second goal is *adaptive frameworks*, which offers a series of frameworks or models to help an organization employ adaptive principles. The third goal is *adaptive collaboration*, which establishes collaboration, i.e., the interaction of people with similar and sometimes dissimilar interests to jointly create and innovate, which becomes the organizational vehicle for generating emergent solutions to product development problems. Collaboration is addressed according to interpersonal, cultural, and structural relationships. The fourth goal is *adaptive scale*, which provides a path for organizations to adapt the approach on larger projects. The

fifth goal is *adaptive management*, which replaces the culture of command–control management with a flexible management style that provides for wider participation in decision–making, and empowerment. This in fact means that "leadership" replaces "command" and "collaboration" replaces "control."

Model: Based on the adaptive development life cycle, it is a model that supports the concepts and provides a practical way of moving from the conceptual to the actionable. The phases of the lifecycle are; *speculate*: which involves the setting of the initial project feasibility study. These are all estimated and flexible to allow for changes later. *Collaborate*: all members of the project team come together to share their own understanding of the system, this sets a stage for the organization to create emergent order. *Learn*: this is where the assumptions made about the iteration are challenged and reviewed to determine the way forward for the next iteration. Project initiation leads to Adaptive Cycle planning which leads to concurrent component engineering followed by the iterative quality review and then final quality assurance and release, (Highsmith, 2000c).

Techniques and Tools: ASD applies the old technique of Joint Application Development (JAD) sessions. Planning is based on results it is also known as component–based planning, which plans based on a group of features (or deliverables). Spreadsheets are used for component–based planning, (Highsmith, 2000b). Customer focus group reviews is another technique that is very useful for planning. Another technique is time–boxing which is the setting of fixed delivery times for iterative cycles and projects; this technique is in fact about focusing and re–evaluating the project’s mission, (Highsmith, 2000b).

Scope: ASD is essentially a management philosophy for agile projects and therefore limited to project management activities. The limits of this methodology when it comes to team sizes depends on the size of the project, but like any other agile methodology the level of agility decreases as the team gets larger. The methodologies

of communication determine the rigor in a project and the support for distributed development, (Highsmith, 2000c).

Outputs: there are several outputs at various phases of ASD. The speculate phase delivers a list of features which are a result of intense planning with estimates on resources. The collaborate phase delivers working components for the quality review. The Final product is a quality reviewed piece of software that is ready for signing off, (Highsmith, 2000c).

Adoption and Experience: As stated in (Highsmith, 2000c), Jim Highsmith has so far participated in more than one hundred projects that were using some form of ASD over the last decade. These projects span the range of internal IT applications, to commercial software products, to embedded software in medical instruments. The ideas of ASD have influenced the thinking of aerospace engineering at MIT and the Lean Aerospace movement in the US.

Methodology Product: When Jim was contacted (Highsmith, 2005), about the methodology product he said there was no product, but some have used his ASD books (Highsmith 2000c and 2004), and he often provides presentations, tutorials and training on the use of ASD.

Roles & Responsibilities: The methodology has no comprehensive list of responsibilities, but mention is made of the following; executive sponsor as person with overall responsibility for the product, participants in joint application development sessions, the developer, and the customer.

3.6.2 Crystal

The overview of crystal methodologies is shown on figure 3.4.

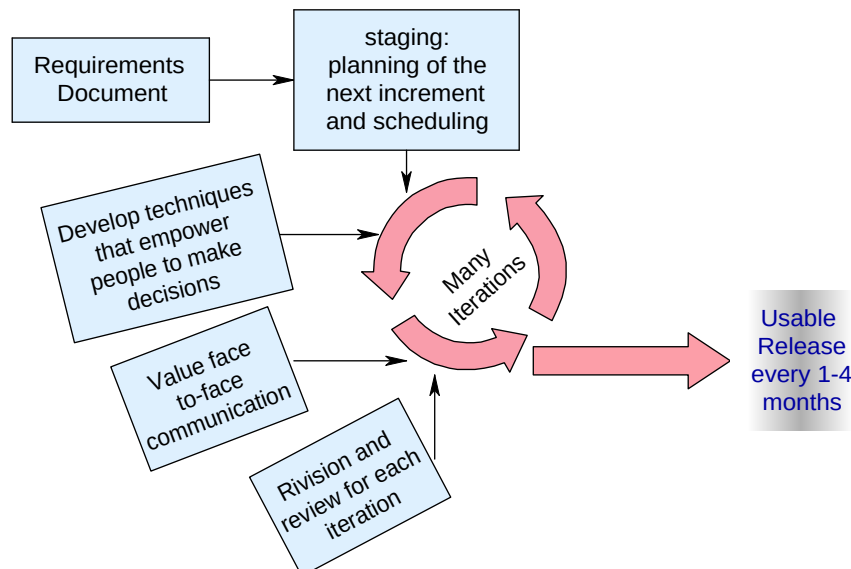


Figure 3-4: Crystal Overview

Philosophy: The crystal philosophy can be summed up in three fundamental principles; the first is a human-powered and adaptive principle, which means that the project success is based on empowering the people involved to make decisions and enhance their work, i.e. think about people, their skills, their communications and their interactions, (Cockburn, 2004a). The second is ultra-light, which means that for whatever project size and priorities, documentation and any paperwork must be kept minimal, in other words keep deliverables, and reviews to a very small number, and writing standards must be very light. The third principle is the “stretch –to–fit” (this may appear as “shrink–to–fit” in some literature, which according to (Alistair, 2004b), has since changed to “stretch –to–fit”), the principle being that it is better to start with something too light and add as needed (rather than starting with something too heavy and trying to remove), (Cockburn, 2004a).

Model: According to Cockburn (1997), Crystal Orange's process can be illustrated by the activities of one increment which are; starting from a requirements document, next comes staging which is the planning of the next increment and scheduling, next is the revision and review of each of the several iterations (each iteration has the following activities: construction, demonstration and review) in an increment. The three factors that are central to the Crystal process are according to Highsmith (2002a); *communication load* –the idea is that as the number of people in the project team increases the ease of communication is compromised. *System criticality* –as system criticality increases the size of the team also increases and the communication becomes less clear which defines another colour in the Crystal family of methodologies. *Project priorities* –here the emphasis is on the way of working, which is fashioned by the priorities of a project such as; time to market, cost reduction, exploration, or legal liability.

Techniques and Tools: the Crystal family in general does not prescribe particular techniques but emphasizes use of any techniques from the team's experience. However, Crystal Clear lists the following tools: Compiler, versioning and configuration–management tool and Printing Whiteboards, (Cockburn, 2002).

For Crystal Orange: Versioning tools, programming tools, testing tools, communication tools, project tracking tools, drawing tools and performance measuring tools. Screen drivers are needed also for repeated GUI tests. There is mention of methodology tuning techniques and use of reflection workshops, (Cockburn, 2002).

Scope: the Crystal family of methodologies is essentially a project management philosophy that defines projects according to team sizes. It is rather difficult to spell out the scope of Crystal because the methodology provides a basis for selecting and tuning other methodologies. In Crystal Clear the team size is up to six developers. In Crystal Orange the team size is from ten up to forty developers. Crystal methodologies are not suitable for life–critical systems, (Highsmith, 2002a).

Outputs: Crystal Clear and Crystal Orange include the following outputs; release sequence, common object models, user manuals, test cases and migration code. Crystal Clear has the following additional outputs; annotated use case/feature descriptions, user viewings and deliveries document, if needed screen drafts and design sketches and notes can be delivered. Crystal Orange has the following additional outputs; requirements document, extensive schedule of the project, user interface design document, inter-team specifications, and status reports, (Cockburn, 2002).

Adoption and Experience: Crystal Orange principles have been used in one documented project, a two-year project called ‘Winifred’, (Cockburn, 1997). Alistair (2004b) mentions the use of Crystal in the following projects:

- “In the mid–1990s, a fixed–price project involving fifty people, between IBM and Ralston Purina, this was a Crystal Orange project, (Cockburn, 1997).
- In the late 1990s, the Central Bank's side of the NICS–NBO project at the Central Bank of Norway, which matches all the bank–to–bank transactions in Norway against the banks' floating cash balances held by the Central Bank (technology was COBOL/assembler/CICS/ on mainframes).... this started at Crystal Clear and moved to what would have been a Crystal Yellow, (Cockburn, 1997).
- Ongoing since 1998, an insurance company project in Munich, about eight people developing a new insurance products framework (technology started in Smalltalk, moved to Java, integration with mainframe) this was a Crystal Clear, (Highsmith, 2002a).
- Around 2003 a series of short six–month projects in Salt Lake City using Java, they were Crystal Clear.
- Around 2003 a start–up company with two programmers, using it over one to two year periods, this was Crystal Clear.

- 2003, a small team in Schlumberger used Crystal Clear for a system replacement project, ten people.
- In 2003 to 2004 the software section of a project in France to build a new post office building to handle all the mail going into and out of northern France, fixed-price contract with the French Post Office, thirty people, using Java, this was Crystal Yellow”.

Methodology Product: The main product that can be purchased is the Crystal book Cockburn (2004b) and the other support texts are: (Cockburn, 1997 and 2002 and Highsmith, 2002a). There is a website that provides online help (<http://alistair.cockburn.us/>) and some help can also be obtained by subscribing to the crystalclear@yahoo.com group.

Roles & Responsibilities: Crystal Clear has only one team and Crystal Orange has several teams. The roles are: *sponsor* finances the project and delivers the mission statement, *senior designer* maintains team structure, implements methodology, designs the system, *designer-programmer* creates screen drafts, design sketches and notes, common object model, source code, packaged system, migration code, and test cases, and *user* helps with use case and screen drafts. The rest of the roles may be merged or come from the previous list; *business expert* may come from the sponsor, the user or the senior designer, *coordinator* may come from the designers, *tester* may come from designers, and *writer* may come from designers, (Abrahamsson et al, 2002).

Crystal Orange has the following additional roles arranged into teams: system planning, project mentoring, architecture, technology, functions, infrastructure and external test teams, (Cockburn, 2002).

3.6.3 Feature Driven Development (FDD)

The structural overview of FDD is illustrated in figure 3.5.

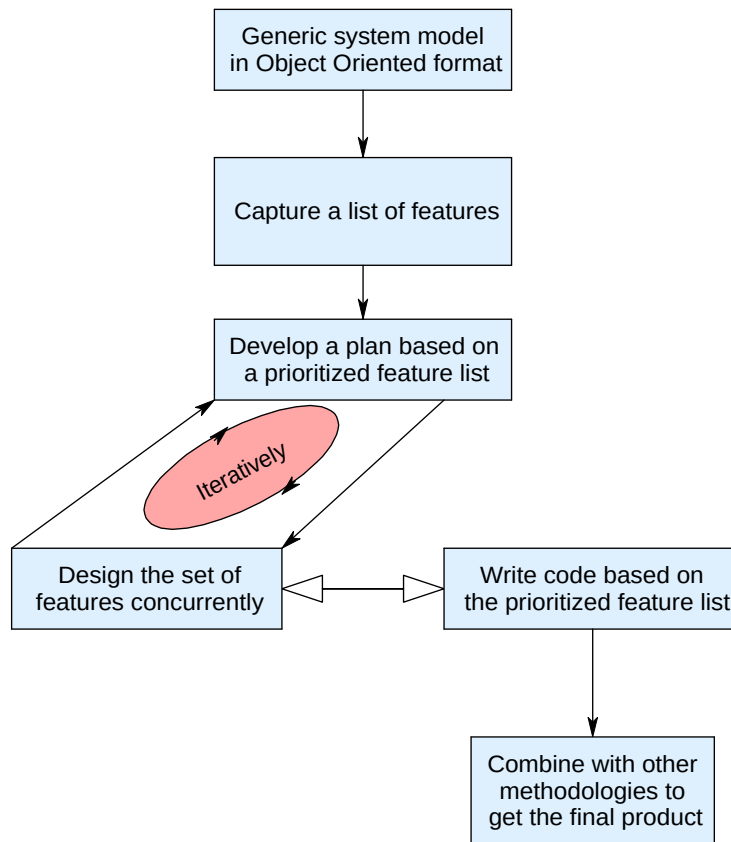


Figure 3-5: Overview of Feature Driven Development

Philosophy: Centered on Object Oriented (OO) modeling of a project to represent it in the simplest possible model that graphically captures the realities of the project. The immediate benefit of this is apparent to the programmer in the form of the short iterative feature-based planning and design process. The manager also benefits from the vivid picture of the project progress and status, which makes steering of the project easier, (Palmer and Felsing, 2002). FDD only deals with design and implementation and has to be combined with other agile processes.

Model: Starts from an overall model (domain object model) from which a features list is built through analysis of client valued functions for each domain area, (Abrahamsson et al, 2002). Then the planning is based on these features (UML

sequence diagrams). The final phase is the iterative design and build by feature (Detailed UML class diagrams) which results in the main build. An iteration of a feature would normally take one to three weeks, (Palmer and Felsing, 2002).

Techniques and Tools: The strength of FDD is in its modeling techniques that are based on Object Oriented Analysis and Design. Detailed UML class diagrams and UML sequence diagrams combine with short iterative agile techniques to provide system development advantage to the development team, (Abrahamsson et al, 2002).

Scope: FDD is limited to small to medium sized teams (four to twenty people). The methodology deals with uncertain requirements and centers on Object Oriented modeling, (Abrahamsson et al, 2002).

Outputs: The initial output of the methodology is a comprehensive feature list that is produced from the analysis of object models. From the feature list an overall model is then produced during the planning by feature phase, which is given to class owners (individual developers). Design and build by feature results in the final code, (Abrahamsson et al, 2002).

Adoption and Experience: First used for a large complex banking project, Palmer and Felsing (2002). Gradual adoption recommended, (Abrahamsson et al, 2002). FDD is highly recommended for use in OO projects though reports on its use are not widely published so far.

Methodology Product: The only available product for the methodology is the FDD book, (Palmer and Felsing, 2002).

Roles & Responsibilities: Three groups of roles: key, supporting and additional, (Palmer and Felsing, 2002). Project Manager is the administrator and financial leader. Chief Architect is a chief designer and runs workshops on design sessions. Developer

Manager runs daily development activities and solves problems in development. Chief Programmer is an experienced developer who participates in requirements, design and analysis. Class Owner design, code tests and documents the system under supervision of the Chief Programmer. Domain expert has knowledge of how various requirements should be implemented. Domain Manager leads domain experts with regard to different opinions on requirements. Release Manager controls progress reports. Language Lawyer is an expert in a specific programming language or technology. Build Engineer sets up, maintains, and runs the build process. Toolsmith builds small tools for development. System Administrator configures, manages, troubleshoots servers, workstations and development and testing environments. Tester verifies if the system meets requirements. Deployer converts existing data to new system format. Technical writer writes user documentation, (Abrahamsson et al, 2002).

3.6.4 Agile Model Driven Development (AMDD)

The structural overview of AMDD is shown in figure 3.6.

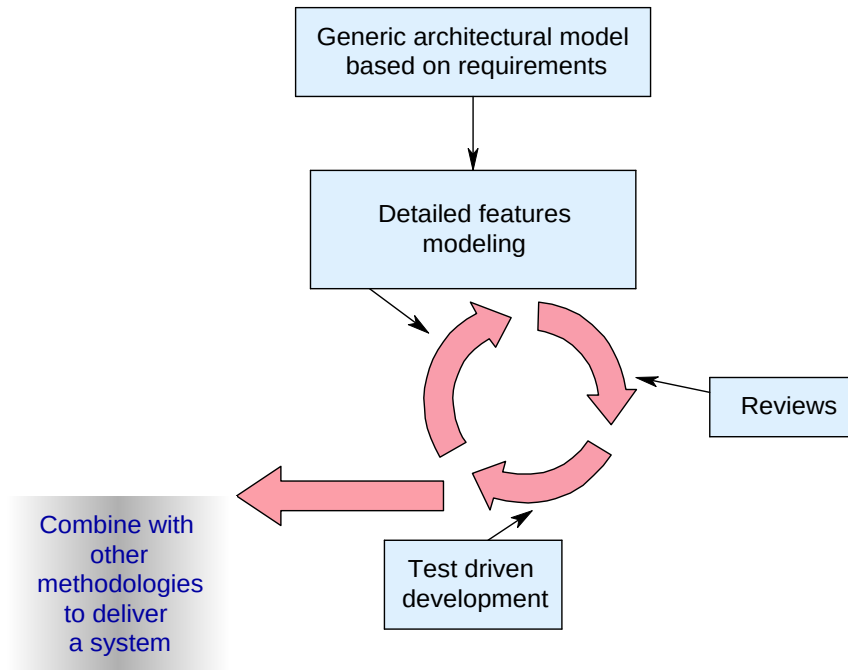


Figure 3-6: AMDD Graphical Overview

Philosophy: Focuses on working software as the main goal of modeling. The underlying principle is to keep the amount of models and documentation minimal but advanced enough to support meticulous design, (Ambler, 2002b). The principle in AMDD is that whatever has to be done to model the problem space or the solution must be as minimal as possible without compromising value as perceived by the stakeholders. Simplicity and communication are virtues in AMDD. Agile Model Driven Development provides a modeling framework for scaling up of agile methodologies to deal with large projects.

Model: AMDD is not a complete methodology but a modeling framework, hence it does not prescribe a process but works well with other agile methodologies.

Techniques and Tools: No specific tools are mentioned except that there must be use of simplest available tools, (Ambler, 2002a).

Scope: No specific team size is mentioned but the methodology aims for small teams. The AMDD framework can be combined with all non-modeling agile methodologies, (Ambler, 2002a).

Outputs: The methodology delivers minimal UML use case models and class diagrams. Though the principle is to travel light on documentation the agility in this methodology lies in the ability to deliver the documents as requested by the customer.

Adoption and Experience: It is quite new (introduced in 2002), there is a lot that is not yet clear about the methodology. The AMDD website (www.agilemodeling.com) outlines some activities and research going on in agile modeling.

Methodology Product: There is an agile modeling book, Ambler (2002a) and a lot of online papers at the AMDD website (www.agilemodeling.com). AMDD should be combined with other methodologies for a complete development process.

Roles & Responsibilities: AMDD teams are expected to come from developers and project stakeholders, Ambler (2002a). Teams should be composed of self-motivated hard working developers. The modeling must be done in teams where everyone must participate.

3.6.5 ICONIX

The overview of ICONIX is illustrated in figure 3.7.

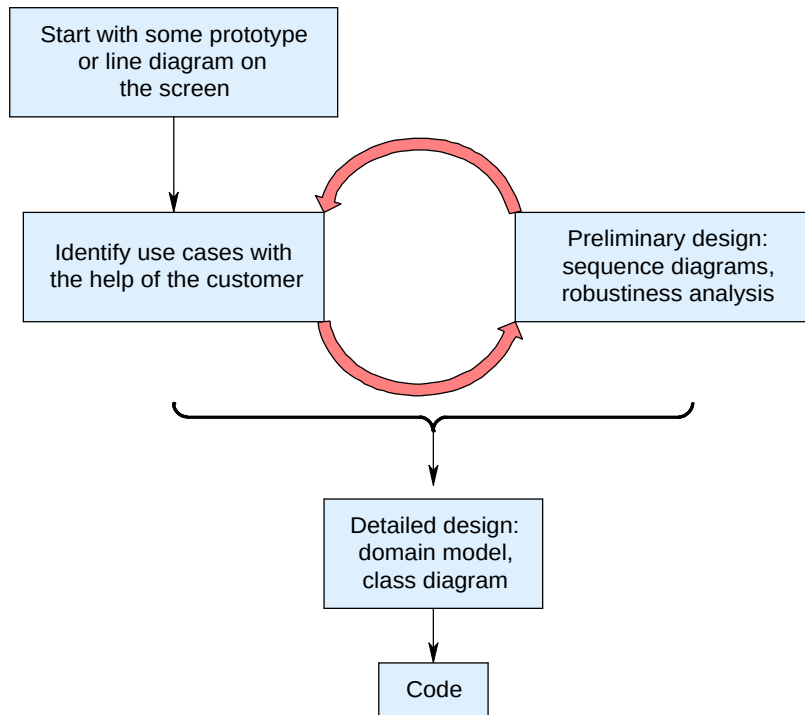


Figure 3-7: ICONIX Graphical Overview

Philosophy: The ICONIX Unified Modeling approach's philosophy was derived from a combination of the most commonly used Object Oriented modeling techniques; Booch, Object Modeling Technique (OMT) and Object Oriented Software Engineering (OOSE). It centers on building fine object models through streamlining UML diagrams to a minimal but sufficient set. Techniques are then applied to provide a quick, efficient, iterative and incremental move from use cases to code. In fact the ICONIX philosophy is based on closing the gap between use cases and sequence diagrams (detailed design), (Rosenberg and Scott, 1999).

Model: The ICONIX process starts from prototypes or line diagrams of the screens, after user assurance of the prototype, followed by identification of use cases on use case diagrams, then comes the writing of use case text, next comes the refinement of text in robustness analysis, which leads to detailed design on sequence diagrams,

followed by breakdown of the system along use case boundaries to derive a detailed object model, (Rosenberg and Scott, 2000).

Techniques and Tools: UML diagrams, Use cases, Object Oriented Analysis and Design (OOAD) Techniques, Prototyping tools and techniques, Iterative/incremental development, Training tutorials.

Scope: ICONIX works better with small to medium sized teams (four to twenty people) but it can be scaled to larger teams. Emphasizes the analysis and design part of the Software Development Lifecycle especially how and what should be done to get code from use cases, (Rosenberg and Scott, 2000).

Outputs: The code delivered is in essence a prototype of the proposed system. A high level version of the given use case model.

Adoption and Experience: Based on UML and other OOAD methodologies that have been in use since the early 1990s, (Rosenberg and Scott, 2000). In many organizations where ICONIX is in use it is applied as tool for quickly changing use cases to code.

Methodology Product: The methodology is sold in the form of CD-ROMs. Online help is also available including tutorials and training and the book, (Rosenberg and Scott, 2000 and Rosenberg, Collins-Cope, and Stephens, 2005).

Roles & Responsibilities: Programmer writes the code. Trainer trains the project team on OOAD and ICONIX. Mentor is a fulltime onsite consultant. Analyst writes use cases. Designer changes use cases into design diagrams. User provides the requirements and reviews the design, (Rosenberg and Scott, 2000).

3.6.6 Scrum

The structure of Scrum is illustrated in figure 3.8.

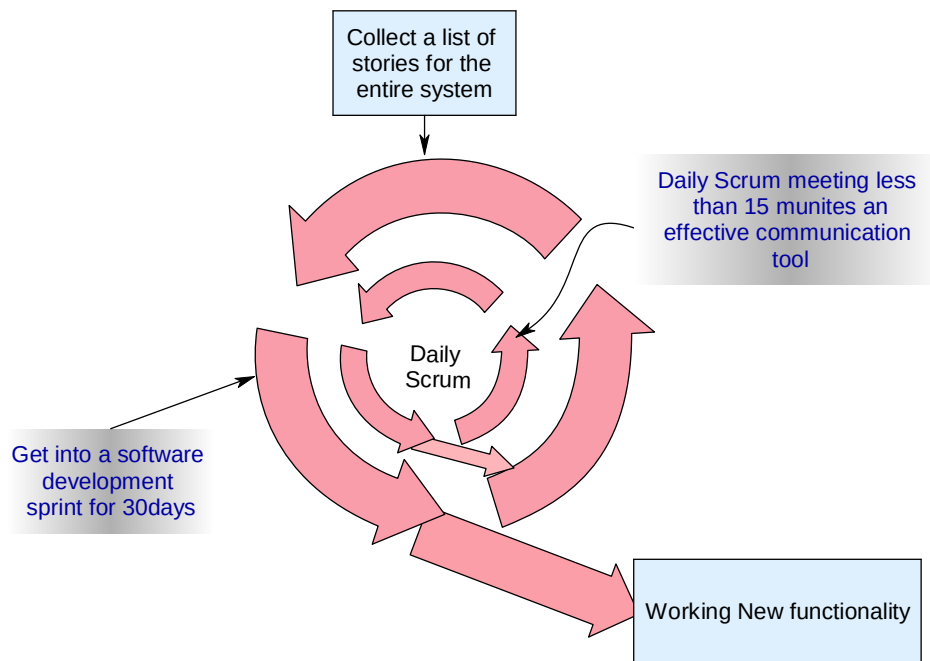


Figure 3-8: Scrum Graphical Overview

Philosophy: According to Schwaber and Beedle (2002), the Scrum philosophy is based on the empirical model of industrial process control used to control complex and unpredictable systems. The fundamental notion of Scrum is that the set of variables that constitute software development such as people, requirements, resources, technology etc are not predictable and hence the use of rules for a defined process in software development would not be appropriated. This scenario puts software development under complex unpredictable systems and the Scrum methodology then spells out the best practices that can be used to manage such processes.

Model: Starts from a product backlog (similar to use cases) list i.e. high-level design. Next is the game phase where the sprint backlog lists are used for analysis and

design, this is the black box and unpredictable part of Scrum. Finally is the post-game phase where testing is done and closure of the release.

Techniques and Tools: Planning game is a useful technique that results in a list of prioritized requirements called a Backlog list. Sprint is a technique where the prioritized list of requirements are developed into an executable product without allowing further changes to requirements within the thirty-day iteration, (Schwaber and Beedle, 2002). Daily Scrum is a great project monitoring technique that gives the management control of the project as they know what happens on a daily basis and have to solve the impediments. There are no specific tools mentioned but most project management tools would be useful.

Scope: Teams should be small comprising about seven to ten people. It can be scaled to larger numbers. The methodology is aimed at project management in changing environments, (Abrahamsson et al, 2002).

Outputs: The methodology delivers three main deliverables: the Product backlog, which is a document containing a list of all the features of the proposed system. Sprint backlog, this is a document that contains a prioritized list of the features that can be developed during a sprint. At the end of each sprint or several sprints if they are developed concurrently, a working product increment is delivered.

Adoption and Experience: Provides project management framework that can be supported by any software development practices. Its success with specific projects is mentioned in (Abrahamsson et al, 2002 and Schwaber and Beedle, 2002).

Methodology Product: the Scrum books, (Schwaber and Beedle, 2002 and Schwaber, 2004a) and there is some online help from the Scrum website [www.mountaingoatsoftware.com/scrum/] including seminars and training for the various roles.

Roles & Responsibilities: Scrum Master ensures Scrum practices and values are followed up to the end of the project. Product Owner manages the project and controls and makes visible the Product Backlog list. Scrum Team has authority to decide actions and is self-organizing so as to complete a Sprint. Customer participates in product backlog items, (Schwaber and Beedle, 2002). Management makes final decision and participates in setting of goals and requirements.

3.6.7 Dynamic System Development Methodology (DSDM)

The overview of DSDM is shown on figure 3.9.

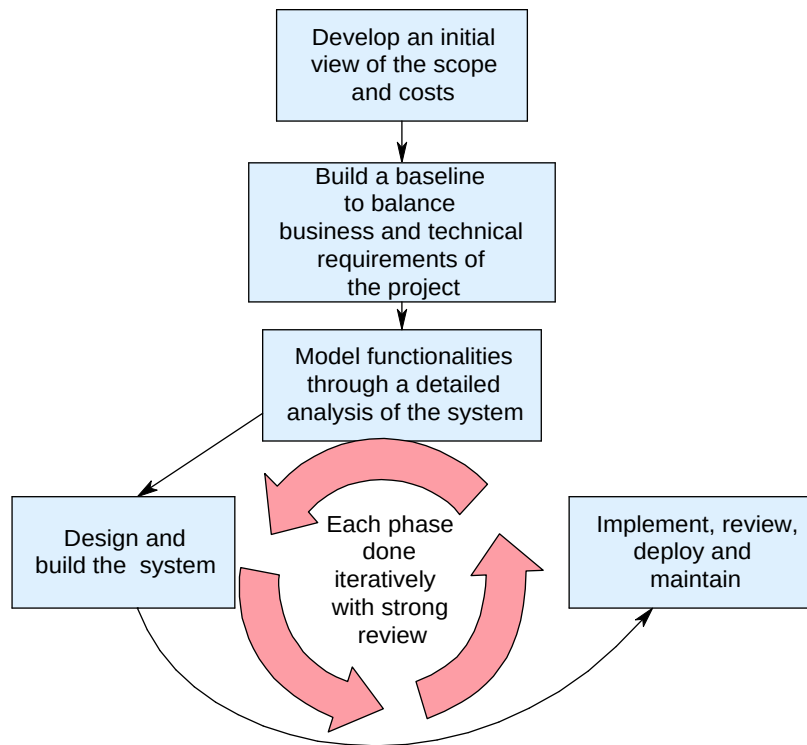


Figure 3-9: DSDM Graphical Overview

Philosophy: DSDM philosophy can be summed up as follows, (Stapleton, 2003):

- Development is a team effort, which combines the customers' knowledge of business requirements with the technical skills of IT professionals.
- High quality demands fitness for purpose and technical robustness.
- Development can be incremental, i.e., not everything has to be delivered at once, and delivering something earlier is often more valuable than delivering everything later.

- The law of diminishing returns applies, which means that resources must be spent developing features of most value to the business.

Model: According to Stapleton (2003), the model is centered on functional prototype and design prototype. There are five phases in the development process. The process starts with feasibility study, which must be done first because it determines whether or not DSDM is suitable for the project and sets the ground rules for the rest of the development. The next phase is the business study, which must be done in sequence with the feasibility study. The business study is a short exercise to achieve enough understanding of the business process to be automated and technical constraints to move forward with safety. The third phase is the functional model, which takes the functional and information requirements identified during the business study and builds analysis and software components models. The fourth phase is the design and build iteration, which provides the engineering of the system and results in a tested system that can be given to the users. Testing is done within the functional model iteration and the design and build iteration. The last phase is the implementation, which mainly involves training the users and handing over the system to them.

Techniques and Tools: Business study workshop is a technique that brings together all stakeholders and elicits business requirements. DSDM training is often recommended in order to effectively introduce it in an organization. DSDM does not specify tools but provides an ideal support environment, which spells out a need for requirement analysis tools, system prototyping tools, design tools, construction tools, testing tools, and reverse engineering tools, (Stapleton, 2003).

Scope: Team size varies from two to six people but there may be many teams in a project. DSDM has been applied to small and large projects. DSDM is mainly applicable to the development of user interface intensive systems and complex business applications, but may also be used for non-IT projects.

Outputs: Feasibility report, outline for development or prototype, functional model prototype, functional prototyping review documents, risk analysis for further development document, design prototype, user manual, Project review report, and final tested system.

Adoption and Experience: DSDM has been in use in Europe since the mid 1990s. It has proved to be a fine alternative for RAD. The DSDM manual spells out the adoption procedure.

Methodology Product: When you join the DSDM consortium you get the following; online DSDM manual, UK government white papers and templates, a wide support network of other members, and receive regular updates on the latest news and events from the consortium, (Stapleton, 2003).

Roles & Responsibilities: Developers and senior developers are involved in development. Technical coordinator defines system architecture and ensures technical quality in the project. Ambassador user brings the knowledge of users into the project and reports project progress to other users. Visionary has accurate view of business objectives of the system and project. Executive Sponsor has the financial authority and responsibility, (Stapleton, 2003).

3.6.8 Lean Development (LD)

The overview of LD is shown on figure 3.10.

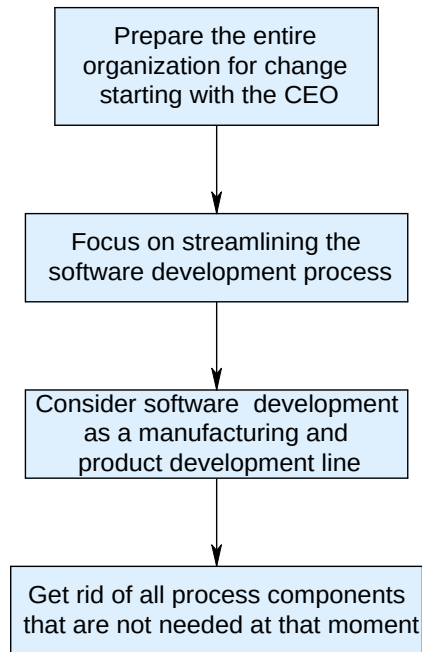


Figure 3-10: LD Overview

Philosophy: LD's philosophy is based on the adoption of a radical change tolerant approach to risk leadership in software development. The fundamental principle is to create change tolerant software with one-third the human effort, one-third the development hours, one-third the investment in tools and methods and one-third the effort to adapt to a new market environment, (Poppendieck and Poppendieck, 2003; Kennedy, 2003 and Highsmith, 2002a). For most organizations this scenario amounts to a turnaround on management policies hence in order for LD to be accepted it aims at changing the attitude of CEOs about change in an organization. LD is based on the Lean Thinking concept from Lean Production, that let customers delay their decisions about what exactly they want for as long as possible, and when they ask for

something it must be given to them so fast they don't have time to change their minds.

Model: LD's view of reality is: creation of visible customer value rapidly, building change-tolerant software, creating only necessary functionality and no more, and aggressiveness, stubbornness, and belief in meeting LD's overall goals.

Techniques and Tools: Value analysis process is used to prove the value of every artifact. Rapid delivery of product is another technique that races with changing requirements, the faster you deliver the least the change in requirements.

Scope: Any software development project where there is need for radical change. Focused at company CEOs. No team size specifications because LD is more of a software development management philosophy than a methodology.

Outputs: Artifacts can be released incrementally as the customer requests before the final product.

Adoption and Experience: Based on Lean Manufacturing with origins as far back as 1935. Poppendieck and Poppendieck (2003) mention many projects where LD has been applied.

Methodology Product: The LD book, (Poppendieck and Poppendieck, 2003), and some online help and tutorials (www.poppendieck.com). Tom and Mary (the authors of LD) also provide a lot of in house training in organizations that need to adopt LD.

Roles & Responsibilities: No specific mention of roles and responsibilities except that LD is aimed at CEOs before it can be implemented in the organization. Project manager and customer are also mentioned in (Highsmith, 2002a).

3.6.9 Extreme Programming (XP)

The graphical overview of Extreme programming is shown in figure 3.11.

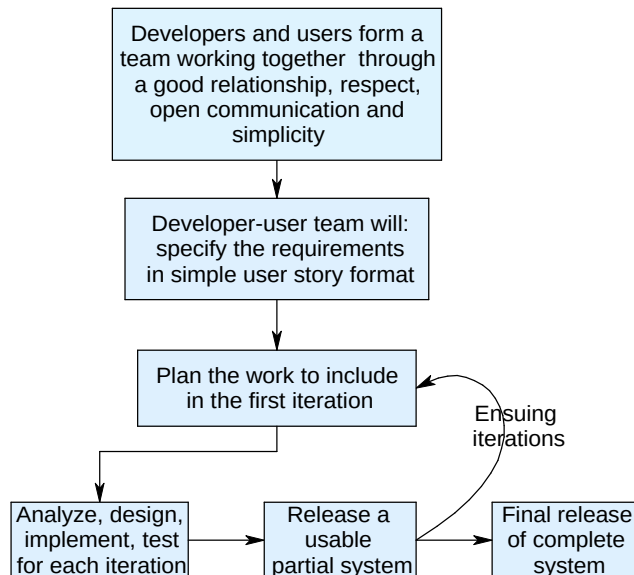


Figure 3-11: XP Overview

Philosophy According to Beck (1999) and Beck and Andres (2004), XP is based on four pillars from which emanate the XP practices, i.e.: technical mastery of programming, aesthetics of code, co-dependency between business and programmers, and building of teams at startup. From these fundamental concepts some important principles are derived such as; *communication* where XP values mutual exchange of knowledge about the problem environment as key to the success of the project, and *simplicity* says the team must build the simplest thing that can work, *feedback* means that the team accepts changes the customer proposes after reviewing the delivered iteration (Highsmith, 2002a; and Brewer, 2001).

Model: User writes stories on cards, developer estimates coding time per story, user chooses sets of stories for first iteration, user writes functional tests for each story,

code is written, Refactoring is used to perfect the design. This amounts to one iteration and the cycle will repeat until the final system is delivered.

Techniques and Tools: Pair programming is a fundamental technique that has contributed to the popularity of XP. Test-first programming means that the system is tested as it is built instead of waiting until the end to test. Refactoring is a technique that allows for change in the design. The other techniques are on-site customer and collective ownership of code. XP does not specify tools for programming. The decision on the tools is determined according to the project's needs.

Scope: XP is best suited for projects that require collocated teams of small to medium size team (three to twenty). On the project side XP is meant for projects where the requirements are unstable and unpredictable.

Outputs: Working code for each small release or increment, and high quality programmers as a result of the learning process in pair programming.

Adoption and Experience: Successfully applied by many companies. XP was developed from practice by Kent Beck and Ward Cunningham, Highsmith (2002a). Experiences of successful experiments with XP are recorded in Williams et al. (2000). XP is one the most widely used agile methodologies and has also been extensively applied in teaching programming in academic institutions, (Williams et al, 2000).

Methodology Product: There are many books on XP, (Highsmith, 2002a), mentions some. Training can also be provided and there is online help on the XP website (www.extremeprogramming.org).

Roles & Responsibilities: Programmer writes tests, and code. Customer writes stories, functional tests and sets implementation priority. Tester helps customer write functional tests and runs functional test and maintains testing tools. Tracker gives feedback on accuracy of estimates and traces progress of iterations, (Abrahamsson et al, 2002). Coach guides team to follow XP process. A consultant is an external member who guides the team to solve problems. Manager makes decisions.

XP Two: the overview given above is mainly based on the first version of XP as presented in (Beck, 1999). In the second version of XP (Beck and Andres, 2004), Kent Beck extends the philosophy of XP to include to a great extent feedback from XP teams. The general notion is that people used XP far beyond the capabilities implied by the first version. XP was initially defined simply as (Beck, 1999): “a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements”. After incorporating the feedback from various XP teams Beck extended the definition of XP to include team size and software constraints as follows (Beck and Andres, 2004):

XP is lightweight: you only do what you need to do to create value for the customer.

XP adapts to vague and rapidly changing requirements: experience has shown that XP can be successfully used even for project with stable requirements.

XP addresses software development constraints: it does not directly deal with project portfolio management, project financial issues, operations, marketing, or sales.

XP can work with teams of any size: there is empirical evidence that XP can scale to large teams.

The second version extends the focus of XP to include how our humanity affects productivity. In this version (Beck and Andres, 2004), Kent Beck presents his experiences in a way that benefits all of us. The two versions however, still share the same values, principles and practices. But in version 2 the way values principles, and practices have been presented emphasizes the relationships among these aspects of

XP. The tone that is clear in the second version is that various teams apply XP differently hence there is no need to be prescriptive.

3.6.10 Notes on Adopting Agile Methodologies

To conclude the sections on the analysis of the most commonly used agile methodologies the following notes from (Mnkandla and Dwolatzky, 2004a), would be of great value to organizations interested in adopting agile methodologies. While limitations of each agile methodology have been mentioned under the scope elements, of greater interest to the purpose of this research would be the common limitations across the agile family. Organizations that are keen on using agile methodologies or have already started using them partially or in full may do well to consider the following limitations:

- Because of the high level of independence and discipline required in the use of agile methodologies a highly skilled and experienced team is required. If that is not possible then include at least one highly skilled and experienced member. As Ken Schwaber often mentions in his agile project management training sessions, “a bunch of novices will deliver a useless product, the choice is yours” (Schwaber, 2004b). Personnel of that caliber may be difficult to get in South Africa because they are very few.
- Agile methodologies were originally meant for small to medium collocated teams of up to ten people and certainly not exceeding twenty. There has been a lot of work in scaling up of teams for large projects, (Eckstein, (2004 and Ambler, 2002b) and some XP and Crystal projects have used up to five hundred people successfully, (Lindval, 2002). Some project priorities may drive towards a more process ceremony or a heavier methodology according to Cockburn (2004b)’s framework.
- One of the limitations to be cautious of according to Bradley (2003) is the apparent serious problem with the no–design–up–front schools of agile development, i.e. evolving from simpler mechanisms frequently results in less

than optimal (and not infrequently deeply flawed) end products. You may assume that these kinds of problems can be solved through refactoring, but refactoring may have serious limitations as you move further into code and deployment where it becomes harder to change things. Bradley (2003) suggests that an up front design through domain modeling as implemented in FDD (one of the agile methodologies) combined with refactoring may solve the problem.

- Agile methodologies have limited support for subcontracting when it comes to outsourcing of software development. Contracts are often based on a plan that includes a process, with milestones and deliverables in sufficient detail to determine a cost estimate (Turk, France, and Rumpe, 2002). However, if the subcontractor has a fine reputation, a contract can be written to allow some degree of flexibility in how the product is developed within time and cost constraints. Such a contract usually has two parts: a fixed part that defines a framework that constrains how changes will be incorporated into the product and the variable part which defines the requirements and deliverables that can vary within the boundaries defined by the fixed part (Turk, France, and Rumpe, 2002). For more details on agile contracts read, (Poppendieck and Poppendieck, 2003; Turk, France, and Rumpe, 2002; Moore, 2004), Martin, Biddle and Noble, 2003 and Morgagni and Bossi, 2001).

3.7 Methodology Selection Approaches

In the context of this research methodology selection refers to the process of choosing the most appropriate software development methodology given a particular software project. The scenario arises because of many software development methodologies in the industry.

According to Cockburn (2000), Glass and Vessey (1994, 1995), Ambler (2002a), Boehm and Turner (2004), and Potts (1993), it is necessary to have multiple

methodologies to choose from because using the same methodology for all application is a weak approach. The techniques for selecting an appropriate methodology for a project at hand become very important. A technique for selecting between agile and plan-driven methodologies was proposed by Boehm and Turner (2004). The technique is based on the risk that a methodology's approach poses to the development of complex applications. In a way this technique is in agreement with Cockburn's technique. However, Boehm and Turner's technique was designed without consideration of the Agile Modeling techniques in (Ambler, 2002b), which deal with risks of agile approaches in complex environments. According to the current evidence as reflected in experience reports from many 2004 agile conferences there is evidence that agile methodologies will always outperform plan-driven approaches in any situation (Poppendieck, 2005). In the same email Poppendieck attributes the poor performance of agile practices to company politics. More success stories can be found in (Highsmith, 2002a; Cockburn, 2002; Hansson, Dittrich and Randall, 2004). Another very interesting overview of agile successes is found in a report on the experiences of agile development by representatives from many European Union companies (Moonen and van Deursen, 2003).

3.7.1 Methodology Selection in South Africa

The software engineering industry in South Africa has a few differences from the rest of the world because of the small numbers of experienced developers and senior management. Such a problem negatively affects the progress of development companies towards mature processes. The results of the Software Skills 2004 Survey, conducted on more than seven hundred ITWeb readers revealed interesting results: Respondents indicated that shortages exist fine business analysis skills (23%), project management (14%), process management (10%) and team leadership (10%). A further 14% indicated their companies have skills gaps, but did not identify which competencies these covered, (Clarkson, 2004).

The main problem that worries leaders in South Africa's Information Technology industry is that most companies do not seem to have any plans in place to address these shortages. This problem has led to a culture in the South African Information Technology industry that discourages specialized skill sets and encourages multitasked skill sets. Specialized skill sets in this case describes the software development skill sets as found in most developed countries where for example a Java developer does Java development and nothing else and a business analyst does business analysis and nothing else in a project. Multiple skill sets characterize the South African industry for a example a Java programmer may also be experienced in C++ and may do business analysis et cetera in a project. As the old saying goes "a jack of all trades ..." to which some have appended the saying "...and a master of none" multiple skill sets may lead to less knowledge in certain areas. The implication of the saying is that broad skills have a tendency of shallowness as opposed to narrow and deeper skills. An example of the multitasked skill set problem is reported in a Monash South Africa (MSA) internal report on an internal review carried out in the Information Technology Services (ITS) department Monash University (2005). In the report the ITS manager acknowledges that the operational problems encountered by his department are partly because of the multiple-skill set nature of his personnel. Another issue of interest that makes the South Africa IT industry peculiar is the problem of internal and external team dynamics, which is strongly affected in a different way from any other by the ethnic differences within the South African environment. This issue will be dealt with in more detail later (section 5.4).

The methodology selection tool proposed in this research is based on classifying methodologies according to usability issues that a developer would analyze before adopting a methodology. The selected elements also look at a methodology from a project manager's perspective, which is a practical way of characterizing methodologies. The software project is classified from a resource planning and quality control perspective. The main difference with the existing techniques is that

the proposed model essentially considers agile methodologies as a family⁹ as opposed to selecting specific agile methodologies (Mnkandla and Dwolatzky, 2004c and Mnkandla, 2005).

3.8 Conclusions

This chapter gave a review of the literature on the characterization or representation of software development methodologies. The special focus was on agile methodologies. While a lot of effort has been put towards the definition of the practices for agile methodologies what still remains a challenge is finding comprehensive techniques to match a specific agile methodology to a given project. Of even more interest is the notion of matching a set of agile methodology practices selected from the agile family to a given project because the general focus in agile methodologies has now shifted from an emphasis on individual agile methodologies to the agile group.

In the next chapter a framework for the matching agile methodology practices to a given project is developed. What is novel about this framework is that though based on existing frameworks, it includes a novel generic agile methodologies model that is based on a selection of agile methodology practices from the entire agile family as opposed to selecting particular agile methodologies.

⁹ Refer to section 4.6 for details and advantages of the family of methodologies approach.

4 Methodology Selection and Tailoring Framework

The challenges and knowledge gaps that exist in the field of software development as identified in the previous chapter are not issues that can be solved by a single all-encompassing solution. In this chapter a framework is proposed that deals with the challenges of matching agile methodology practices to a given project.

The framework is composed of three stages. It starts by classifying projects in order to determine the suitability of using agile methodologies in a given project. The set of parameters used to classify software projects is based on a combination of existing project taxonomy techniques and results of surveys. Then assuming that agile development is found appropriate in the first stage, the second stage of the framework selects the appropriate set of practices. There are two steps in this stage: the methodology selection step performed to determine whether the project's requirements (as determined in stage one) map to one or more agile methodologies. If the requirements map to one agile methodology then it is recommended to use that methodology. If it maps to more than one agile methodology then the second step of stage two is implemented. In step two of stage two, a set of practices is selected from the relevant agile methodologies. The set of parameters used for the methodology selection process is based on a combination of the methodology selection techniques found in (Avison and Fitzgerald, 1995; Abrahamsson et al, 2002; and Boehm and Turner, 2004).

The third stage assumes that agile methodology practices have been selected and provides the details of how to tune them to the given project's environment. The concept of tuning a methodology is not novel but the approach to the tuning is a novel technique called the Generic Agile Methodologies (GAM) model. The reason for developing a new model for this purpose is based on the realization from published literature (Abrahamsson et al, 2003) that many agile methodologies do not specify

how tailoring to different environments can be done and are therefore based on what Abrahamsson et al (2003) calls abstract principles.

The proposed framework forms the main knowledge contribution of this thesis. The framework contributes knowledge towards techniques for traversing the methodology “treasure fields”, particularly the agile methodology “treasure field”. Agile methodologies and the rest of the systems development methodologies provide “treasure fields” of innovative ways that can be effectively used to build relevant systems. Looking for a methodology that will be appropriate for a proposed system is similar to finding treasure in a treasure field. A novel technique for selecting and tuning a set of agile methodology practices suitable to a given project is proposed.

4.1 Background

Existing methodology selection and classification techniques described in the available literature are mainly academic, in the sense that the choice of methodology elements though subjective gives the user a philosophical understanding of the methodologies, see; (Avison and Fitzgerald, 2002; Abrahamsson et al, 2002; Boehm and Turner, 2004; Fitzgerald, 1996; Iivari, Hirschheim and Klein, 1999 and Osterweil and Song, 1996 and 1992c) for such frameworks. The understanding gained from the use of such approaches is fine for teaching and research on methodologies. However when it comes to practical use of methodologies these approaches have limitations. The limitations are caused by the choice of elements for representing the methodologies. Cockburn (2000) proposed a framework that is based on his practical experience (Cockburn, 2000). The specific focus of this research is on methodology representations and tailoring to various environments. Cockburn’s focus was on people and methodologies.

The underlying assumption for selection techniques is that methodologies are not universal, meaning that there is no single methodology that fits all projects,

(Cockburn, 2000; Ambler, 2004; Ambler, Nalbone, and Vizdos, 2005 and Glass and Vessey, 1995).

Software development organizations should be able to choose from more than one methodology. For the purposes of conservatism and balancing between legacy systems and contemporary development technologies it may be necessary to have at least one methodology from the Iterative Incremental Development (defined in section 3.5.2) group and at least one from the plan-driven group. The differences between these two groups are defined in Chapter Three of this thesis. The popular assumption among agile proponents is that there is no need to use plan-driven methodologies at all, (Poppendieck, 2005). There are projects however, that may be very complex and at the same time very risky. These require a balance between a more rigorous structure than what agile provides, and an agile approach. There are projects that have failed because using one approach or the other instead of balancing, (Highsmith, 2005; Boehm and Turner, 2004; Paulk, 2002).

4.2 Architecture of the Framework

The selection framework can be represented by the schematic diagram shown in figure 4.1. The methodology selection process itself is composed of three stages the first two of which are implemented using selection matrixes. It is very important to understand that this framework is only a tool that will help a software development professional to make informed decisions about issues of matching a methodology to a project. While the principles applied in this framework may have elements of subjectivity they are based on sound principles that have been successfully applied by others in this field. However the application of this tool, or any such tool for that matter, does not guarantee success¹⁰ of the project.

¹⁰ Success as described in chapter two.

4.2.1 Philosophy of the Framework

The epistemology of the framework is to view software development processes as phenomena that are made up of two groups of activities namely; social activities and technical activities. The departure point of the framework is then to consider customer values and priorities in a given project and tune the selected practices accordingly. This not only agrees with the heart of agility but also introduces a novel perspective to agile development, which is a simplified twofold view of development activities –social and technical.

These concepts can be summarized in two statements as the philosophy of the framework:

- All activities should be driven by what the customer values and prioritizes in a given project.
- All practices should be classified into social practices, and technical practices.

4.2.2 Process of the Framework

The first stage deals with the classification of the software development projects and leads to the choice of the generic group of methodologies that would be suitable for the class of project. The underlying principle according to Glass and Vessey (1995) and Sol (1983) is that in order to be able to determine the appropriate process or methodology for a project there is a need to first understand the project itself, hence the need to classify the project. To that effect the first stage of this framework presents a matrix (developed by the author) that can be used to classify projects in order to determine the suitability of using agile methodologies for a given project. *While project taxonomies have been done as reported in (Glass and Vessey, 1995 and Sol, 1983) using the proposed set of parameters to classify projects is new.*

The second stage applies the methodology representation approaches described in (Avison and Fitzgerald, 1995; Abrahamsson et al, 2002; and Boehm and Turner,

2004) to classify agile methodologies in order to select a set of relevant agile practices. A selection matrix (developed by the author) is used for this purpose. *Combining the three cited approaches in order to select the most relevant agile methodologies so that the appropriate agile practices are selected is new.* The selection matrix is composed of one methodology for each column. The matrix is not exhaustive as some partial methodologies (for example Agile Testing (AT)) are not included. The reason is that it would be difficult to analyze such methodologies using the suggested set of methodology parameters. In this matrix each methodology element occupies a single row.

The third stage assumes that agile methodology practices have been selected and provides the details of how to tune them to the given project's environment. *The tuning is done through a novel technique called the GAM model.* The process of the GAM model consists of three phases: phase one identifies practices that relate to people issues, phase two identifies practices that relate to technical issues, and phase three outlines the practical tuning steps.

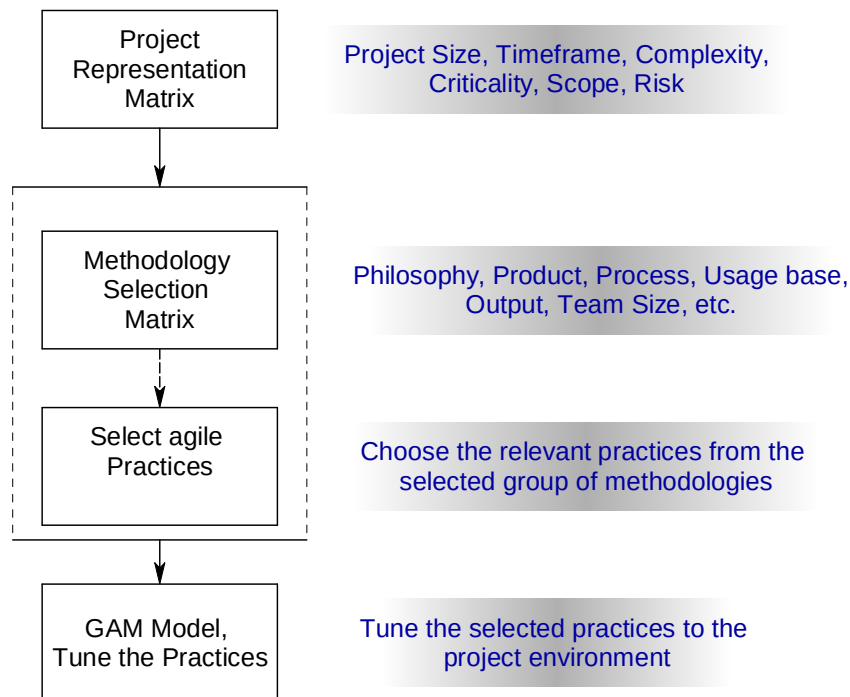


Figure 4-1 Agile Methodologies Selection Framework

Each rectangle represents a stage of the framework; the rectangle with dotted sides represents a phase that is composed of two steps. The arrows represent the movement in between the stages. The dotted arrow represents the link between the two steps in stage two.

The next sections now describe each of the stages of the framework in detail.

4.3 Stage one: Project Classification Matrix

In order to be able to match a methodology to a given project it is necessary to first classify the project and then classify the methodologies. This stage provides a top-level evaluation of project parameters to determine the applicability of agile methodologies to the project. While the choice of project parameters is generally considered subjective by many in the IT industry, the parameters used in this analysis were obtained through interviews with project IT professionals and a review of

literature (Glass and Vessey, 1995; and Andrews and Johnson, 2002) and thus represent a comprehensive set of elements for project classification.

Semi structured interviews were used to derive a set of parameters for the classification of software development projects. Project managers and software developers were chosen for this exercise. The reason for this choice was to get practical information from those in the practice. The questions that were asked are:

- What are the most important parameters that help your understanding of a software development project?
- Where you have the choice of a development process what project parameters influence the choice?

The other interview details are shown in Table 9.1 and Table 9.2 Appendix D.

Semi structured interviews allow discussions between the interviewer and the interviewees. The final analysis of the data involved a comparison with Boehm and Turner's approach and the conclusion was that five parameters shown in Table 4.1 would represent a project in a more practical way than the approach in Boehm and Turner, (2004) which focuses on risk. The details of how to determine the relevance of agile development using Table 4.1 are illustrated with an example in chapter five.

Table 4.1 is important at this stage of the framework. The information that is obtained during project planning is used to decide on the rating of each parameter and is listed in the second column of Table 4.1. For the final selection a low rating implies more relevance to agile development and a higher rating means more relevance to rigorous methodologies. The ratings are explained in the next paragraph.

Table 4-1: Stage One: Determine Between the Two Major Paradigms

Project Parameters	Rating 0 to 5	Methodology Choice
Requirements Stability		
Project Size		

Development Timeframe		
Project Complexity		
Project Risk		

Key:

Requirements stability: 0 means very unstable requirements with a high risk of scope creep. The level of stability increases to a rating of 5 representing very stable requirements where scope creep is minimal. The smaller the rating the more relevant agility is to the project.

Project size: 0 represents a project team¹¹ with two to ten people and 5 represents more than hundred people. The relevance of agility decreases as the rating increases.

Development timeframe: 0 represents a time from one week to two months. Any development timeframe beyond two years would have a rating of 5. The shorter the development timeframe the more relevant agility is to the project.

Complexity: 0 represents those projects not characterized by heavy calculations under high change, high speed, and uncertainty. The complexity increases as the rating increases to 5, which would represent critical projects such as spacecraft or nuclear control software. Agile methodologies would be less applicable as the complexity increases (Boehm and Turner, 2004; Highsmith, 2002a)

Project risk: 0 represents high-risk projects and as the risk decreases the rating increases to 5, which would represent very low risk projects.

4.3.1 Requirements Stability

One of the major challenges in the software development process that is solved by agile methodologies is requirements stability. In agile methodologies the challenge of ever changing requirements is solved through a process that allows the customer to introduce changes at the start of each iteration and through the application of the just-in-time (JIT) approach to modeling as defined in (Ambler, 2004). The techniques

¹¹ Members of the project team include developers and the customer specially invited to be part of the team.

may vary per methodology but the principle of embracing changing requirements prevails. This approach allows the system to improve on quality by satisfying the customer requirements as they emerge. Taking note that the goal of requirements engineering is to create and maintain a system requirements document, (Sommerville, 2004). There are dozens of ways that can be followed in defining the stability of requirements, but the reality is that requirements will always change. In the traditional methodologies approach the principle is that requirements changes are predictable and therefore an upfront plan can be devised to avoid the changes, (Sommerville, 2004). The agile principle is that the changes in requirements must be included in the prioritized list and included in the next iteration as the system is developed iteratively. The agile focus is on starting the development process with the minimum possible requirements allowing changing of the requirements. Refactoring is applied to the code in order to improve the quality of the code fine readability and structure. When documentation is necessary according to customer requirements a less document-driven¹² approach can be assumed according to (Ambler, 2002b). Sommerville (2004) classifies volatile requirements into four types:

Mutable requirements –requirements which change because of changes in the organization’s operating environment.

Emergent requirements –requirements which emerge as the customer’s understanding of the system develops during system development.

Consequential requirements –requirements which result from the introduction of the computer system.

Compatibility requirements –requirements which depend on the particular systems or business processes within an organization.

Both waterfall –based and agile approaches deal with volatile requirements though in different ways.

Project Scope –Schwalbe (2004) defines project scope as the size of tasks that must be done to create physical entities (products) of a project. The size of the scope can

¹² Document-driven in this case refers to a process where detailed documents are the major deliverables on which important decisions about the way forward are based.

increase because of continued addition of products and features and this is called scope creep. Instability of the requirement can also result in scope creep. There is also a tendency on the programmer side to be more creative and add more anasthetics than necessary, which results in scope creep. Scope creep is important because it affects the development timeframe and the project costs. Project scope must therefore be controlled effectively.

4.3.2 Project Size

The size of a software development project can be measured using any of the following parameters:

- Number of people in the development team.
- The estimated cost of the project.
- Complexity of the project.
- The development timeframe.
- Or a combination of any of these parameters.

There is no “best way” to describe the size of a software development project. According to (Eriksson, 2004), the size of a project is usually considered fine the estimated time of the project. This approach means that the project is divided into chunks of two months each and each of these chunks can be developed in parallel according to the prioritized set that can form an iteration.

The roles that people assume in a software project will basically be similar regardless of the size of the project. However if the project is small there is a tendency to assign more than one role per person. In a case where the project is large there may be more than one person per role. In the case of agile development processes which follow a concurrent approach there would be more than one team for large projects developing the system concurrently. From these outlined principles it is clear that there is some proportionality between the project team size and the size of the project. The project size is important in the classification of projects because most agile methodologies

were originally meant to tackle projects of not more than twenty people in the development team. There has been a lot of work in trying to scale agile methodologies with some reported successes and failures in the following sources, (Eckstein, 2004; Basili et al, 2001; Schwaber, 2004a and Lindvall et al, 2004).

4.3.3 Project Development Timeframe

During the planning phase of the software development process the length of time it will take for the system to be developed is estimated. In the case of agile methodologies the estimation is done at the level of the user stories (in XP) or backlog items (in Scrum) or features (in FDD) depending on the specific methodology being used. What this essentially means is that for each small part of a problem (user story) whose solution is to be coded the developer will estimate how long it will take to code. Another technique that is applied to determine the development timeframe at user story level in XP is the developer velocity calculated during the XP planning game (Beck, 1999; Beck and Fowler, 2000; Auer and Miller, 2001 and Crispin and House, 2002). These can then be grouped according to the customer prioritization and put into iterations. The total time can then be estimated. An allowance should be made for additional time resulting from adding new requirements at the beginning of each iteration. The development timeframe is important in the classification of projects in this case because most agile projects are characterized by short development timeframes of not more than two months, (Beck, 1999).

4.3.4 Project Complexity and Criticality

Measuring the complexity of a software project can be a complex undertaking in itself.

Complexity: can be considered as the nature of projects ranging from those marked by heavy real-time calculations, characterized by high change, high speed, and uncertainty, to the consideration of the software program itself, (Highsmith, 2000).

A common method for determining the complexity of a project is to measure the complexity of the software program being developed using Function Point Analysis. Function Points (FP) have been in use for many years and are still very popular even for teaching software metrics. FPs are used to determine the complexity of a software program usually for the purposes of estimating things like effort and the cost of developing a system. However, there are few views that question the reliability of FPs as pointed out in these references (Dekkers and Vogelesang, 2003; Abran and Robillard, 1994; Ahn et al, 2003 and Keuffel, 2001).

According to Boehm and Turner (2004), agile methodologies are unsuitable for dealing with complex software projects, and plan-driven methods on the other hand are suitable for dealing with complex systems. However the principles of Agile Modeling, (Ambler, 2002b) teach that modeling must be done in an agile manner before coding, which gives the developer an opportunity to think through the complexities. Ambler (2005) says that whilst Boehm and Turner's book is fine, the authors completely missed Agile Modeling in their literature review and as a result they missed the concept of Agile modeling, which deals with how agile approaches model complex software projects. There is more evidence from (Eckstein, 2004 and Schwaber, 2004a) on techniques for handling complex problems using agile methodologies.

The complexity of a program can also be determined according to Lehman (1985) using a system of classifying programs into S-type, P-type, and E-type (Specified, Practical, Evolutionary) types of programs.

Where the S-type programs are defined as programs that can be fully specified upfront and Pfleeger and Atlee (2005) suggests that the S stands for static programs, which is in agreement with upfront specification.

E-type programs are software programs that implement an application or address a problem in the real world, which in essence means they evolve to address the changing needs of the real world, (Lehman, 1985).

P-type programs are based on a principle that the problem domain is so wide that though the entire solution is known it would not be practically possible to represent it. To develop the solution, the problem is described in an abstract way and then the specification is written from the abstract view. The program is called P-type because it is based on a practical abstraction of the problem rather than on a completely defined specification, (Pfleeger and Atlee, 2005). Therefore project complexity is more than just a way in which the various lines of code interact with each other and with the processor, memory and peripheral devices and systems.

Criticality: Refers to a system whose undetected defects will result in a lot of damage (Cockburn, 2000). There are four levels of criticality in a project according to (Cockburn, 2000). Defects in the system would cause the loss of the following from the less critical to the most critical:

Comfort –it means that the failure of the system would cause things like delays in delivering some documents etc or meeting some deadlines or even result in long queues at banks etc.

Discretionary money –means that the failure of the system would result in loss of monies that could be recovered otherwise.

Essential money –means that the failure of the system would result in loss of money that cannot be recovered.

Life –the failure of the system would result in the loss of life.

When a project has been identified as critical it means there is more risk in using agile methodologies except with some modification to include more ceremony in the particular methodology.

4.3.5 Project Risk

Project risk is defined as losses or injuries that can threaten the meeting of project objectives, (Schwalbe, 2004). There is need to plan for risks in any project by identifying the risk factors, and coming up with techniques to reduce the effect of the risks, or avoid the risks if possible. An interesting point here is that a methodology in the first place is designed to deal with certain fears and threats to the project. It is therefore important that the methodology spell out what fears it deals with and this should be defined in the philosophy of the methodology. Looking at the nature and values of agile methodologies it would be reasonable to assume that because of their focus on short development timeframes and the iterative incremental approach there would be less risk in applying them to high risk projects, as Ken Schwaber often comments in his presentations; “with agile methodologies you are given a few weeks to decide on the success or failure of the project as opposed to waiting for months or years to reach the same decision” (Schwaber, 2004b). It is worth noting that although agile methodologies have been designed to deal with risk as explained in the preceding sentences, many organizations still find it risky to use agile methodologies because of their relatively new status in the software development profession.

4.3.6 Customer’s Priorities/Availability

This parameter is related to all the other parameters and hence it has not been included as a row in the project classification matrix Table 4.1. This parameter essentially deals with what the sponsor of the project has put as the values and priorities of the project. In one project the development time may be of the highest priority, meaning that the developer is given a fixed timeframe within which to complete the project. The option would then be to vary the cost and scope to achieve the goal. In another project the priority of the customer might be on fixing the cost, which would imply that the time and scope would then be varied to meet the goal. Yet another scenario may involve a fixed scope priority, meaning that the time and cost would then be varied to achieve the goal. To strike a balance among all these

parameters there would be a need to also take into consideration the issue of the quality of the intended product.

Not to be overlooked is the importance of customer availability i.e. the ability of the customer to provide information and make decisions in a timely manner. These are critical to the success of agile projects, (Ambler, 2002a and Beck, 1999). Therefore customer priorities and availability is a parameter that must be considered against each of the other parameters before making the final decision.

4.3.7 Final Choice

A first pass through the project classification matrix will normally give an indication of which group of methodologies is the possible choice. However, the customer priorities and issues of customer availability affect the final choice. There is therefore a need to go through each project parameter in the matrix again reviewing the rating in relation to the priorities of the customer. In some cases people may want to mix traditional and agile practices. There is some common ground between these two groups see (Boehm and Turner, 2004), but there are also some fundamental difference. It is therefore important to consider the impact of these differences before mixing the practices.

Stage one helps to determine the suitability of using agile development in a given project. For the purpose of clarifying the sequence from stage one to stage three of the framework it will be assumed that agile development has been chosen in stage one.

4.4 Stage two: Methodology Selection Matrix

This section describes stage two of the framework. The aim of this stage is to determine which agile methodologies would be more applicable to a given project and to select agile practices from these methodologies. The practices will then be

tuned to the project environment in stage three. Agile methodologies literature generally shows that most agile methodologies work better when combined with other agile methodologies because they focus on different areas of software development (Schwaber and Beedle, 2002; Boehm and Turner, 2004; Highsmith, 2002a; Ambler, 202b). The reason for selecting practices from the agile set is to fully equip the developer for the development activities.

There are two steps in this stage: the methodology selection step performed to determine whether the project's requirements (as determined in stage one) map to one or more agile methodologies. If the requirements map to one agile methodology then it is recommended to use that methodology. If it maps to more than one agile methodology then the second step of stage two is implemented. In step two of stage two, a set of practices is selected from the relevant agile methodologies.

The choice of the methodology taxonomy parameters is derived from a combination of work by (Avison and Fitzgerald, 2002; Sol, 1983; Brandt, 1983; Abrahamsson et al, 2002; and Cohen, Lindvall, and Costa, 2003. After comparing Avison and Fitzgerald's framework with two frameworks one by (Cohen, Lindvall, and Costa, 2003) and another by (Abrahamsson et al, 2002) two parameters; model and scope were replaced by process in the proposed framework. Scope in this framework is means something different from Avison and Fitzgerald (2002). Two other parameters were introduced; roles and responsibilities, and support for distributed teams. These two were derived from (Cohen, Lindvall, and Costa, 2003 and Abrahamsson et al, 2002) because of their relevance in how the methodology defines the size of the team and the physical locations of the team.

Step One of Stage Two

This step involves first describing all the parameters for each methodology as shown in Table 4.2. The first column lists the methodology parameters. The second column provides the corresponding description. The third column explains how each

parameter relates to the customer values and priorities as identified in stage one. *The reason for using customer values and priorities as the focus for this selection process is to emphasize one of the agile development values centered on satisfying the customer.* In the fourth column are entered the scores for each parameter based on how well the description of the parameter satisfies the values and priorities of the customer. The value of zero means full satisfaction and the value of five means complete variance.

Each agile methodology under consideration will have its own Table 4.2 the parameters and the corresponding description remain constant for each project and the last two columns of Table 4.2 will change per project. Table 4.3 summarizes the scores from each methodology's Table 4.2 results.

Figure 4. 2 summarizes how the stage two works according to the given tables. Rectangles describe the activities and arrows show the sequence.

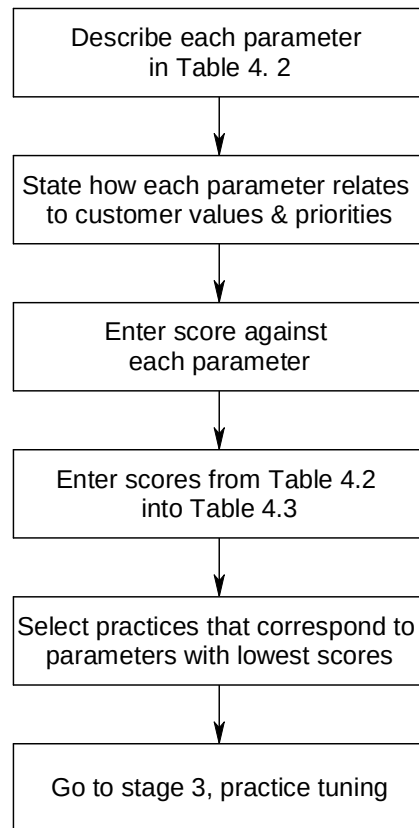


Figure 4-2: Summary of Stage Two

Table 4-2: Stage Two: Methodology Description

Parameter	Parameter Description	Relation to Customer Values and Priorities	Score
Methodology Philosophy			
Methodology Process			
Methodology Techniques and Tools			
Methodology Scope			
Methodology Outputs			
Adoption and Experience			
Methodology Product			
Roles and Responsibilities			
Support for Distributed Teams			

Table 4-3: Stage Two: Methodology Selection Matrix

Parameters	ASD	Crystal	FDD	AMDD	ICONIX	Scrum	DSDM	LD	XP	0 Score
Methodology Philosophy										
Methodology Process										
Methodology Techniques and Tools										
Methodology Scope										
Methodology Outputs										
Adoption and Experience										
Methodology Product										
Roles and Responsibilities										
Support for Distributed Teams										
Score per methodology										Score per parameter

The following sections define each of the methodology parameters according to the context of the framework.

4.4.1 Methodology Philosophy

This is a set of principles underlying the methodology. This includes the practices of the methodology, which spell out the objectives among other issues. The philosophy of a methodology essentially describes the fears and risks that the author of the methodology originally intended to solve, (Avison and Fitzgerald, 2002). It is in the philosophy that the types of problems that the methodology deals with are outlined. From a practical perspective the philosophy would give the user an understanding of the original area of problems the methodology was designed to solve so that if the actual problem at hand is outside the intended domain the user will be ready to tailor the methodology accordingly.

4.4.2 Methodology Process

This parameter describes how the methodology views reality. The model may be reflected in the lifecycle or development process of the methodology. The model provides a means of communications, captures the essence of a problem or a design, and gives insight into the problem area (Avison and Fitzgerald, 2002). The importance of this parameter is that it gives the user a real world view of the series of activities that are carried out in the development process.

4.4.3 Methodology Techniques and Tools

This parameter helps the user to identify the techniques and tools applicable to the methodology. Tools may be software applications that can be used to automate some tasks in the development process, or they can be whiteboards and flip charts. In fact it is the use of tools that makes the implementation of a methodology enjoyable. Organizations therefore tend to spend a lot of money acquiring tools and training staff

on tools. As technology evolves and new tools emerge more acquisitions and training are usually done.

Each methodology comes with its own techniques that may be relevant or irrelevant to the problem at hand. Examples of techniques, say in Extreme Programming would be pair programming, and the scrum meeting in Scrum methodology. The user then analyzes these techniques in relation to the present project, if there is need to adapt some of them and include variations that will be part of tailoring the methodology. However, since the purpose of this comparative analysis is to select the most suitable methodology for a given project, each of the parameters is compared against other methodologies and the one that best meets the requirements of the project is chosen.

4.4.4 Methodology Scope

This parameter outlines the details to which the methodology's development framework is spelt out. This is where the methodology specifies what it covers within a project. The importance of this parameter is to help the user to identify the list of tasks that the methodology will help manage and anything outside that will have to be done by other means. It should be noted that a methodology does not prescribe everything that needs to be done in a software development process it simply gives guidelines that help in managing a project.

4.4.5 Methodology Outputs

This parameter defines the form of deliverables to be expected from the methodology. For example if an organization purchased Lean Development methodology, would they generate code from application of the methodology, or would they get some documents, online help etc, (Avison and Fitzgerald, 2002). Each agile methodology will give different outputs hence the user can choose the methodology that gives them the output they require.

4.4.6 Adoption and Experience

Some organizations have a policy of not using methodologies that have no experience base. This parameter spells how long the methodology has been in use, and who has used it. The parameter is measured by the following elements: was the methodology developed from practice (industry) or from theory (laboratory)? How many times has the methodology been in use? i.e. user base. Is it used by developers or management? What skill levels are required to use the methodology? (Avison and Fitzgerald, 2002).

4.4.7 Methodology Product¹³

If somebody buys the methodology what do they get for the purchase? Is it software, written documents, online help hours of training etc? (Avison and Fitzgerald, 2002). It is interesting to note the value of a methodology is not necessarily in the product but in the general attitude of the entire organization intending to purchase the methodology. The ability of the organization to implement change is also important when considering purchasing a methodology.

4.4.8 Roles and Responsibilities

According to Abrahamsson et al (2002) this parameter refers to how the methodology caters for the allocation of specific roles through which the software production in a development team is carried out. This element specifies the specific tasks that exist in a project according to that methodology and the activities involved in that task. The team structure is in fact defined in this parameter.

4.4.9 Support for Distributed Teams

This parameter defines how the methodology caters for a system that is developed by project teams that are distributed in various geographical areas. Issues of interest in

¹³ The difference between methodology output and product is; output is what you get from the vendor when you buy the methodology, product is the final deliverable after using the methodology.

such a situation would be direct communication, meetings, etc. (Cohen, Lindvall, and Costa, 2003).

4.4.10 Partial Agile Methodologies

There are some agile methodologies that do not define a full process and it is not so easy to represent them according to the proposed matrix. These methodologies are usually not used by themselves but work as part of other agile methodologies. Among such methodologies are found AMDD, FDD, Agile Data (AD), (Ambler, 2003) and Agile Testing (AT) (Marick, 2001). To deal with such methodologies it may be better to start by modeling them according to the Generic Agile Methodologies model (GAM) in section 4.5.2 developed and described in the research presented in this thesis.

4.4.11 The Role of the Human Expert

The contribution of the human expert cannot be ignored in a selection process like this one. Software development experts with practical experience in this field have a lot of knowledge that can be classified as tacit knowledge because it is gained through experience and is not written down in any form. This framework provides for tacit knowledge by allowing the human expert to analyze the results of both selection matrixes and then make the final selection of the methodology. As tacit knowledge is difficult to quantify, this phase of the framework may seem quite subjective, but the strength of tacit knowledge rests in the team spirit that puts trust on experts to do what they know best within their professional ethics.

Step Two of Stage Two

In this step practices are selected based on the results from step one. There are two possible ways of selecting the practices. One way is to pick the methodologies with scores of zero up to two (with three to five representing scores from parameters that are less supportive of the customer priorities and values) and then select their

practices. The limitation of this way is that if for example a methodology like XP has a score of say four against one parameter its total score exceeds the required value though XP might be relevant with regard to the rest of the parameters. This approach is therefore not recommended. The other way is to relate methodology parameters to methodology practices, for example what would be the practices in XP that implement a parameter such as support for distributed teams. So for each parameter practices are then selected from the methodologies in which the parameter has a score of between zero and two. The practices are then entered into the second column of Table 4.4 and the first column contains the corresponding methodology.

Table 4-4: Methodology Practices

Methodology	List of Practices
XP	
Scrum	
LD	
DSDM	
FDD	
AMDD	
ICONIX	
Crystal	
ASD	

4.5 Stage Three: Tailoring Agile Methodology Practices

This section describes the Generic Agile Methodologies (GAM) model for tailoring agile methodology practices to a given environment. Tailoring of agile practices helps developers to find feasible ways of working with the selected agile practices. Research reveals that many agile methodologies are based on abstract principles (Abrahamsson et al, 2003), which means that they do not specify the details of how to

tailor their practices to a project's environment. Literature also reveals that some agile methodologies have no defined process and for those that have a process the process is very complex making it difficult for the methodologies to be useful.

The GAM model proposes a way that clearly describes how to tune agile practices to a given environment. The GAM approach views information systems development as a social–technical activity (Roque, Almeida, and Figueiredo, 2004). Hence the GAM model defines the social and technical aspects of the selected practices and provides ways to tailor them.

4.5.1 A Family of Methodologies Model

Motivation for the GAM: The motivation for dealing with agile as a family of methodologies is based on findings that as agile methodologies progress towards maturity in the software development industry practitioners are tending to select practices that are relevant to their projects from any of the agile methodologies rather than adopting individual methodologies (Beck and Andres, 2004; Basili et al, 2001; Lindval et al, 2002).

One of the major findings about agile methodologies is the striking similarity amongst the agile practices. They might be called by different names but when analyzed they are found to implement the same principle. An example is the 'stand up meetings' in XP, which is similar to the scrum meeting in Scrum. Of importance also is that even the authors of agile methodologies no longer emphasize their methodology boundaries and would use practices from other methodologies, see (Beck and Andres, 2004). It can therefore be concluded that agile practices address the same issues using various real life models. For example, Lean Development views software development using a manufacturing and product development metaphor, Scrum views software development using a control engineering metaphor, Extreme Programming views software development using a sitting together social

activity metaphor, Adaptive Systems Development views software development using the theory of complex self-adaptive systems metaphor, etc. hence the philosophy of the GAM model assumes an agile family of methodologies perspective and uses the selected set of practices that are relevant to a given project.

4.5.2 The Generic Agile Methodologies (GAM) Model

This section describes the GAM model in detail. The following structure will be used to define the GAM model: values, and process. The GAM Philosophy is expressed within the two values of the GAM model. There is no need to deal with philosophy separately because only a full methodology needs a detailed description of the philosophy see (Avison and Fitzgerald, 1995; Checkland, 1981) for the difference between methodology, method, and model.

Values of the GAM model

The GAM model has two values namely:

- 1) Bonding, which is concerned with bringing together of everyone who is involved in the project. The major groups involved would normally be the development team, the management team, the executives, and the rest of the stakeholders. If these groups can work together harmoniously then chances of project success would be increased meaningfully.
- 2) Quality, which is concerned with all technical issues such standards of development, development environments, programming tools, management tools and everything else that contributes to the delivery of a high quality product.

The idea behind the GAM model is therefore to identify the practices that relate to social issues in order to achieve effective bonding and also identify practices that relate to technical issues in order to deliver quality. These two groups of practices are then tailored to the environment of the project to set up a feasible way of working.

GAM Process

The GAM model is implemented in three phases as shown in figure 4.3. The first phase is the organic aspects, which focuses on identifying agile practices that are concerned with social issues of the development process. The second phase is the mechanistic aspects, which focuses on identifying agile practices that are concerned with technical issues. The third phase is the synergistic adaptation, which focuses on how to bring all the identified practices into the context of the project's environment. The details of each phase are explained in the following paragraphs.

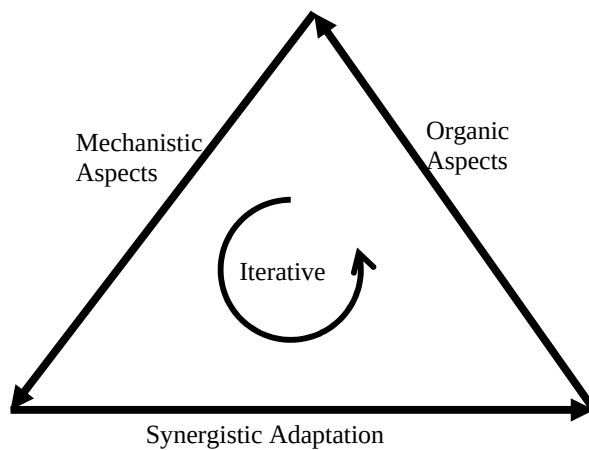


Figure 4-3 Generic Agile Methodologies (GAM) Model

Organic Aspects: phase one defines how the methodology caters for social issues also known as people issues (Cockburn, 2003), and the way these social issues affect the software architecture and how these human issues drive project outcomes.

This phase of the GAM model includes a representation of one of the agile Manifesto values that emphasizes the importance of individual members of the software development team and the way they interact and work as a team.

The Agile Manifesto values are shown on Table 4.5

Table 4-5: Values of the Agile Manifesto

Individuals and interactions over processes and tools
Working software over comprehensive documentation

Customer collaboration over contract negotiation
Responding to change over following a plan

The first activity in phase one is to determine from the list of practices selected in step two of stage two which practices relate to people issues in the project. The next activity is to pick each of the practices that relate to people issues and list the corresponding tasks and resources. Tasks are the different work item that each member of the project team is expected to perform. Resources are hardware, software and manpower requirements of the project.

Mechanistic Aspects: the second phase defines how the methodology values the technical aspects of software development such as the development platform, development technology and form of the deliverables (is it a document or code etc?). This phase of the GAM model includes a representation of another of the agile Manifesto values that regards working software as more valuable than documents.

The first activity in phase two is to determine from the list of practices selected in step two of stage two which practices relate to technical issues in the project. The next activity is to pick each of the practices that relate to technical issues and list the corresponding tasks and resources.

Synergistic Adaptation: the third phase represents the tailoring of the selected practices in the two preceding phases of the GAM model. The GAM model in this phase provides specific practical guidance on how to tailor the selected agile practices to a given project environment.

The first activity in phase three is to go to phase one and consider the selected group of practices that relate to social issues. The tailoring activity in this case involves reviewing the feasibility of each practice. Where it is not possible to implement a

specific practice the general principles that relate to the practice should be considered in order to find a feasible way of satisfying the principles.

The *iterative* core of the GAM model means that the tailoring of practices is done for each iteration of the development process.

4.6 Conclusions

This chapter gave a detailed description of the framework for the selection and tailoring of agile practices. The framework has three stages. The first stage determines the appropriateness of using agile methodologies in a given project. The process of project taxonomy is implemented using a set of project parameters determined from a combination of related past work and a survey of IT professionals. Stage one ends by assuming validity of using agile development and leads to stage two. In stage two the aim is to select the most relevant agile methodologies to the project after which practices from these agile methodologies are selected according to customer values which leads to the tailoring of the practices in stage three of the framework. Stage three of the framework presents a novel technique (GAM model) for tailoring the practices selected in stage two. The tailoring technique follows a three phase process. The principle behind the GAM model is to separate the practices into two major groups of social related practices and technical related practices, which forms the two phases. The last phase then deals with the tailoring of these two groups into the project's environment.

The next chapter presents an example of how the selection and tailoring is done using a hypothetical project.

5 Implementation of the Framework

In this chapter the three stages of the framework will be analyzed in detail to illustrate the feasibility of the framework.

Business Scenario

A retail and logistics company has clinched a BEE (Black Economic Empowerment) deal to do distribution of retail goods mainly in the black townships of Johannesburg. The majority of the customers are owners of spazza¹⁴ shops, taverns, shebeens¹⁵ and supermarkets, bottle stores and general dealer stores. The existing method of payment is manual. Because of the nature of the customers, the system is cash on delivery meaning that the delivery personnel have to collect the cash as they deliver the goods. This meets with operational problems such as unavailability of cash at the time of delivery, implying collection of cash later or no delivery. Another problem with the current system is the possibility of losing the cash to robbers. Worse still, as the group managing director outlined; “the delivery personnel may fake a hijacking and convert the cash to their own use”. Tired of these kinds of problems the company decided to consult IT specialists for an automated wireless card reader (WCR) solution that would provide cashless transactions.

The proposed system therefore intends to solve these problems by providing an automated cashless payment system. This however, means that the bank will have to agree to open debit accounts for the retailers concerned and provide the debit card services that will allow electronic transfer through the WCR. The debit card system already exists in the bank, meaning that the prospective customers would have to comply with the bank’s rules about debit cards and pass through the bank’s vetting system like any other customer in order to qualify for the debit card.

¹⁴ A South African term for small corner shops usually found in Townships and informal settlements.

¹⁵ An unlicensed drinking establishment found in South Africa. According to the online dictionary.com shebeens are also found in Ireland and Scotland.

The proposed WCR system should provide two functions, a function by which the retailer may check their bank balance, and a function by which retailers may pay the distributors. The retailers are major users of this system hence they are expected to apply for the debit card from the bank. The retail and logistics company will provide the WCR system and the delivery personnel will always carry the wireless card reading device with them during their errands.

The client's values in this project include: a quick (one week) proposal of the solution and patenting of the device to beat competition, negotiating with a bank to sponsor the project, building the full system within three months, and using the GSM network for the wireless transactions.

5.1 Implementation of Project Classification Matrix

According to the project information the ratings for this project are shown in Table 5.1. The way the ratings are arrived at is explained in following paragraph.

The customer team presents the project requirements to the developer team at a project-planning meeting. In order to determine the suitability of using agile development in this project the developers apply the project classification matrix. Each project parameter is analyzed to determine the appropriate ratings.

Requirements stability: the developer and the customer agree that there will be changes in the requirements since the project involves wireless technology, which is characterized by fast changes in technology and applications. So they agree on a value of '1', which indicates that agile development is the most suitable process. The meaning of the numbers is explained in chapter four.

Project size: the development team was estimated to be composed of five people, which is suitable for agile development hence the rating for this parameter was given a value of '0'.

Development timeframe: the customer needed the system to be completed within three months hence the rating for this parameter was given a value of ‘1’, which is suitable for agile development due to the short development timeframe.

Project complexity: the system would process valuable financial transactions hence the developer and the customer agreed that the system was of a complex nature. The value of ‘5’ was assigned to this parameter, which indicates the suitability of rigorous development.

Project risk: the major concern was that the proposed system ran the risk of being overtaken by other technologies for example smart cards, hence the parameter was assigned a value of ‘1’, which indicates the suitability of agile development.

Table 5-1: Project Classification Matrix for SS Technologies Project

Project Parameters	Rating 0 to 5	Methodology Choice
Requirements Stability	1	Agile
Project Size	0	Agile
Development Timeframe	1	Agile
Project Complexity	5	Rigorous
Project Risk	1	Agile

From this matrix the project ratings indicate that the use of agile development is suitable. The main reasons are the size of the project, unstable requirements, the short development timeframe constraint, and high risk.

5.2 Implementation of Agile Methodologies Selection Matrix

This section illustrates how to implement stage two of the framework. The results from stage one of the framework show that agile development would be suitable for this project. The planning team meets again to determine the most relevant agile approach for the given project. This is done in two steps the first step involves rating methodology parameters according to customer priorities and values for the project.

Table 4.2 is completed for each methodology as shown in Table 5.2 in the case of XP. Table 5.3 then summarizes the results from each methodology's Table 5.2. Note that the methodology parameters are described as given in section 3.6. and summarized in Tables 9.1 to 9.9 Appendix G.

Step One of Stage Two

Explanation of Table 5.2

The analysis involves determining how well each methodology parameter meets the corresponding customer values and priorities. A number is then assigned where a value of zero means full correspondence between what a particular parameter implies and the customer values and priorities, and the value of five means complete variance, as described in 'step one of stage two section 4.4'. Methodology parameters are described according to section 3.6 and customer values and priorities are obtained from the business case.

XP philosophy: implies activities such as: sitting together and writing code in a socially conducive environment, following a high coding standard, and co-dependency between business and programmers. These are applicable to the basic values of the case at hand; hence the score of '0' would be appropriate.

XP Process: Starts from the writing of user stories on cards this is done by the customer who is part of the development team. The user stories are then given to the developer who estimates coding time per story. The customer chooses sets of stories that form the first iteration. The customer writes functional tests for each story. The process of coding then follows, which includes refactoring to perfect the design. These activities form the first iteration and the cycle will repeat until the final system is delivered. In order to implement the XP process in this case there would be a need to train the customer to understand what a story is and how to test the stories. Probably the first iteration would involve more work than the rest of the iterations; hence the score of '2' would be appropriate.

XP Techniques and Tools: Pair programming, test-driven development, refactoring, and collective ownership of code would be possible. The practice of having an on-site customer would not be possible in this case. The score of '2' would be appropriate since there is compromise on one technique.

XP Scope: Since XP is best suited for projects that require collocated teams of small to medium size team (three to twenty) and the proposed team is small and collocated it is appropriate to consider requirements of this parameter (scope) as compatible with the priorities and values of the customer and a value of '0' would be appropriate.

XP Outputs: The use of XP is intended for the delivery of a full working software system, which is only part of expected project output. Since the project includes a wireless card reader a value of '2' would be appropriate indicating that the software part would not represent the complete product.

XP adoption and experience: since there is evidence of successful use of XP and also its failures it would be possible to make informed decisions about the use of XP in such a project. Therefore a value '0' would represent full compatibility.

XP Product: Training material and online help on XP is readily available from many organizations hence this parameter is fully compatible with the values and priorities of the customer. A value of '0' would therefore appropriate.

Roles and Responsibilities: the size of the team would not allow for each role per individual. Hence this parameter can only be implemented by assigning more than one role per individual. A value of '4' would be appropriate showing incompatibility.

Support for Distributed Teams: this parameter is not relevant to this project it therefore be left without a value.

Table 5-2: Methodology Description for Extreme Programming

Parameter	Parameter Description	Relation to Customer Values and Priorities	Score
Methodology Philosophy	Technical mastery of programming, aesthetics of code, co-dependency between business and programmers, and building of teams at startup.	High quality code, customer must know what is happening at all times	0
Methodology Process	User writes stories on cards, developer estimates coding time per story, user chooses sets of stories for first iteration, user writes functional tests for each story, code is written.	Difficult to implement because of the need to train users in this respect	2
Methodology Techniques and Tools	Pair programming, test-driven development, refactoring, on-site customer, and collective ownership of code.	The case involved two programmers, so most techniques would be possible except for the on-site customer.	2
Methodology Scope	XP is best suited for projects that require collocated teams of small to medium size team (three to twenty).	The proposed team was small and collocated	0
Methodology Outputs	Working code for each small release or increment, and high quality programmers as a result of the learning process in pair programming.	The software part meets what the customer expects though the final product would include a wireless card reader electronic device.	2
Adoption and Experience	Successfully applied by many companies (Highsmith, 2002a). Experiences of successful experiments with XP are recorded in Williams et al. (2000). XP has also been extensively applied in teaching programming in academic institutions, (Williams et al, 2000).	Fully corresponds	0

Methodology Product	There are many books on XP, (Highsmith, 2002a), mentions some. Training can also be provided and there is online help on the XP website.	Fully corresponds	0
Roles and Responsibilities	Programmer, customer, tester, tracker, coach, and manager.	Due to the size of the proposed team some roles need to be combined	4
Support for Distributed Teams	XP has been used in distributed environments with tailoring o practice like pair programming and standup meetings.	Not relevant in this case	

Table 5-3: Agile Methodologies Selection Matrix for SS Technology Project

	ASD	Crystal	FDD	AMDD	ICONIX	Scrum	DSDM	LD	XP
Methodology Philosophy	0					0			0
Methodology Process	3					1			2
Methodology Techniques and Tools	2					2			2
Methodology Scope	1					0			0
Methodology Outputs	3					3			2
Adoption and Experience	1					0			0
Methodology Product	2					0			0
Roles and Responsibilities	2					4			4
Support for Distributed Teams									

Table 5.2 is created for each methodology and the values in the last column are then copied to Table 5.3. The next step involves identification of the appropriate practices for this project. Each parameter in Table 5.3 is analyzed. As an example each methodology for which the philosophy has a value of '0' (or the lowest value) is noted as the potential source of the relevant practices. For example ASD, Scrum, and XP have values of '0' against methodology philosophy. That leads to the next step in this stage.

Step Two of Stage Two

This step involves the identification of practices that can implement the philosophical requirements in each of these three methodologies. The practices would then be ready for the next stage of tuning described in section 5.3. The process continues until each parameter is covered. The practices are selected using Table 5.4 where the practices for each methodology are listed. The analysis is done by picking each methodology parameter with the lowest score from the most relevant set of methodologies as identified in step one using Table 5.3 and then finding the relevant practices and list them in Table 5.5. Some of the practices imply similar activities but are named differently in different methodologies for example; scrum meeting in Scrum is similar to standup meeting in XP. Stage three provides the details of how to adapt these practices to the given environment.

Table 5-4: Agile Practices per Methodology

Methodology	Practices
XP	The planning process, small releases, metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, forty-hour work week, on-site customer, and coding standard.
Scrum	Scrum meeting, thirty-day Sprint with no changes during a Sprint, Sprint planning meeting, Capture requirements as a product backlog, self-organizing teams.
LD	Eliminate waste, Minimize inventory, Maximize flow, Pull from demand, Meet customer requirements, Do it right the first time, Empower workers, Ban local optimization, Partner with suppliers, and create a culture of continuous improvement.
DSDM	Because DSDM is a framework it does not specify practices but the following are commonly used by practitioners: Joint Application Development (JAD) sessions, workshops, create a team of peers responsible for key project decisions, frequent releases, get consensus from all stakeholders, prioritize development based on business benefit not risk, and timeboxing.
FDD	Domain Object Modeling, Developing by Feature, Individual Class (Code) Ownership, Feature Teams, Inspections, Regular Builds, Configuration Management, Reporting/Visibility of Results.

AMDD	Active Stakeholder Participation, Apply the Right Artifact(s), Collective Ownership, Consider Testability, Create Several Models in Parallel, Create Simple Content, Depict Models Simply, Display Models Publicly, Iterate to Another Artifact, Model in Small Increments, Model With Others, Prove it With Code, Use the Simplest Tools, The supplementary practices are: Apply Modeling Standards, Apply Patterns Gently, Discard Temporary Models, Formalize Contract Models, Model to Communicate, Model to Understand, Reuse Existing Artifacts, and Update Only When It Hurts.
ICONIX	Identify and describe all the usage scenarios for the system to be built, Identify reusable classes, Define problem domain, Identify classes that belong to that domain, account for all functional requirements in the design, Design classes that will capture required system behavior through fine design principles such as: minimizing coupling, maximizing cohesion, generality, and sufficiency, and so forth.
Crystal	Working together in the same room, Expert within earshot of the team, information radiators (papers posted on team's room or hallway), Spectral sample of the main components of the project to determine meaningfulness of project, Frequent delivery, Reflection workshops, Accessible customer and project interviews.
ASD	JAD sessions, Plan cycles, Timeboxing, collective ownership, Customer focus groups for reviewing the application, project postmortems.

Table 5-5: Customer Values/ Priorities and Practices for WCR Case

Customer Values and Priorities	Practices
Automated cashless transaction, One week proof of concept, Patenting of the WCR, Getting a sponsor, Three months for the full system, Use of GSM network for the system.	Timeboxing, iterative development, frequent releases, Scrum meeting, standup meeting, get consensus from all stakeholders, active stakeholder participation, working together in the same room,

5.3 Implementation of the GAM Model

This section implements the third stage of the framework. This stage illustrates how to apply the GAM model to tune the practices selected in section 5.2 to the given project's environment. The tuning is done in three phases the first two phases classify the practices and the last phase tunes the practices as described in the following paragraphs.

The practices are classified into social and technical groups by considering each practice and analyzing the nature of the activities implied in the practice. Social practices deal with people and their interactions. Technical practices deal with the how-to details of system design, coding, testing and deployment.

Organic aspects: the first phase identifies practices that deal with people issues. In the given case the practices are:

Scrum meeting, standup meeting, get consensus from all stakeholders, active stakeholder participation, and working together in the same room.

Mechanistic aspects: the second phase identifies practices that deal with technical issues. In the given case the practices are:

timeboxing, iterative development, and frequent releases.

Synergistic adaptation: both standup meetings and Scrum meetings implement the agile principle of accountability and face to face communication. Since both are applicable to the given environment applying just one of them would do. Suppose Scrum meetings are used there could be situations where the team may not be able to meet at the appointed time. The tuning would then involve setting a time that is convenient to all members of the team. The meetings should not be cancelled.

Getting consensus from all stakeholders and active stakeholder participation are practices that implement the agile principle of involving all stakeholders. In this case it means that the bank (potential sponsor), retailers and logistic companies would be the stakeholders.

Technical practices relevant to this project include: frequent releases, iterative development, and timeboxing, which implement the same principle of working with minimum requirements for a fixed time and releasing a usable deliverables of the system. Different agile methodologies have varying lengths of the iterations for example Scrum uses one month Sprints. In this case the customer expects the full system to be released within three months. It would be reasonable to agree on an iteration of one month. At the end of the first iteration the length of the iteration could be reviewed.

The *iterative center* in the diagram on figure 4.3 means that the model is applicable to the chunks of iterations that the problem domain is broken down into according to agile philosophy. The way to do it is by performing the analysis at the beginning of each iteration in order to select the most relevant practices for that iteration as new requirements are introduced. There could be situations where practices change from one iteration to the other. Remember that the focus of the framework is to provide

satisfaction to the customer through the emphasis on values and priorities and not to satisfy the methodology process.

5.4 Methodology Tailoring in a South African Context

In this section the GAM model is applied to the South African context. The South African context is peculiar because in this environment a software developer is not necessarily a specialist in one area such as systems analysis or architecture or programming et cetera. The South African developers are typically multitasked individuals who normally fill more than one role in a project. Therefore of interest would be issues such as how does an organization provide training for such individuals? One of the reasons for multitasking is that the sizes of most software development companies in South Africa can be classified as small with ten or less developers, which makes it difficult to fill the roles and responsibilities in the methodologies except by assigning more than one role per individual.

The use of an analysis model like the GAM would make it possible for the team to select only those practices that can sufficiently cover the project at the same time meeting the team's abilities.

Most agile methodologies come without a set of implementation tools. A nonspecific tool approach ties in well with the recommended adoption of open source software that the government of South Africa has put into operation. This reduces the costs associated with training, which can be unjustifiably high for small companies.

As part of the research presented in this thesis an interesting model for software development team-building was discovered in a study of one small (less than ten developers) organization. The company is based in Zimbabwe a neighboring country to South Africa. The managing director who develops software himself and has a Java and Lotus development background believes in the principle of cultivating a

software developer–family model to strengthen the team structure. He believes that the team must interact very well socially fine language, extra curricular interests etc. Therefore his company spent a lot of time during its initial years developing trust and what he calls the right attitude. Now he says that the team spirit is so contagious that even university students who join the company temporarily (six months to one year) for industrial attachment get so hooked to this family of developers that they would want to join the company after completing their studies.

One peculiar thing about this company is that while the business language is English, its working culture is such that members of the team communicate in both English and other home languages. This means that the internal team dynamics are predominantly implemented in their indigenous languages. While this situation may sound highly unusual and certainly undesirable, the MD believes that it is the secret to their success. He however acknowledges that if a developer of a different culture would join the team there would be a lot of learning to be done and the internal dynamics of the team would be compromised. In its present situation the company is doing well in their country, however, the company is in the process of establishing their business in South Africa and their model will face dire challenges. In South Africa because of the ills of the previous regime, the black majority are not in a position to profitably form software companies that are based on cultural groupings. The main reason being that while the internal dynamics of the development teams would be strengthened the external dynamics would be highly compromised as most of the business would be coming from the other formally privileged cultures, and the political demeanor would certainly be misdirected. Hence the issues of languages and cultures in the South African community are very important, in fact so important that they could affect the future strength and capability to develop world class software products. The battle is not an easy one; the South African Government is tackling this issue at the level of education institutions.

What all this means is that whatever development practices are adopted in the South African environment, they must strongly cultivate a social culture that is cognizant of the diverse cultures and technical backgrounds of all South Africans. The GAM model is one such framework that can be used to tailor agile methodologies to this peculiar environment, (Mnkandla, Dwolatzky, and Mlotshwa, 2005).

5.5 Framework Reasoning Process and Knowledge Contribution

This section summarizes the framework to provide a graphical illustration of the reasoning process followed in its application. The specific contribution to the body of knowledge in agile methodologies is also listed for clarification.

The framework reasoning process is illustrated in Figure 5.1.

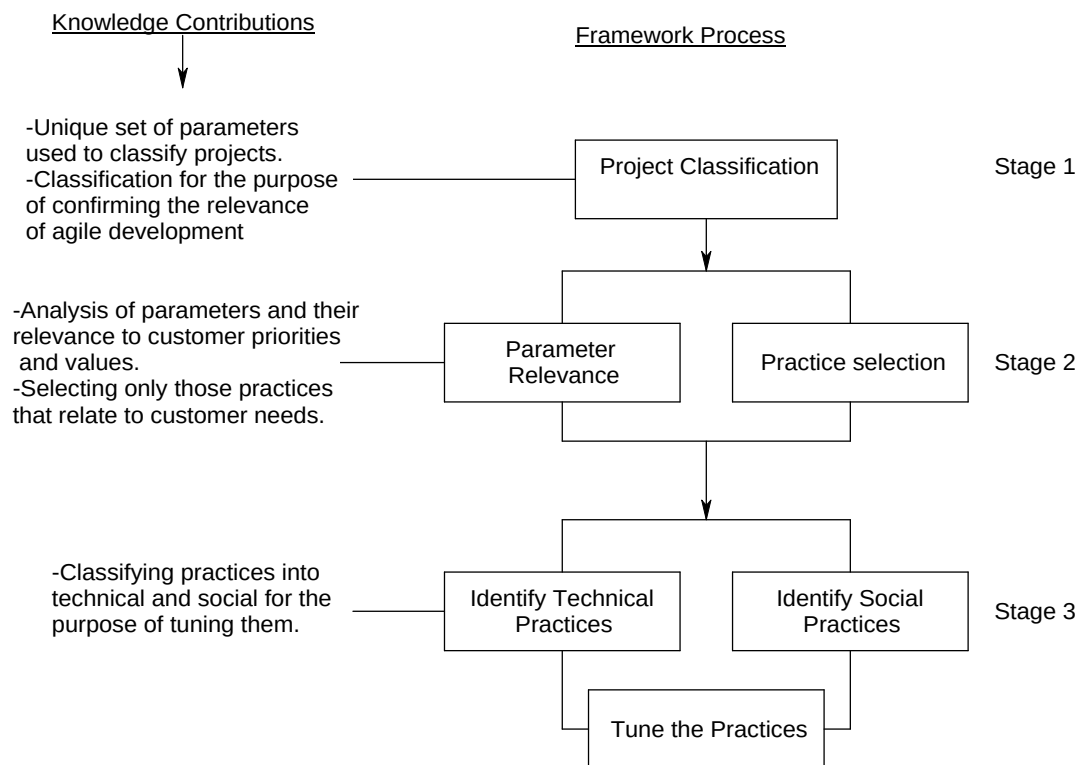


Figure 5-1: Framework Reasoning Process Graph

The main contribution to knowledge is the development of an agile methodologies selection framework that is composed of three main parts:

- A project taxonomy matrix –which uses a unique set of project parameters
- A methodology taxonomy matrix that matches parameters of a methodology to a given software development project and selects methodology practices.
- A novel Generic Agile Methodologies (GAM) model for the tailoring the selected practices to a given environment.

5.6 Conclusions

This chapter gave a case based illustration of this research’s major contribution to knowledge. This case is based on a project in which the author was involved. The

project later on failed to continue due to lack of a sponsor and executive buy-in. The three stages of the framework were explained at length using data from the case. Clear steps were used to illustrate practical activities that can be performed in implementing the framework.

The next chapter describes the research approach applied in theory building and partial validation of the framework proposed in this chapter.

6 Research Methodology and Validation

This chapter describes the research approach used in the process of design and validation of the Agile Methodology Selection Framework.

6.1 Qualitative Research Method

According to Myers (1997) “qualitative research involves the use of data, such as interviews, documents, and participant observation data, to understand and explain social phenomena”. There are many varied approaches, methods and techniques that are used in different fields.

Qualitative research helps researchers to understand the social and cultural contexts of people (Myers, 1997). Published research in software development reveals that the neglect of the social and cultural aspects in software development processes is the main cause for project failure. Using qualitative research in the area of software development therefore provides a better understanding of the social and cultural contexts of software development. Using textual data in this context would limit the understanding to a certain extent (Kaplan and Maxwell, 1994).

The underlying assumptions of qualitative research are based on three philosophies: positivist, interpretive, and critical theories, according to (Chua, 1986; Orlikowski and Baroudi, 1991 cited in Myers, 1997). There are other suggestions on the underlying philosophies see (Guba and Lincoln, 1994). Positivism involves testing of a theory in order to increase the predictive understanding of phenomena (Myers, 1997). Interpretive research attempts to give understanding to the meaning of observable facts (Boland, 1985). The observable facts in the case of an information system include its context and the processes of influence between the context and the information system (Walshman, 1993). Critical research aims to reveal and help

eliminate the oppositions, conflicts, and contradictions in contemporary society (Myers, 1997).

The researcher's assumptions depend on the problem and the proposed solution in a given environment. These assumptions that form the epistemology of the research which could fall under any of the three philosophies defined here. There can be situations as in this study where the assumptions can stretch over more than one epistemology. In this study positivist and critical theories were used.

The choice of the qualitative research method –*strategy of inquiry* used should influence the way the researcher collects data (Myers, 1997). There are many research methods that can be used in qualitative research such as action research, case study, and grounded theory. In some cases more than one research method can be used.

Qualitative research also includes many research instruments according to Myers, (1997) “these techniques range from interviews, observational techniques such as participant observation and fieldwork, through to archival research. Written data sources can include published and unpublished documents, company reports, memos, letters, reports, email messages, faxes, newspaper articles, and so forth”. This research used interviews, questionnaires, participant observation, and email messages.

6.2 The Research Approach

As outlined in Chapter One of this thesis the research project was oriented toward the following activities:

- Studying methodologies of software development
- Collecting observations from literature on past projects

- Developing a framework for the selection of a set of agile practices relevant to a given project
- Conducting opinion surveys on the practicality of the proposed framework through questionnaires, emails and discussions.
- Improving the framework based on the contributions of the domain experts.
- Partial validation of the revised framework through interviews.

The first two activities were presented in Chapter Three of this thesis. The third activity was presented in Chapter Four of this thesis. The last three activities are presented in this chapter.

The main research approach used was qualitative. The reason for the use of the qualitative approach is because of the mix of social and technical phenomena in software development methodologies. As will be explained in this chapter the philosophy of the research combined positivist and critical theories. The research method was mainly grounded theory and the tools used were interviews and questionnaires.

Unlike other sciences and engineering fields, software engineering's main product is software whose structure is different from buildings, bridges, engines, aircrafts, industrial machines, chemical plants, mining equipment and computers. Likewise the control and management processes applied in software engineering are different from those applied in other engineering fields. Software engineering is implemented through balancing between the social activities and the technical bolts and nuts of crunching code and data hence the most relevant research approach to this field is a qualitative one as opposed to the quantitative methods. Some have also found that a hybrid of various types of methodologies brings more value to the research results, (Cockburn, 2003).

Armour (2004) believes that the entire discipline of software development is being viewed wrongly. He says that software development is not a product producing

activity because the true product is not the software but the knowledge contained in the software. He therefore considers software development to be a knowledge acquisition activity. According to Armour (2004) it is only the coding part, which is a transcription of knowledge into executable form, which gets easier with the level of certainty of the knowledge. He then concludes that real work in software development is in the search for the correct knowledge, and therefore our processes and expectations must not be centered on the creation of a product (delivery of the system), but rather on the process of knowing what has to be done, what the system should do and how to construct a system that can do that.

The overview of the research process is shown on Figure 6.1.

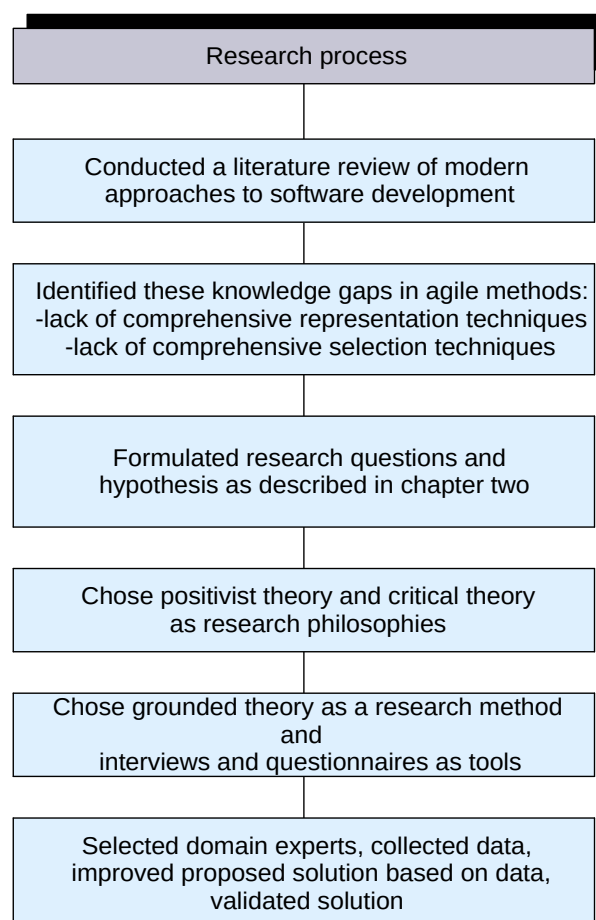


Figure 6-1: Graphical Overview of the Research Process

As summarized in Figure 6.1 the study began from an initial selection of software development methodologies which are a branch of software engineering. The study was specifically focused on agile methodologies. The research work started by the identification of knowledge gaps both from literature reviews of published material on agile methodologies and observations from participating in a software development project. The following knowledge gaps were identified:

- A lack of comprehensive representation techniques for agile methodologies.
- A lack of comprehensive techniques for selecting appropriate agile methodologies in a given project.

These knowledge gaps led to the formulation of research questions as described in chapter two of this thesis. In developing a solution to the identified problems the initial framework was sent out to some domain experts in an attempt to test the theory in order to increase the predictive understanding of phenomena. The feedback was used to improve the framework. This is a positivist epistemology according to Myers (1997). After improving on the framework, the improved framework was given to some domain experts to test and the results are recorded later in this chapter.

6.3 Rationale for Choice of Approach

The proposed agile methodologies selection framework is targeted towards assisting IT professionals faced with a software development project and in need of mapping out a process from the group of agile methodologies currently on the market. This model contributes knowledge in the area of software development methodologies.

Validation of such a model can be quite a challenge and some of the ways of validating it are described below.

One approach would be to select a number of organizations that develop software and have either the innovative culture to try agile methodologies or are already using them, and then ask them to try using the proposed GAM model on a set of projects and let other projects follow the usual approach as a control. Results would then be

collected from both groups and comparatively analyzed and tested to determine success (as defined in Chapter Two of this thesis) for the projects that used the GAM model. This way is somewhat a challenge as it depends on many issues such as getting organizations that would be willing to experiment with real world projects. Software projects sometimes fail because of company politics (company decides the project is not a priority at that time), and software projects can take many years to complete hence rendering the results too late for the duration of PhD research.

Another way would be to put many methodology experts in one room and give them the GAM model and ask them to evaluate it and come up with an agreed approach. This method of validation is practically not possible.

The approach that was adopted in this thesis was to select a number of methodology experts knowledgeable on software development methodologies and software development processes and some IT practitioners and ask for their opinions on the practicality proposed GAM model. The collected opinions were then reviewed and the model was improved based on the comments. The limitation of this approach however, is that each methodology expert uses some methodology in his or her practice and would therefore give an opinion that is biased towards the methodology they use. To balance such opinions the approach defined above could be used where domain experts are brought together and given the proposed GAM model and asked to analyze it and give consolidated opinions. The problem again as stated above is that while a situation like this would give a more balanced opinion, it is practically impossible.

The initial survey resulted in the improvement of the framework which was taken to some domain expert for validation and the results are reported later in this chapter.

6.3.1 Choice of Methodology Experts

At the beginning of this research work back in 2003 there was little published academic literature on agile methodologies. Most of the literature on agile methodologies was for the purposes of marketing particular agile methodologies, (Mnkandla and Dwolatzky, 2004a). It was therefore difficult to understand most agile methodologies in detail. The researcher then resorted to directly contacting the agile methodology authors by email to ask specific questions about the methodologies. This approach proved to be very effective because of the cooperation received. When the need arose to contact methodology authors about the proposed methodology selection framework the principle for choosing methodology experts was based on the following characteristics: design knowledge in software development methodologies, experienced in the use of agile methodologies, interested in improving the agile development process, and interested in research in software project management. It was from such a perspective that methodology experts were chosen and the individual agile methodology authors were the first choice. The researcher believes that agile methodology authors are the most relevant domain experts who can contribute constructive analysis of the proposed methodology selection framework. The assumption was that having struggled with authoring methodologies before the methodology authors would be in a position to identify shortfalls of innovative tools that intend to add value to agility such as the framework proposed in this study.

6.3.2 Data Gathering

Data collection was done in two ways; the first was to send emails to methodology experts with the proposed framework and the questionnaire as attachments. The second way was through interviews. The interviews were contacted in a less formal approach in order to put the interviewee in a comfortable position. The interview sessions were structured in an open –ended discussion format.

Interviews

Where possible at least two people from the team, would be interviewed. The roles considered for interviews are: developer, team lead, project manager, or process designer. The developer and the project leadership provide a balanced perspective of the project as the leadership describes the project according to what they expect and wish to happen. Hence they give a smooth story about the alleged process in action. The developer, on the other hand, views the project from a perspective that emphasizes on getting things done and not just mindless following of organizational process models.

Typical Interview Questions

These are the questions that were asked at each interview (adapted from Cockburn, (2003)):

Give a brief history of the project: timeframes, changes in project size, problem domain, technology used, key events in the life of the project, the general approach.

Mention things that worked in the project.

What things did not work well?

If you were going to start another project, or give advice to someone starting a similar project, what would be your prioritized recommendations?

If you were going to start another project, or give advice to someone starting a similar project, what would you avoid, or keep from happening?

How would you prioritize the tools you used?

Do you think all the tools used in project were necessary?

Do you have any issues about the project?

These types of questions basically elicit information about the way or methodology that was used to work on the project. This is elicited without specifically using the word methodology because once that is mentioned people tend to be say what they think you would like to hear.

Questionnaires

The sample of the questionnaire is given in Appendix D. The questionnaire was basically designed to elicit information about the current use of software development methodologies in South Africa. Leading questions were avoided. The major focus of the questionnaire was to capture the realities of software development practice, not the ideal practice that people know should be done and somehow fail to do.

The following questions were asked and the complete description of the proposed methodology selection framework was attached to the sent emails:

1. Apply the proposed framework in at least three past projects and at least one current project if any.
2. Analyze the results say what would have been done differently in the project if this tool had been available for the past projects. For the current projects what will be done differently because of what the tool has availed?
3. Remember this framework was designed to be practical do you think it is practical?
4. Any other comments about the tool.

6.3.3 Results

Table 6.1 shows the list of domain experts that were contacted via for the opinion survey of the framework. Table 6.2 describes the domain experts that were interviewed. Table 6.3 summarizes the results and findings.

Table 6-1: List of Methodology Experts Contacted

Informant Number	Name	Job Title	Organization	Focus Area	Methodology Authored
1	Robert Glass	CEO, Author and Consultant	Computing Trends	Software Quality	Authored more than twenty books on Software Quality issues
2	Jutta Eckstein	Principal Consultant	Objects in Action	IT Project Management	Book on agile project management for large projects
3	Tom Poppendieck	Senior Consultant	Poppendieck LLC	Application of Lean Thinking in Software Development	Lean Development (LD)
4	Scot Ambler	CEO, consultant, author	Ambysoft Inc.	Software Development Consultancy	Agile Modeling (AM), author of four books on Modeling

5	Jim Highsmith	Sr. VP & Director Agile Software Development	Cutter Consortium LLC	IT Project Management and Consultancy	Adaptive Systems Development (ASD)
6	Kent Beck	Founder and Director of Three Rivers Institute.	Three Rivers Institute.	software development	XP, xUnit family of testing tool, XP books.

Table 5.2 shows the list of IT professionals and practitioners that were contacted for the opinion survey of the methodology selection framework.

Table 6-2: List of IT Professionals and Practitioners Interviewed

Informant Number	Name	Job Title	Organization	Focus Area
7	Fazel Mayet	CEO, Consultant	Psybergate	Software Development Consultancy
8	Barry Myburgh	Managing Director and Principal Consultant	Insyte Information Engineering Systems	Consultancy in Business Process Innovation and IT Project Management
9	Sifiso Mlotshwa	Managing Director	eSolutions	Software Development
10				
11				

The numbering between Tables 6.1 and 6.2 is continuous to make the relationship with Table 6.3 simpler.

The results of the evaluation process are shown on Table 6.3.

Table 6-3: Framework Evaluation Results

Informant Number	Is Model Acceptable?	Positive Comments on Model	Challenges about the Model
1	Yes	Tying together of project and methodology taxonomies in order to select a methodology.	Clarify on the link between the two matrixes.
2	Yes	Tacit knowledge may not necessarily relate to experience.	Overview of the methodology selection should include knowledge and experience of the project members. Every member of the project team is an expert in agile projects. Selecting a methodology is only the beginning, to be followed by adaptation, hence the need for project retrospectives.
3	Yes	Before 2004 era selecting specific methodologies was the main focus. Selecting specific agile methodologies	Consider selection of agile practices in general. Agile when done well will always

		should only be a starting point.	outperform rigorous. Chaotic development of software is related to rigor especially when projects go over budget and time. Discipline is related to agile not in the sense of following a plan, but in the sense of skillful, thoughtful application of appropriate practice and skills.
4	Yes	Consider extending the methodology selection matrix in Table 4.2 to include partial methodologies.	Include more references on agile modeling to give an agile perspective on complex projects.
5	Yes	Agile methodologies are more useful when there are areas of uncertainty to be explored. Decision whether or not to try agile is greatly influenced by culture.	Clarify the purpose of the project classification matrix Table 4.1. Requirements risk should also include technology risk. Consider a project where there is high risk (needs agile) and very complex

			(needs rigor).
6	Yes	The first version of XP was rather prescriptive and people took XP far beyond the initial intentions. The current approach inline with this model is to less prescriptive and leave the specifics to be determined according to an organization's needs and project environment.	Practical application would bring a lot of value.
7	Yes	Deep analysis of the nature of software projects.	Consider including customer priorities and risk in the project classification matrix. Include detailed definitions of major terms like complexity, criticality of a

			software development project.
8	Yes	Similar in principle to the Situational Process Model.	Consider the point that each situation even within one project may need various processes. More control in a project is more of an organizational strategic issue than a technical issue.
9			
10			
11			

Table 6.2 shows the survey on what domain experts thought about the proposed methodology selection framework. In the next section the analysis of the data received from the domain experts is done.

6.3.4 Analysis

The process of data analysis was iterative between examinations of positive comments and the development of further theoretical interpretations. Some of the comments gave insight into the problem under study and revealed innovative perspectives on how to improve the limitations of the model as highlighted in the comments.

In the hermeneutic process of qualitative research, analysis of informants' individual statements represented the parts while the evolving conceptual framework represented the whole, (Klein and Myers, 1999). In agreement with the principle of dialogical reasoning (Klein and Myers, 1999; Goubil–Gambrell, 1991), prior research literature informed the analysis but a variety of other theoretical interpretations were considered in the course of the analysis. Dialogical reasoning facilitated the emergence of innovative ideas, which eventually combined into a set of coherent themes that were linked with, but not constrained by, prior theoretical frameworks, (Pawlowski and Robey, 2004). These themes comprised the elements of the proposed framework that resulted from the analysis.

The recommended practice in qualitative research according to Mason (2002), was carried out, that is, data analysis was conducted in parallel with data collection in order to allow the two processes to inform each other. The goals of the initial data analysis were to attain an understanding of the initial planning activities that lead to the selection of software development approaches. First, the interview data was written down to identify the flow of ideas and unfolding of development processes.

Second, the data would then be used to generate reasoning graphs and mind maps as brainstorming tools for the birth of new ideas and insight.

As a result of this kind of data analysis the proposed methodology selection framework was improved upon progressively from the initial version that was sent out to domain experts sometime in January 2005 to the final version. The challenges posed by each of the domain experts were meticulously analyzed and compared to the overall theme of this research.

Table 6.4 summarizes the improvements made on the initial version of the model in response to the comments made by the domain experts.

Table 6-4: Changes Made to the Initial Version of the GAM Model

Challenges about the Model	Changes made
Clarify on the link between the two matrixes.	Sections 4.3 and 4.4 were changed to clarify the relation since they constitute phase one and two respectively, of the overall framework. Phase one leads to the classification of a project either as an agile or traditional project.
Overview of the methodology selection should include knowledge and experience of the project members.	This was included in section 4.4.11 on the role of the human expert.
Consider selection of agile practices in general.	Including this change presented a fundamental contribution to the body of knowledge in software development, the selection of agile practices as opposed to selecting particular methodologies.

Include more references on agile modeling to give an agile perspective on complex projects.	This changed extremist views that considered agile methodologies as applicable to small projects.
Clarify the purpose of the project classification matrix Table 4.1.	The rationale is that there are projects that still need more traditional approaches to development.
Consider including customer priorities and risk in the project classification matrix.	This was done in section 4.3.6

6.4 Validation Approach

The framework was presented to a software development company for partial validation. The process of validation involved an initial description of the framework by the researcher. Use was made of figure 4.1 which illustrates the three stages of the framework. The matrixes and tables that correspond to each stage of the framework were explained in detail as they are important in the practical implementation of the framework. The stages and corresponding matrixes are summarized in Table 6.5 for clarity.

Table 6-5: Top-level Summary of the Framework

Stage	Brief Description	Purpose of stage	Matrixes and Tables
One	Software Project taxonomy.	Determine the suitability of agile development in a project.	Table 4.1
Two	Methodology and practice selection.	Select the relevant agile methodology practices for a given project.	Figure 4.2, Table 4.2, 4.3, and 4.4
Three	Practice tuning	Adapt the practices to the given environment.	Table 4.5

The IT professionals interviewed for the validation of the framework are described in Table 6. 6.

Table 6-6: IT Professionals Interviewed

Name	Job Title	Organization	Focus Area
Dimitri Vratsanos	Director, Operational Excellence	Psybergate	Software Development Consultancy

Typical Questions

The questions asked during the interview are summarized below.

- Using past or present software development project data list the most important requirements and priorities and values. This was focused on getting an understanding of customer values and priorities for the project.
- Go to stage one of the framework (framework process diagram and matrix tables presented) and use your information to determine the suitability of agile development in the project. The aim was to determine the relevance of using agile development in this project.
- Did the use of the project taxonomy matrix improve your understanding of the project with regard to process issues? The aim was to evaluate the contribution of the framework to the understanding and use of agile methodologies.
- Go to stage two of the framework and assume that the previous stage indicated applicability of agile development. Apply the methodology selection matrix and select the set of practices that are relevant to your project. Did the use of the methodology selection matrix improve your understanding of the set of activities expected in your project? The aim of this question is to evaluate the framework's contribution to knowledge.

- Go to stage three of the framework and classify the practices into social and technical. Consider what needs to be changed in each practice in order for it to be practically usable in your project environment.
- Given the opportunity would you use this framework in planning your next software development project?
- Would you recommend the use of this framework to other IT professionals?
- Comment on any notable limitations of the framework.

6.5 Results and Analysis

The results of the interviews are summarized in Table 6.7, for the detailed validation document received from the company with responses to various questions see Appendix E.

Stage One of the Framework

With regard to the questions that relate to stage one of the framework when applied to some past projects the project matrix was found to be useful. The comment was:

“I think the factors you present contribute to making a decision. There may be an additional factor in the way of commercial constraints. In my experience consumers of software development services (whether the IT organization providing the services is internal or external to the enterprise) are adamant of the need for an upfront fixed cost and fixed time commitment.”

Stage Two of the Framework

At stage two of the framework Tables E4.2 E4.3 and E4.4 (in Appendix E) give the detailed use of the selection matrix.

The developer found the selection stage to be “helpful in making explicit the relevance of an approach in a particular context”

Stage Three of the Framework

Applying the tailoring techniques in Stage Three of the framework provided the following results for the project used:

Planning game: Social (is practically usable in vanilla form – use use-cases rather than stories)

Whole team: Social (is practically usable in vanilla form – sometimes have to compromise on customer on site – poses considerable risk)

Informative workspace: Social (is practically usable in vanilla form – sometimes compromised by distributed work force)

Weekly cycle: Social (is practically usable – found 2 weekly cycles better)

Ten minute build: Technical (practically usable through use of build tools)

Continuous integration: Technical (practically usable)

Test first programming: Technical (practically usable)

Incremental design: Technical (practically usable – some broad based design needs to be done upfront to establish broad scope – determined largely by client need for upfront commitment re cost and time)

Real customer involvement: Social (practically usable)

Shared code: Social (practically usable)

Daily deployment: Social (practically usable)

In general the framework was found to be helpful and the participant expressed willingness to use it in future projects if given the opportunity and was willing to recommend it to other developers. However, the participant believes that the framework defines extensive diversity in projects and may therefore find minimal use in organizations that focus on ‘niche software development’.

Asked to comment on possible areas of improvement on the framework the following points were given:

1. Presentation of the framework and ease of use could be improved

2. Consider more fine grain determinants of project description
3. Make the practices themselves the target of choice (rather than methodology)

6.6 Discussion

The findings of this research provide a detailed account of the role played by software development professionals in choosing approaches to development. The primary activity of the selection process is to analyze the project and get a detailed understanding of the nature of the project and hence get a feel of the suitable process for that situation including the priorities and values imposed by the customer. There is always room to apply knowledge and experience acquired from previous projects. There is also a need to understand methodology representations. That is, in order to correctly select a methodology it must be properly defined and classified. This is where reviewed literature revealed a gap in agile methodologies. Therefore, in this research a framework for the comprehensive representation of agile methodologies was developed.

With a better understanding of agile methodologies (confirmed at the XP 2005 conference in Sheffield June 18 to 23 2005), it became clearer that individual agile methodologies are in fact very similar they only differ in the metaphor used to view reality. This is an important fact as it leads to a conclusion that there is no need to select a specific agile methodology for a given situation. The recommended way to adopt agile methodologies according to this study is to select the most relevant practices from the agile family. This is the emphasis of the proposed methodology selection framework described in Chapter Four of this thesis. The issues of value in the software development process are to do with; the overall organizational strategic focus in relation to software development, the general attitude of the entire team towards the work at hand, the general and specific goals of the project, and the customer values. This leads to an important principle that of being professional and

honest without compromise. Agile methodologies tend to emphasize these principles as primary to the success of any project.

The proposed GAM model is not just another methodology; in fact it is not meant to be a methodology but a thinking framework. It is a tool that helps software development experts to transform from doing software development in the confines of the traditional meaning of ‘methodology’ which presumes that there exists one best way of doing a task and that the expert can describe the way, (Poppendieck, 2005). The proposed methodology selection framework therefore introduces a more liberal perspective (liberated agility), which is to say:

- Allow the members of the project team to understand tasks that must be done
- Project leaders (managers) must provide the tools, training, skills, and political climate (with regards to the general organizational strategies) needed to reach the goals.

6.7 Evaluation

Self-evaluation of conformance to established principles is rather imperative in interpretive research approaches. It is however a bigger challenge to evaluate research approaches such as the one used in this study.

Whether domain experts or not, when system developers are given some specifications and asked to design a system they would normally come up with different systems that may meet the specifications but with emphasis on different issues. What determines the differences in the emphases would be rather subjective. Despite this challenge it remains true that experts in a given domain will always be able to identify common traits in the processes that define their day-to-day activities. It is from such a perspective that the opinions of the software development and methodology experts contacted in this research have been considered to amount to partial “validation” of the proposed GAM model.

The principle of suspicion compels researchers to doubt the respondent's reports as superficial and to probe more deeply for underlying issues, (Pawlowski and Robey, 2004). So to deal with problems associated with the principle of suspicion where possible discussion sessions were done with those domain experts that the researcher could meet with in person both locally (in South Africa) and internationally.

6.8 Conclusion

The trend in the adoption of agile software development methodologies is towards the adoption of a set of practices (as detailed in Table 3.2) relevant to a given situation rather than dogmatic adoption of particular agile methodologies. Surprisingly even some of the authors of certain agile methodologies have become less prescriptive about the choice of methodologies, for example the second edition of Kent Beck's Extreme Programming there is a higher tone of generality which has seen the dropping of the XP theme of taking everything to the extreme, hence allowing practitioners to tune the methodology to their specific environments. However, what was still lacking is a comprehensive technique to perform the selection of best practices. This is where the proposed framework fills the gap by providing a novel tool for selecting and tuning a set of practices that are relevant to the given environment. Taking into consideration the volatility of the requirements in contemporary business systems the proposed framework further suggests a selection and tuning criteria that is as much as possible confined to the boundaries of an iteration and is reapplied as the system grows from iteration to iteration.

7 Summary and Future Work

In the previous chapters an evolution of software development practices was introduced, from early “chaos” through current efforts in methodology “wars” as reflected through the Agile Movement. A motivation of the need for a structured methodology selection framework was defined, and a comprehensive agile methodology selection framework with all its components was proposed as a solution approach. Finally, a qualitative study giving partial validity to the proposed framework was presented. This body of work is shortly summarized and an opportunity for future work related to the proposed GAM model is presented. Limitations of this research work are also outlined here.

7.1 Summary

This work was motivated by the findings through detailed research on agile methodologies and some experience from discussions at South Africa Computer Society and Project Management South Africa public meetings where a mere mention of the term ‘agile methodologies’ would be met with mixed feelings: some showing resistance and criticism, and others (especially those who were using agile methodologies) showing satisfaction and process pride. From the review of literature on agile methodologies the challenge was to find out why so many software developers were not comfortable with agile methodologies despite their success as evidenced by published literature. After further literature research and observations it became clear that agile methodologies were surrounded by a lot of uncertainty because of a lack comprehensive representation techniques that could define in full what agile methodologies are and what they are not, what they are capable of, and how they could be adopted to the competitive advantage of an organization. This study revealed that little work had been done in trying to address issues of project taxonomy for the purposes of matching a methodology to a project. To this Glass (2005), in an email communication confirmed that the researcher is trying to address

a problem that he (Robert Glass) had posed for more than a decade; that of taxonomizing methodologies and taxonomizing projects in order to match methodology and project.

Therefore a framework was developed composed of three main phases; the project classification phase whose main purpose is to determine the suitability of applying agile development to a given project, the methodology classification and selection phase whose main purpose is to represent methodologies and select relevant practices, and the Generic Agile Methodologies (GAM) model whose main purpose is to classify the selected practices into social and technical groups and tailor them to a given environment. The original idea behind the development of this novel model – the GAM, was to come up with a thinking tool that could be used to tune a methodology to a given environment. However as more data became available through conferences, discussions and literature review it became clear that the trend in agile methodologies had shifted from a concern for selecting particular agile methodologies to a more general level of selecting relevant practices from wherever. It was at this point that it dawned that agile methodologies were in fact not that different from each other but implemented the same practices using various terminology, for example what XP calls a standup meeting is called a “scrum” in Scrum (because the fundamental principle in both cases is communication), and what XP calls a user story is in fact a feature in FDD. A gap was therefore realized in the availability of tools to assist IT professionals in selecting a set of practices relevant to a given situation from the agile family of methodologies or even any other methodology for that matter. Then the proposed GAM model was improved to cover this need.

7.2 Knowledge Contributions

The study addresses the problem of choosing the most suitable agile methodology practices and tailoring them to a given environment. This section describes the contributions of this thesis.

Agile Methodologies

The major contributions of this thesis to the area of agile methodologies are:

- A comprehensive framework for selecting the most suitable agile practices in a given environment was designed. The selection process is unique because it represents agile methodologies using a comprehensive set of parameters that define a methodology. The set was derived from a number of existing frameworks.
- A complete tool for tailoring agile methodologies (the GAM model) was designed. The GAM model is unique because it classifies all development practices into technical and social activities and specifies how to tailor a given practice into a given environment.
- The understanding of agile methodologies is taken to a higher level by defining each methodology from the perspective of the parameters that represent a methodology, the practices that are implemented in each methodology, and the value of the practices with regard to the needs and constraints of the customer.
- The framework in general encourages the application of agile methodologies as a unified process rather than a many related methods.

Project Management

The contributions of this thesis to the area of project management are:

- A comprehensive technique for classifying software development projects. The technique uses a unique combination of parameters to fully define a software development project. This technique of classifying software projects opens new horizons of analyzing and critiquing methodology choices.

- The detailed analysis and simplified classification of the agile practices into social and technical provides specific tailoring techniques for the agile practices.

7.3 Future Work

The proposed GAM model is designed to accommodate any existing agile methodology and those that may be authored in future, which means that it is an open ended framework. The future work could involve the expansion of the proposed GAM model into a more comprehensive framework that can handle other methodologies that are not agile such as the Rational Unified Process (RUP), Waterfall model etc. Such work would certainly involve a change of the major phases of the proposed GAM model and their themes. A concern is however that there is a potential of overloading the scope of the proposed GAM model, which may lead to yet another methodology, which certainly is not a desirable result according to the original intent of this research. Other points of importance for future work are:

- The success enjoyed by agile methodologies would not be possible if the individuals involved were not both highly committed and highly skilled. When we assume that agile teams will do their work without a heavy process acting as a schoolmaster we are assuming that they are highly committed and highly skilled. May be such a setup is only applicable to certain types of individuals. Investigating such issues should bring interesting results the future application of agile development.
- How the proposed GAM model can be applied to partial agile methodologies.
- Finding out what happens after an organization successfully adopts agile practices, may be surveys into those organizations that have been agile for many years would be a good start. A review of the agile aftermath.
- Many projects require aspects of both agility and rigor to be successful (Highsmith, 2005). It should be challenging to explore the possibilities of

applying the proposed GAM model to the selection of practices from both agile and rigorous methodologies.

The Generic Agile Methodologies (GAM) Toolbox

In an effort to assist organizations and individuals interested in adopting the proposed GAM framework a toolbox has been designed (top level design). The intention is to provide prospective users with an application that can be installed in a computer so that the user can enter information about their project and get a recommended set of agile practices. Appendix F shows the screen dumps of the toolbox. Note that these are just screen dumps. The toolbox is still future work and is not part of this thesis.

7.4 Limitations

Software engineering is a highly complex discipline in a different way from other engineering disciplines. It is not about complex mathematical formulas and is not about complex design diagrams for building complex multistory structures. It is about discovering behavior patterns in human beings and then utilizing those patterns to collaboratively produce knowledge, which can be translated into executable code. In the animal kingdom behavior patterns can be reasonably predicted but not so among human beings, at least not in software building activities. Herein lies the main limiting factor in this study: what may have worked for one team of software developers may not necessarily work for another team.

7.5 Lessons Learnt

This section summarizes the lessons learnt during this study.

People and Processes

This study revealed two perspectives to the challenge of developing software, the developer's side and the customer's side. On the part of the software development organization people should be given more value than processes. This phenomenon

has been raised by many methodologists, however a further point that was revealed is that for now people cannot do without processes and processes cannot do without people. This status is likely to continue until we create intelligent systems that can properly assume the role of a software developer. Imagine an environment where we do not need to worry about people issues. If only machines could develop the software because they do not get hungry, or have sick relatives, or have family duties to attend to, they do not go on leave, and they do not have emotions for now.

On the side of the customer an important lesson was that outsourcing software development to an agile developer carries with it more responsibility than in the traditional approaches. It is in fact more of a corporative game between the developer and the customer than outsourcing because the customer is required to be part of development team. This setup also implies that project risk is shared between the customer and developer, which is different from the traditional approach where the customer has to pay for any risks involving requirements changes.

Agile Development

The value of people and process in software development forms the main difference between agile development and traditional development. In agile development trusting people and believing that people will get the work done is the fundamental principle. The corollary is that the people must be highly skilled and highly committed. The irony is that process was introduced because of an understanding that people need specific guidelines of what should be done.

Traditional process-centered development approaches trust that following a given process will get the work done. There is also the issue of measurability and documentation whose implementation is important for quality but makes traditional development rather heavy. Therefore the lesson here is that for many projects a bit of both approaches is required because a people centered approach is still a process in any case, so we cannot get rid of process in software development. There are some

projects that may be on the agile extreme and others which may be on the process-centered extreme. The wisdom in software development lies in identifying on which side the project belongs and striking the balance.

Organizational Maturity

As an organization repeatedly develops software there is a general tendency to systematize the repeatable activities into standard processes. The organization will normally learn from its previous projects and improve its processes into a culture of development. The danger of culture in software development is that people may end up not questioning it, which may lead to lack of innovation. Process pride may also result which limits dynamism. However, organizations can still mature and reach high levels of process improvement and optimization by maintaining a balance between people issues and process issues.

8 References

Abrahamsson, P., Warsta, J., Siponen, M.T., and Ronkainen, J. (2003), New Directions on Agile Methods: A Comparative Analysis, *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*.

Abrahamsson, P., Salo, O., Ronkainen, J. and Warsta, J. (2002), Agile Software Development Methods: Review and Analysis, *VVT Publications*, No. 478, pp. 7–94.

Abran, A., Robillard, P.N. (1994), Function Points: A Study of their Measurement Processes and Scale Transformations, in *Journal of Systems and Software*, vol. 25, pp. 171–184.

Agile Alliance, (2001), Manifesto for Agile Software Development, [online]. Available from: <http://www.agilemanifesto.org> [Accessed 02 May 2005].

Ahn, Y., Suh, J., Kim, S. and Kim, H. (2003), Journal of Software Maintenance and Evolution: Research and Practice, Vol. 15, pp. 71–85. [online]: Available from: <http://www.ifi.unizh.ch/groups/req/ftp/kvse/ahn03.pdf>, [Accessed 13 June 2006].

Ambler, S.W. (2002a), Introduction to Agile Model Driven Development, Agile Modeling Essays, *Ronin International Inc.* [online]. Available from: <http://www.agilemodeling.com/essays>, [last accessed 30 May 2006].

Ambler, S.W. (2002b), *Agile Modeling*. John Wiley and Sons, pp. 7–18.

Ambler, S.W. (2003), Agile Database Techniques: Effective Strategies for the Agile Software Developer, John Wiley and Sons, pp.3–18.

Ambler, S.W. (2004), *The Object Primer: Agile Model–Driven Development with UML 2.0*, Cambridge University Press.

Ambler, S.W. (swa@ronin-intl.com) (13 January 2005) *Tool for Matching A Methodology to a Project*. Personal email to E. Mnkandla (Ernest.Mnkandla@infotech.monash.edu)

Ambler, S.W. Nalbone, J., and Vizdos, M.J. (2005), *The Enterprise Unified Process : Extending the Rational Unified Process*, Prentice Hall PTR.

Andrews, D.H, and Johnson, K.R. (2002), *Revolutionizing IT: The Art of Using Information Technology Effectively*. John Wiley & Sons.

Anthes, G., (2003), Agents of Change, *Computerworld*, pp.23–28.

Armour, PG. (2004), *The Laws of Software Process: A New Model for the Production and Management of Software*. Auerbach Publications.

Auer, K. and Miller, R. (2001), *Extreme Programming Applied: Playing to Win*, Addison–Wesley, Reading, Mass, pp. 3–10.

Avison, D.E. and Fitzgerald, G. (1995), *Information Systems Development: Methodologies Techniques and Tools*. McGraw–Hill.

Avison, D.E. and Fitzgerald, G. (2002), *Information Systems Development: Methodologies Techniques and Tools*. McGraw–Hill.

Avison, D.E. and Wood–Harper, A.T. (1990), *Multiview, An exploration in information systems development*, Blackwell Scientific Publications, Oxford.

Basili, V.R., Tesoriero, R., Costa, P., Lindvall, M., Rus, I., Shull, F. and Zelkowitz, M.V. (2001), Building an Experience Base for Software Engineering: A Report on the First CeBASE eWorkshop, *Proc. Profes (Product Focused Software Process Improvement)*, 110–125.

Beck, K. (1999), *Extreme Programming Explained: Embrace Change*, Addison–Wesley, Reading, Mass, pp. 10–70.

Beck, K. and Andres, C. (2004), *Extreme Programming Explained: Embrace Change*, Addison–Wesley Professional.

Beck, K. and Fowler, M. (2000), *Planning Extreme Programming*, Addison–Wesley, Reading, Mass, pp. 20–54.

Beck, K. (2005), Another Notch, invited talk, *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering. XP 2005*, Sheffield, UK. In Baumeister, H., Marchesi, M., and Holcombe, M. (eds.). *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering*. Springer–Verlag GmbH, pp. 201.

Boehm, B. (2002), Get Ready for Agile Methods With Care, *IEEE Computer*, pp. 64–69.

Boehm, B. and Turner, R. (2004), *Balancing Agility and Discipline: A guide for the perplexed*, Addison–Wesley, USA, first edition, Appendix A, pp. 165–194.

Boehm, B. and Turner, R. (2005), Management Challenges to Implementing Agile Processes in Traditional Development Organizations, *IEEE Software* Vol. 22, No.5, pp. 30–39.

Boland, R. (1985), "Phenomenology: A Preferred Approach to Research in Information Systems," in *Research Methods in Information Systems*, E. Mumford, R.A. Hirschheim, G. Fitzgerald, and T. WoodHarper (eds.), NorthHolland, Amsterdam, pp. 193-201.

Bradley, P. (2003), *Agile as Evolution and Exploration*, FDD Website, [online]. Available from: <http://www.featuredrivendevelopment.com/node/view/592>, [last accessed 10 June 2006].

Brandt, I. (1983), A Comparative Study of Information Systems Development Methodologies, *Proceedings of the IFIP WG8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies*. In Olle, T.W., Sol, H.G., and Tully, C.J., (eds.). *Information Systems Design Methodologies: A Feature Analysis*. Amsterdam, Elsevier, pp. 9–36.

Brewer, J. (2001), Extreme Programming FAQ, *Jera Design Technical Information*, [online], Available from: <http://www.jera.com/techinfo/xpfaq.html> [Accessed 02 May 2006].

Brooks, F. (1987), No Silver bullet: essence and accidents of software engineering, *IEEE computer Magazine*, Vol. 21, No. 4, pp. 10–19.

Bryant, A., (2000), Software Engineering: solution to the software crisis, or part of the problem? *Proceedings of the 22nd International Conference on Software Engineering, ICSE 2000*, pp. 78–87.

Cathpole, C. P. (1987), *Information Systems design for the Community Health Services*, PhD Thesis, Aston University, Birmingham, UK.

Checkland, P. (1981), *System, Thinking, Systems Practice*, John Wiley & Sons.

Christensen, C.M. (2003), *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*. HarperBusiness.

Clarkson, D. (2004), The Future is Java, *ITWeb BrainStorm*, November, pp. 48–54.

Cockburn, A. (1997), *Surviving Object-Oriented Projects*. Addison–Wesley.

Cockburn, A. (2000), Selecting a Project's Methodology, *IEEE Software*, pp. 64–71.

Cockburn, A. (2002), *Agile Software Development*. Addison–Wesley.

Cockburn, A. (2003), *People and Methodologies in Software Development*. PhD Thesis, University of Oslo, Norway, UK.

Cockburn, A. (2004a), *Crystal Clear: A Human-Powered Methodology for Small Teams*, Addison–Wesley.

Cockburn, A. (alistair.cockburn@acm.org) (16 April 2004b) *Use of Crystal in Real Projects*. Personal email to E. Mnkandla (Ernest.Mnkandla@infotech.monash.edu)

Cohen, D., Lindvall, M. and Costa, P. (2003), *Agile Software Development, Fraunhofer Center for Experimental Software Engineering*, Maryland, DACS SOAR 11 Draft Version, pp. 11–52.

Collins–Cope, M. (2002a), Planning to be Agile? A discussion of how to plan agile, iterative and incremental developments, *Ratio Technical Library White paper* [online]. Available from: http://www.ratio.co.uk/whitepaper_12.pdf [Accessed 2nd May 2006].

Collins–Cope, M. (2002b), Planning to be Agile? A discussion of how to plan agile, iterative and incremental developments, *Ratio Technical Library White paper* [online]. Available from: http://www.ratio.co.uk/whitepaper_12.pdf [Accessed 2nd May 2006].

Cortada, J.W., (2002), Researching the History of software since the 1960s, *IEEE Annals of the History of Computing*, January to March, pp. 72–79.

Crispin, L. and House, T. (2002), *Testing Extreme Programming*, Addison–Wesley, Reading, Mass, pp. 39–45.

Cronholm, S. and Ågerfalk, P.J. (1999), On the Concept of Method in Information Systems Development, *Electronic Articles in Computer and Information Science*, Linköping University Electronic Press, Vol. 04, No. 019. [online], Available from: <http://www.ep.liu.se/ea/cis/1999/019/> [Accessed 2nd May 2006].

Davis, A.M., Bersoff, E.H., and Comer, E.R., (1988), A Strategy for Comparing Alternative Software Development Lifecycles, *IEEE Transactions on Software Engineering*, Vol. 14, No.10, pp. 1453–1459.

Davison, R.M., (1998), *An Action Research Perspective of Group Support Systems: How to Improve Meetings in Hog Kong*, City University of Hong Kong, Hong Kong.

Dekkers, T., and Vogelesang, F. (2003), **COSMIC** Full Function Points: Additional to or replacing FPA, in *Australian Conference on Software Measurement*, Sydney, Australia. [online]: Available from: <http://www.lrgl.uqam.ca/publications/pdf/819.pdf>, [Accessed 13 June 2006].

Eckstein, J. (2004), *Agile Software Development in the Large: Diving into the Deep*, Dorset House Publishing.

Elssamadisy, A., and Schalliol, G., (2002), Recognizing and Responding to ‘Bad Smells’ in Extreme Programming, *Proceedings of the International Conference on Software Engineering ICSE 2002*, pp.617–622.

Enzenhofer, W., and Chroust, G., (2001), Best Practice Approaches in Know–How and Technology Transfer Methods for Manufacturing SMEs, *Proceedings of the EUROMICRO 2001*, pp. 279–286.

Eriksson, M. (2004), Project Management Skills –Tutorial, *Proceedings of the IEEE AFRICON 2004 Conference*.

Fayad, M.E., (1997), Software Development Process: A Necessary Evil, *Communications of the ACM*, Vol. 40, No. 9, pp.101–103.

Feldmann, R.L., and Pizka, M., (2002), An Online SE Repository for Germany’s SME: An Experience Report, *Proceedings of the 4th International Workshop on Advances in Learning Software Organization, LSO 2002*, pp. 34 – 43.

Fitzgerald, B. (1996), Formalized Systems Development Methodologies: a critical perspective, *Information Systems Journal*, Vol. 6, pp.3–23.

Fitzgerald, B., Nancy, L.R., and Stolterman, E. (2002), *Information Systems Development: Methods in Action*, McGraw–Hill.

Ford, G. (1990), 1990 SEI Report on Undergraduate Software Engineering Education, *CMU Technical Report no. CMU/SEI-90-TR-003*.

Fowler, M. (2002), The Agile Manifesto: where it came from and where it may go, *Martin Fowler article* [online]. Available from: <http://martinfowler.com/articles/agileStory.html>, [Accessed 2nd May 2005].

Fowler, M. (2000), Put Your Process on a Diet, *Software Development* 8(12), pp.32–36.

Fowler, M. (1999), *Refactoring: Improving the Design of Existing Code*. Menlo Park, California. Addison Wesley.

Friedman, A. (1989), *Computer Systems Development: History Organization and Implementation*. Wiley & Sons.

Glass, R.L. and Vessey, I., (1994), Rethinking the Nature Application System Development, *working paper, Pennsylvania State University*.

Glass, R.L. and Vessey, I., (1995), Contemporary Application Domain Taxonomies, *IEEE Software*, pp. 63–76.

Glass, R.L. (1998), Is there Really a Software Crisis? *IEEE Software*, vol. 15, no. 1, pp. 104–105.

Glass, R.L. (2001), The Great Methodologies Debate: Part 1: agile versus traditional: Make love not war! *Cutter IT Journal*, Vol. 14, N0.12, pp.12–18.

Glass, R.L. (2005), IT Failure Rates–70% or 10–15%? *IEEE Software*, Vol. 22, N0.3, pp.110–112.

Goubil–Gambrell, P. (1991), What do Practitioners need to know about Research Methodology? *Proceedings of the IEEE International Professional Communication Conference IPCC91, IEEE Software*, pp. 243–248.

Hansson, C., Dittrich, Y., and Randall, D., (2004), Agile process enhancement user Participation for small Providers of Off–the–Shelf Software. *Proceedings of the Extreme Programming and Agile Processes in Software Engineering: 5th International Conference, XP 2004*. In Eckstein, J. and Baumeister, H. (eds.). *Proceedings of the 5th International Conference on Extreme Programming and Agile Processes in Software Engineering*. Springer–Verlag GmbH, pp.175–183.

Highsmith, J. (1997), Messy, Exciting, and Anxiety–Ridden: Adaptive Software Development, *American Programmer*, Vol. 10, No. 1, [online], Available: <http://www.ksinc.com/articles/CAS.html> [Accessed 2nd May 2006].

Highsmith, J. (2000a), Project Management at the Edge. *The IT Project Leader*.

Highsmith, J. (2000b), Retiring Lifecycle Dinosaurs: Using Adaptive Software Development to meet the challenges of a high–speed, high–change environment, *Software Testing & Quality Engineering (STQE) magazine*, pp.22–28, [online]. Available from: <http://www.adaptivesd.com/articles/Dinosaurs.pdf>. [Accessed 2nd May 2006].

Highsmith, J. (2000c), Adaptive Software Development: a collaborative approach to managing complex systems, Dorset House, pp. 1–20.

Highsmith, J. (2001), The Great Methodologies Debate: Part 1: Today, a new debate rages: agile software development versus rigorous software development, *Cutter IT Journal*, Vol. 14, No.12, pp. 2–4.

Highsmith, J. (2002a), *Agile Software Development Ecosystems*, Addison–Wesley, pp. 1–50.

Highsmith, J. (2002b), *Agile Software Development: why it is hot! Cutter Consortium white paper*, Information Architects, Inc, pp. 1–22.

Highsmith, J. (2002c), What is Agile Software Development? *STSC CrossTalk*, *The Journal of Defense Software Engineering*, issue 10, [online], Available from: <http://www.stsc.hill.af.mil/crosstalk/2002/10/highsmith.html>. [Accessed 18th May 2006].

Highsmith, J. (2004), *Agile Project Management*, Addison–Wesley.

Highsmith, J. (jim@jimhighsmith.com) (25 January 2005) *Tool for Matching A Methodology to a Project*. Personal email to E. Mnkandla (Ernest.Mnkandla@infotech.monash.edu)

Highsmith, J., Orr, K. and Cockburn, A. (2000), *Extreme Programming, E–Business Application Delivery*, pp. 4–17.

Hoffer, J.A., George, J.F. and Valacich, J.S. (2004), *Essentials of Systems Analysis and Design*, Prentice–Hall.

Hofstede, A.H.M. and van der Weide, Th.P. (1992), Formalisation of Techniques: chopping down the methodology jungle, *Information and Software Technology*, Vol.34, No.1, pp. 57–65.

Humphrey, W.S. (1990), *Managing the Software Process*, Addison–Wesley, pp. vii–xi.

IEEE. (1990), IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12.

Iivari, J., Hirschheim, R. and Klein, H.K. (1999), Beyond Methodologies: Keeping up with Information Systems Development Approaches through Dynamic Classification, *IEEE, Proceedings of the 32nd Hawaii International Conference on System Sciences*.

Kaplan, B. and Maxwell, J.A. (1994), "Qualitative Research Methods for Evaluating Computer Information Systems," in *Evaluating Health Care Information Systems: Methods and Applications*, J.G. Anderson, C.E. Aydin and S.J. Jay (eds.), Sage, Thousand Oaks, CA, pp. 45-68.

Karam, G.M. and Casselman, R.S. (1993), A Cataloging Framework for Software Development Methods. *IEEE Computer*, 26 (2), 34–45.

Kennedy, M. N. (2003), Product Development for the Lean Enterprise: Why Toyota's System is Four Times More Productive and How You Can Implement It, The Oaklea Press.

Keuffel, W. (2001), Functionally Superior: Is function point analysis the leading software metric? *Software Development Magazine*, [online]: Available from: <http://www.sdmagazine.com/documents/s=1136/sdm0105l/>, [Accessed 13 April 2006].

Klein, H.K., and Myers, M.D., (1999), A set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems, *MIS Quarterly*, Vol 23, No. 1, pp.67–93. . [Online]. Available from: <http://www.qual.auckland.ac.nz/>, [last accessed 03 May 2006].

Kotonya, G. and Sommerville, I. (2002), *Requirements Engineering: Processes and Techniques*, John Wiley & Sons.

Labuschagne, L., (2002), *Cases in IT Project Management*, Published in South Africa, pp. 79–102.

Landwehr, C.E., Bull, A.R., Mcdermott, J.P., and Choi, W.S., (1994), A Taxonomy of Computer Program Security Flaws, with Examples, *ACM Computing Surveys* 26, 3.

Larman, C., and Basili, V.R., (2003), Iterative and Incremental Developments: A Brief History, *IEEE Computer*, Vol. 36, No.6, pp. 47–56.

Larman, C. (2004), *Agile and Iterative Development: A Manager's Guide*, Addison–Wesley, USA, pp. 9–39.

Lehman, M.M. and Belady, L.A. (1985), *Program Evolution, – Processes of Software Change*, Academic Press, London.

Lehman, M.M. and Ramil, J.F. (2002), Software Uncertainty. In Bustard D., Liu W. and Sterritt R., Eds., *Proceedings of Soft–Ware 2002: First Intl. Conf. on Computing in an Imperfect World*, Belfast, North Ireland, 8–10 April, LNCS 2311, Springer–Verlag, Heidelberg, 2002, pp. 174 –190.

Lindvall, M., Basili, V. R., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L. and Zelkowitz, M.V. (2002), Empirical Findings in Agile Methods, *Proceedings of Extreme Programming and Agile Methods – XP/agile Universe*, pp. 197–207.

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., May, J., and Kähkönen, T. (2004), Agile Software Development in Large Organizations, *IEEE Computer*, pp.26–33.

Mahony, M.S., (1998), Finding a history for software engineering, *Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering*, p.87.

Marchesi, M., Succi, G., Wells, D. and Williams, L. (2002), *Extreme Programming Perspectives*, Addison Wesley, pp. 17–20.

Marick, B. (2001), Agile Methods and Agile Testing, *STQE Magazine* Vol. 3, No. 5.

Martin, A., Biddle R. and Noble J. (2003) “A Tale of Three Companies: A Story About Inter–Organizational Contract Structures in Agile Development”, *18th Annual Conference on OOPSLA*.

Martin, R.C. (1999), Iterative and Incremental Development (IID), *Engineering Notebook Column, C++ Report*.

Martin, R.C. and Lindstrom, L. (2000), Interview with Robert C. Martin on Extreme Programming, *Object View Magazine*, Issue 4, [online]. Available from: <http://www.iconixsw.com/ObjectiveView/ObjectiveView4.pdf>, [last accessed 10 January 2006] pp. 30–38.

Mason, J. (2002), *Qualitative Researching*, Sage Publications.

Maurer, F. and Martel, S. (2002), Extreme Programming: Rapid Development for Web–Based Applications, *IEEE Internet Computing*, pp.86–90.

McBreen, P. (2001), *Software craftsmanship: The New Imperative*, Addison–Wesley Professional, pp.25–30.

McMunn, D. and Nielsen, J., (2005), Showcasing Agile Development: A Powerful Tool for “Crossing the Chasm”, *Digital Focus White Paper*, [online], Available from: <http://www.digitalfocus.com/research/whitepapers.php>. [Accessed 12 January 2006].

Mnkandla, E. and Dwolatzky, B. (2004a), A Survey of Agile Methodologies, *Transactions of the South Africa Institute of Electrical Engineers*, Vol.95, No.4, pp.236–247.

Mnkandla, E. and Dwolatzky, B. (2004b), Balancing the Human and the Engineering Factors in Software Development, *Proceedings of the IEEE AFRICON 2004 Conference*, pp.1207–1210.

Mnkandla, E. and Dwolatzky, B. (2004c), A Selection Framework for Agile Methodologies. *Proceedings of the Extreme Programming and Agile Processes in Software Engineering: 5th International Conference, XP 2004, Garmisch–Partenkirchen, Germany*. In Eckstein, J. and Baumeister, H. (eds.). *Proceedings of the 5th International Conference on Extreme Programming and Agile Processes in Software Engineering*. Springer–Verlag GmbH, pp.319–320.

Mnkandla, E. (2005), A Thinking Framework for the Adaptation of Iterative Incremental Development Methodologies. *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering. XP 2005, Sheffield, UK*. In Baumeister, H., Marchesi, M., and Holcombe, M. (eds.). *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering*. Springer–Verlag GmbH, pp. 315–316.

Mnkandla, E., Dwolatzky, B. and Mlotshwa, S. (2005), Tailoring Agile Methodologies to the Southern African Environment. *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering. XP 2005, Sheffield, UK*. In Baumeister, H., Marchesi, M., and Holcombe, M. (eds.). *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering*. Springer-Verlag GmbH, pp. 259–262.

Monash University, (2005) Information Technology Services MSA Internal Survey, Internal Report October 2005, pp.21.

Moonen, L., and van Deursen, A., (2003), Report on the Second Benelux Workshop on Agile Method Experiences, Research Report, Software Evolution Research Lab, Delft University of Technology, [online]. Available from: <http://name.case.unibz.it/NameResearchRoadmap-v2/NAMERoadmapVf.pdf> [Last accessed 10 May 2006].

Moore, C. (2004), Fixed Price Contracts and Agile Development, *Poppendieck.LLC*, [online]. Available from: www.poppendieck.com/pdfs/, [Last accessed 10 May 2006].

Morgagni, M. and Bossi, P. (2001), Optional scope project, *Quinary SpA*, [online]. Available from: <http://www.quinary.com>, [last accessed 10 June 2005].

Myers, M.D. (1997), Qualitative Research in Information systems, *MIS Quarterly* (21:2), pp. 241–242. *MIS Discovery*, archival version. [Online]. Available from: <http://www.qual.auckland.ac.nz/>, [last accessed 16 January 2006].

Nandhakumar, J. and Avison, J. (1999), The fiction of methodological development: a field study of information systems development, *Information Technology & People*, vol. 12, pp. 176–191.

Naur, P., Randell, B. and Buxton, J. (1996), *Software Engineering: Concepts and Techniques*, Charter Publisher, in: Fitzgerald, B., Formalized Systems Development Methodologies: a critical perspective, *Information Systems Journal*, Vol. 6, No. 1, pp.3–23.

Olerup, A. (1991), Design Approaches: A comprehensive study of information system design and architectural design, *The Computer Journal* Vol. 34, pp. 215–224.

Osterweil, L.J. and Song, X. (1996), Toward Objective Systematic Design–method Comparisons, *IEEE Proceedings of IWSSD–8*, pp. 170–171.

Palmer, S.R. and Felsing, J.M. (2002), *A Practical Guide to Feature–Driven Development*, Upper Saddle River, NJ, Prentice–Hall, pp.55–73.

Parr, E.A. (1982), *Beginner’s Guide to Microprocessors*, Butterworth & Company, pp.3–9.

Paulk, M.C. (2002), Agile Methodologies and Process Discipline, *CrossTalk: The Journal of Defense Software Engineering*, issue 10, pp. 15.

Pawlowski, S.D., and Robey, D., (2004), Bridging User Organizations: Knowledge Brokering and the Work of Information Technology Professionals, *MIS Quarterly*, Vol. 28, No. 4, pp.645–672.

Pfleeger, S. L., and Atlee, J.M., (2005), *Software Engineering: Theory and Practice*, Prentice Hall.

Pikkarainen, M., and Passoja, U., (2005) An Approach for Assessing Suitability of Agile Solutions: A Case Study. *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering. XP 2005, Sheffield, UK*. In Baumeister, H., Marchesi, M., and Holcombe, M. (eds.). *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering*. Springer-Verlag GmbH, pp.171–179.

Poppendieck, M. and Poppendieck, T. (2003), *Lean Software Development: An Agile Toolkit for Software Development Managers*, Addison Wesley, pp. xxi –xxviii.

Poppendieck, T. (tom@poppendieck.com) (13 January 2005) *Tool for Matching A Methodology to a Project*. Personal email to E. Mnkandla (Ernest.Mnkandla@infotech.monash.edu.)

Potts, C. (1993), Software Engineering Research Revisited, *IEEE Software*, pp. 19–28.

Pressman, R.S. (2000), *Software Engineering: A Practitioners Approach*, McGraw Hill.

Radding, A. (2002), Extremely Agile Programming, *Computerworld*, Vol. 36 Issue 6, pp. 42.

Rasmusson, J. (2003), Introducing XP into Greenfield Projects: Lessons Learned, *IEEE Software*, Vol.20, No. 3, pp.21–28.

Reifer, D.J., (2003), XP and the CMM, *IEEE Software*, Vol.20, No. 3, pp.14–15.

Rietmann, M. (2001), DSDM in a Bird's Eye View, DSDM Consortium White Paper, available: <http://www.dsdm.org>.

Roque, L., Almeida, A., and Figueiredo, A.D., (2004), Context Engineering: An IS Development Research Agenda, in Proc. of the European Conference on Information Systems, ECIS 2004, [online] Available from: <http://csrc.lse.ac.uk/asp/aspecis/20040140.pdf> [Accessed 09 December 2006]

Rosenberg, D. and Scott, K. (2000), Goldilocks and the three software processes, *ObjectiveView Magazine* Issue 5, pp.35–38.

Rosenberg, D. and Scott, K. (1999), *Use Case Object Modeling with UML: A Practical Approach*, Addison–Wesley, pp. 1–20.

Rosenberg, D., Collins–Cope, M., and Stephens, M. (2005), *Agile Development with ICONIX Process: People, Process, and Pragmatism*, Apress, pp. 13–25.

Royce, W.E., (1998), *Software Project Management: A Unified Framework*, Addison–Wesley, pp.299–362.

Schuh, P. (2004), *Integrating Agile Development in the Real World*, Charles River Media, Massachusetts, USA, pp. 1–6.

Schwaber, K. (2004a), *Agile Project Management with Scrum*, Microsoft Press.

Schwaber, K. (2004b), Scrum Mater Tutorial, Proceedings of the Extreme Programming and Agile Processes in Software Engineering: 5th International Conference, XP 2004, Garmisch–Partenkirchen, Germany. In Eckstein, J. and

Baumeister, H. (eds.). *Proceedings of the Extreme Programming and Agile Processes in Software Engineering*. Springer–Verlag GmbH.

Schwaber, K. and Beedle, M. (2002), *Agile Software Development with SCRUM*, Prentice–Hall, pp.23–30.

Schwalbe, K. (2004), *Information Technology Project Management*, Thomson Course Technology.

Shemer, I. (1987), Systems Analysis: a systematic analysis of a conceptual model, *Communications of the ACM*, Vol. 30, No. 6, pp. 506–512.

Siau, K. and Rossi, M. (1998), Evaluation of Information Modeling Methods: A Review, *IEEE Proceedings of the 31st Annual Hawaii International Conference on Systems Sciences*, pp. 314 –322.

Sol, H.G. (1983), A Feature Analysis of Information Systems Design Methodologies: Methodological considerations, *Proceedings of the IFIP WG8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies*. In Olle, T.W., Sol, H.G., and Tully, C.J., (eds.). *Information Systems Design Methodologies: A Feature Analysis*. Amsterdam, Elsevier, pp. 1–7.

Sommerville, I. (2004), *Software Engineering*. Addison–Wesley.

Song, X. (1995), A Framework for Understanding the Interpretation of Design Methodologies, *ACM SIGSOF Software Engineering Notes*, Vol. 20, No. 1, pp. 46–54.

Song, X. and Osterweil, L.J. (1992a), Toward objective, systematic design–method comparisons, *IEEE Software*, Vol. 9, No. 3, pp. 43–53.

Song, X. and Osterweil, L.J. (1992b), A Process–Modeling Based Approach to Comparing and Integrating Software Design Methodologies, *IEEE Proceedings*, pp. 225–229.

Song, X. and Osterweil, L.J. (1992c), Comparing Design Methodologies Through Process Modeling, *IEEE Software*, pp. 29–44.

Song, X. and Osterweil, L.J. (1994), Experience with an Approach to Comparing Software Design Methodologies, *IEEE Transactions on Software Engineering*, Vol. 20, No. 5, pp. 364–384.

Standish Group International. (1994), *The CHAOS Report*, [online], Available from: http://www.pm2go.com/sample_research/chaos_1994_1.php [Accessed 2nd May 2005].

Stapleton, J. (2003), *DSDM Consortium, DSDM Business Focused Development*, Addison–Wesley, pp. xix–21.

TechWeb, TechEncyclopedia: Online Technical Encyclopedia, [online] Available from: <http://www.techweb.com/encyclopedia/> [Accessed 09 May 2006]

Turk, D., France, R. and Rumpe, B. (2002), Limitations of Agile Software Processes, *Agile Alliance*. [online] Available from: www.agilealliance.org/articles/articles/LimitationsofAgile.pdf, [Accessed 09 May 2006]

van Vliet, H. (2003), *Software Engineering: Principles and Practice*, John Wiley and Sons.

Walsham, G. (1993), *Interpreting Information Systems in Organizations*, Wiley, Chichester.

Williams, L., Kessler, R.R., Cunningham, W., and Jeffries, R. (2000), Strengthening the Case for Pair-Programming, *IEEE Software*. p. 19–25.

Yeh, R. (1991), System Development as a Wicked Problem, *International Journal of Software Engineering and Knowledge Engineering*, Vol. 1, No. 2, pp.117–130.

9 Appendices

9.1 Appendix A Agile Manifesto for Agile Software Development

From www.Agilealliance.org

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Mike	Beedle	Grenning	Martin
Arie van Bennekum		Jim Highsmith	Steve Mellor
Alistair Cockburn		Andrew Hunt	Ken Schwaber
Ward Cunningham		Ron Jeffries	Jeff Sutherland
Martin Fowler		Jon Kern	Dave Thomas
		Brian Marick	

9.1.1 Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals.

Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development.

The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and fine design enhances agility.

Simplicity –the art of maximizing the amount of work not done –is essential.

The best architectures, requirements, and designs emerge from self –organizing teams.

At regular intervals, the team reflects on how to become more effective, then tailors and adjusts its behavior accordingly.

Appendix B Examples of Plan-driven and Agile Approaches
 From Boehm and Turner (2004) pp. 14–15, and 21–22.

Examples of Agile Methods		
Method	Players	Description
eXtreme Programming (XP)	Kent Beck, Ward Cunningham, Ron Jeffries, Daimler Chrysler	Probably the most famous agile method. Refined from experience gained developing an information system for Daimler Chrysler Corporation. XP is fairly rigorous and initially expects all of the practices defined in the method to be followed. These practices include stories, pair programming, simple design, test first, and continuous integration.
Adaptive Software Development (ASD)	Jim Highsmith	Responding to industry turbulence and the need for rapid business development, ASD provides a philosophical base and practical approach using iterative development, feature-based planning, and customer focus group reviews within a leadership-collaboration management style.
Crystal	Alistair Cockburn	A family of methods that provide different levels of "ceremony" depending on the size of the team and the criticality of the project. Practices draw from agile and plan-driven methods as well as psychology and organizational development research.
Scrum	Ken Schwaber, Jeff Sutherland, Mike Beedle	More a management technique, Scrum projects are divided into 30-day work intervals in which a specific number of requirements from a prioritized list (backlog) are implemented. Daily 15-minute "Scrum meetings" maintain coordination.
Feature-Driven Development (FDD)	Jeff DeLuca, Peter Coad	A very lightweight architecturally based process that initially establishes an overall object architecture and features list. It then proceeds to design-by-feature and build-by-feature. Maintains Chief Architect and Chief Programmer roles. The use of UML or other object-oriented design methods is strongly implied.

Examples of Plan-Driven Approaches

Method	Players	Description
Military standards	DoD	DoD-STD-2167 was a document-driven approach that specified a large number of Data Item Descriptions for deliverables. Tailoring was encouraged, but was rarely effectively done. MIL-STD-1521 details a set of sequential reviews and audits required. MIL-STD-498 revised 2167 to allow more flexibility in systems engineering, planning, development, and integration. MIL-STD-499B defines the contents of a systems engineering management plan.
General process standards	ISO, EIA, IEEE	EIA/IEEE J-STD-016 was a generalization of MIL-STD-498 to include commercial software processes. ISO 9000 is a quality management standard that includes software. ISO 12207 and 15504 address the software life cycle and ways to appraise software processes.
Software factories	Hitachi, General Electric, others	A long-term, integrated effort to improve software quality, software reuse, and software development productivity. Highly process-driven, emphasizing early defect reduction.
Cleanroom	Harlan Mills, IBM	Uses statistical process control and mathematically based verification to develop software with certified reliability. The name "Cleanroom" comes from physical clean rooms that prevent defects in precision electronics.
Capability Maturity Model (SW-CMM)	SEI, Air Force, Watts Humphrey, Mark Paulk	A process improvement framework, SW-CMM grew out of the need for the Air Force to select qualified software system developers. Collects best practices into Key Practice Areas that are organized into five levels of increasing process maturity.
CMM Integration (CMMI)	SEI, DoD, NDIA, Roger Bate, Jack Ferguson, Mike Phillips	CMMI was established by DoD and the National Defense Industrial Association (NDIA) to integrate software and systems engineering CMMs, and improve or extend the CMM concept to other disciplines. CMMI is a suite of models and appraisal methods that address a variety of disciplines using a common architecture, vocabulary, and core of process areas.
Personal Software Process (PSP) / Team Software Process (TSP)	Watts Humphrey, SEI	PSP is a structured framework of forms, guidelines, and procedures for developing software. It is directed toward the use of self-measurement to improve individual programming skills. TSP builds on PSP and supports the development of industrial-strength software through the use of team planning and control.

9.2 Appendix C The Questionnaire and Interviews

Structure of the questionnaire

The questionnaire used to elicit the relevant information was structured as shown below:

9.2.1 Software Development Questionnaire

SOFTWARE DEVELOPMENT PROCESS QUESTIONNAIRE

The purpose of this questionnaire is to elicit information about the approaches and principles that are applied by software development organizations to do the following:

Decide on the process that can be followed for a particular project.

Select the software development methodology practices appropriate for the project at hand.

Identify the issues of organizational culture that affect the ultimate plan of a project.

This questionnaire is meant to take not more 20 minutes of your time.

The results of this document are intended purely for research purposes and will lead to the improvement of the design of a thinking tool for the selection of software development practices from a group of methodologies.

Please try to be as informal as possible and say what you do in projects as opposed to what you are expected to do.

You may send it back by email or by fax the details are given on the last page.

Definition of Terms

The following terms will be defined according to the way they have been used in the research.

Methodology

A methodology is a study of a group related methods. For the purposes of this research a methodology will be defined as a codified set of recommended practices, sometimes accompanied by training materials, formal educational programs, worksheets, and diagramming tools.

Organizational Culture

Organizational culture is the way people in an organization carryout their daily endeavors, how they relate to each other, and what motivates them to do what they do. This may not necessarily be what is written down in corporate documents as that organization's culture, rather what has emerged to be a way business is done in the organization. For the purposes of this questionnaire the aspects of the culture as they relate to software projects would be more relevant.

Project

A project is a temporary endeavor that is undertaken to create a unique product or service. A software development project in this case can be defined as a temporary effort to create a unique software product. In this context the term product encompasses any software applications including computer-based systems, embedded software, and problem solving programs that are to be built. A project is characterized by five attributes:

Unique purpose. Means that every project should have a well defined objective.

Temporary. A project has a clear-cut beginning and an explicit end.

Requires resources. A project requires resources often from various areas. The resources required include people, hardware, software or other assets.

Primary customer or sponsor. Although most projects might have many interested parties or stakeholders, someone must take the primary role of sponsorship.

Involves uncertainty. Every project is unique which brings uncertainty in defining the project's objectives, estimate how long it will take to complete, or how much it will cost

Process

The notion of a process in developing software refers to a description of activities that must be carried out to realize the intended product. The software development process can be defined as a lifecycle.

Project Parameters

In this case ‘project parameters’ refers to the elements that define a software project such as project size, risk, scope, cost, timeframe, criticality, and complexity etc.

Iterative Incremental Development

Iterative development is an approach to building software in which the overall lifecycle is composed of several iterations in sequence. The principle behind iterative development is to develop each iteration as a full realization of the system based on an initial set of requirements and follow a software development process until an increment is released. The next set of requirements can then be added and another iteration begins. The iterations can be developed concurrently.

Management of Projects

What factors do you use to decide on whether a project can be done the Waterfall Iterative way?

Is there any other process that you follow that may not necessarily be classified as waterfall based or iterative? Briefly describe the process.

How do you use (if at all) the following project parameters when planning your project?

Size_____

Requirements

Stability_____

Risk_____

Customer

Availability_____

Complexity_____

Criticality_____

Methodology Issues

How do you decide on the set of practices to follow in the development?

How do you deal with issues of product quality?

How do you plan for fears and risks in each project?

Project Success Rate

How many projects has your organization delivered over the past three years?

How many projects failed before start over the past three years?

How many projects were cancelled before close over the past three years?

What would you say were the main reasons for project failure?

The Organization

How many employees does your company have?

How long has it been in existence? years.

Does your organization have other branches? If yes where _____

How many senior management staff does your organization have?

How many middle management staff does your organization have?

How many developers does your organization have?

Any other comments can be included on a separate paper.

My Contacts: Ernest.Mnkandla@infotech.monash.edu,

Tel: (011) 950 4038,

Cell: 073 219 6927

Fax: (011) 950 4033.

APPENDIX C

Please enter your contact details below. No names of companies or participants will be revealed and you can be given the final analysis of the survey if you need it.

Your Full name	
Position in Company	
Company's Name	
Number of projects you have done (including current ones)	
Your email address	
Telephone Number	

This survey is being conducted purely for research purposes and the researcher is working towards a PhD in Engineering with the School of Electrical and Information Engineering, University of the Witwatersrand, Johannesburg.

Promoter: Professor Barry Dwolatzky

Whose **contacts are:** b.dwolatzky@ee.wits.ac.za, Tel: +27 11 717 7206.

9.3 Appendix D Personal Correspondence

This section lists the interview data.

9.3.1 Interview Data

Table 9-1: List IT Professionals Interviewed

Job Title	Organisation	Focus Area
Chief Executive Officer	Psybergate	Software Development and IT Consultancy
Managing Director and Principal Consultant	Insyte Information Engineering Systems	Consultancy in Business Process Innovation and IT Project Management
Senior Programmer	TSS Education	Application Development and Database Engineering
Business Analyst	TSS Education	Application Development and Database Engineering
Principal Consultant and Trainer	Objects in Action	Coaching, Consulting and training in IT Projects.
Technical Architect	Telenor Mobil	Mobile and fixed Network Services Provider, and ICT Research and Development
Senior Consultant	Poppendieck LLC	Application of Lean Thinking in Software Development

Table 9.2 lists the themes resulting from data analysis classified into three categories. The first category, project taxonomy, includes seven themes relating to the classification of a project. The second category, methodology representation, includes three themes that define the fundamentals of a methodology. The third category, Methodology tailoring, includes two themes that define the key points in tuning a methodology to a specific environment.

Table 9-2: Themes Identified in the Data

Project Taxonomy • Requirements stability • Project size • Development timeframe • Project complexity • Project criticality • Project scope • Project risk
Methodology Representation • People or social issues • Technical issues • Change tolerance issues
Methodology Tailoring • Organization culture • Individual style

9.4 Appendix E Framework Validation Document

Agile Methodology Selection Framework

The philosophy of the framework is:

All activities should be driven by what the customer values and prioritizes in a given project. All practices should be classified into social practices, and technical practices.

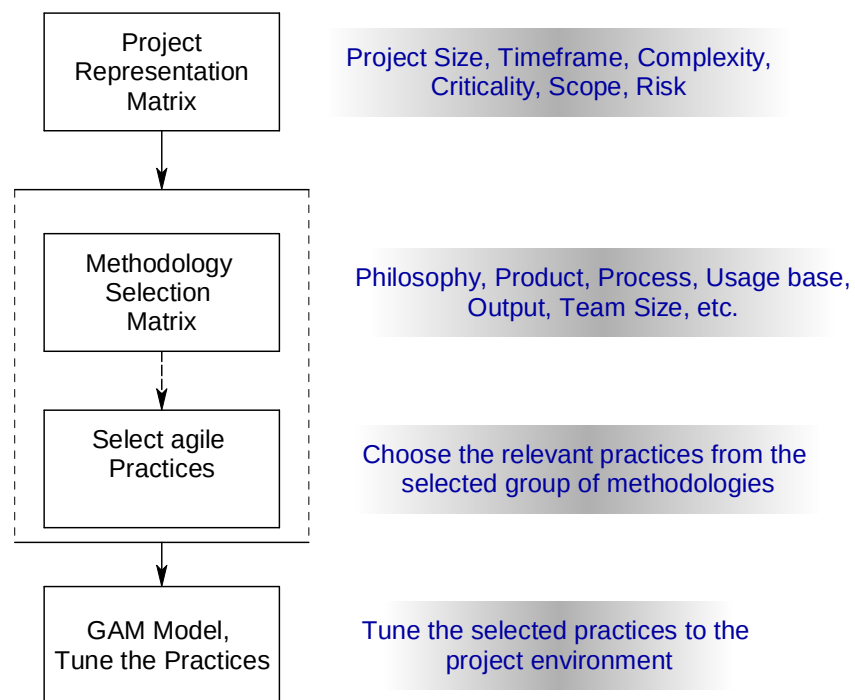


Figure E4-1 Agile Methodologies Selection Framework

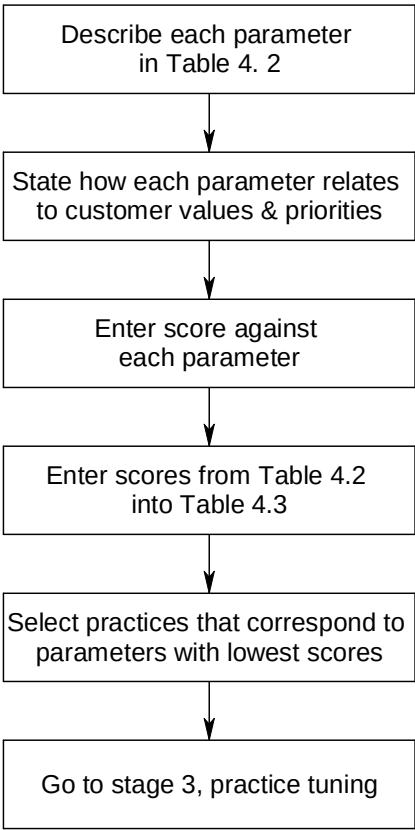


Figure E4-2: Summary of Stage Two

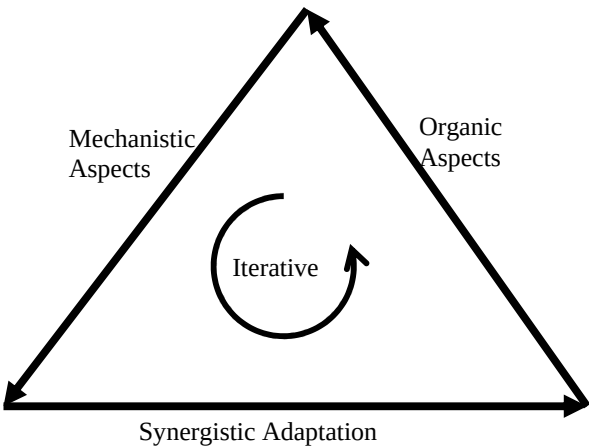


Figure E4-3 Generic Agile Methodologies (GAM), Model stage 3

TableE1.1: Top-level Summary of the Framework

Stage	Brief Description	Purpose of stage	Matrixes and Tables
One	Software Project taxonomy.	Determine the suitability of agile development in a project.	Table 4.1 Figure 4.1
Two	Methodology and practice selection.	Select the relevant agile methodology practices for a given project.	Figure 4.2, Table 4.2, 4.3, and 4.4
Three	Practice tuning	Adapt the practices to the given environment.	Table 4.5 Figure 4.3

Typical Questions

The following questions will guide the validation process:

Using past or present software development project data list the most important requirements and priorities and values. This focuses on getting an understanding of customer values and priorities for the project.

Quality

Cost efficiency

Go to stage one of the framework (framework process diagram and matrix tables presented) and use your information to determine the suitability of agile development in the project. The aim is to determine the relevance of using agile development in this project.

See Table (4.1)

Did the use of the project taxonomy matrix improve your understanding of the project with regard to process issues? The aim is to evaluate the contribution of the framework to the understanding and use of agile methodologies.

I think the factors you present contribute to making a decision. There may be an additional factor in the way of commercial constraints. In my experience consumers of software development services (whether the IT organization providing the services is internal or external to the enterprise) are adamant of the need for an upfront fixed cost and fixed time commitment.

Go to stage two of the framework and assume that the previous stage indicated applicability of agile development. Apply the methodology selection matrix and

select the set of practices that are relevant to your project. Did the use of the methodology selection matrix improve your understanding of the set of activities expected in your project? The aim of this question is to evaluate the framework's contribution to knowledge.

See tables 4.2, 4.3 and 4.4

It was helpful in making explicit the relevance of an approach in a particular context.

Go to stage three of the framework and classify the practices into social and technical. Consider what needs to be changed in each practice in order for it to be practically usable in your project environment.

Planning game: Social (is practically usable in vanilla form – use use-cases rather than stories)
 Whole team: Social (is practically usable in vanilla form – sometimes have to compromise on customer on site – poses considerable risk)
 Informative workspace: Social (is practically usable in vanilla form – sometimes compromised by distributed work force)
 Weekly cycle: Social (is practically usable – found 2 weekly cycles better)
 Ten minute build: Technical (practically usable through use of build tools)
 Continuous integration: Technical (practically usable)
 Test first programming: Technical (practically usable)
 Incremental design: Technical (practically usable – some broad based design needs to be done upfront to establish broad scope – determined largely by client need for upfront commitment re cost and time)
 Real customer involvement: Social (practically usable)
 Shared code: Social (practically usable)
 Daily deployment: Social (practically usable)

Given the opportunity would you use this framework in planning your next software development project?

The framework was useful in highlighting the explicit choices one needs to make to match practice to project context. In terms of its use – we are a niche software developer and do not get confronted with extensive diversity in projects as defined by the framework.

Would you recommend the use of this framework to other IT professionals?

I would.

Comment on any notable limitations of the framework.

-
4. Presentation of the framework and ease of use could be improved
 5. Consider more fine grain determinants of project description
 6. Make the practices themselves the target of choice (rather than methodology)
-

Table E4-1: Stage One: Determine Between the Two Major Paradigms

Project Parameters	Rating 0 to 5	Methodology Choice
Requirements Stability	3	Agile
Project Size	0	Agile
Development Timeframe	3	Agile / Unified process
Project Complexity	2	Agile
Project Risk	4	Agile

Key:

Requirements stability: 0 means very unstable requirements with a high risk of scope creep. The level of stability increases to a rating of 5 representing very stable requirements where scope creep is minimal. The smaller the rating the more relevant agility is to the project.

Project size: 0 represents a project team¹⁶ with two to ten people and 5 represents more than hundred people. The relevance of agility decreases as the rating increases.

Development timeframe: 0 represents a time from one week to two months. Any development timeframe beyond two years would have a rating of 5. The shorter the development timeframe the more relevant agility is to the project.

Complexity: 0 represents those projects not characterized by heavy calculations under high change, high speed, and uncertainty. The complexity increases as the rating increases to 5, which would represent critical projects such as spacecraft or nuclear control software. Agile methodologies would be less applicable as the complexity increases (Boehm and Turner, 2004; Highsmith, 2002a)

Project risk: 0 represents high-risk projects and as the risk decreases the rating increases to 5, which would represent very low risk projects.

¹⁶ Members of the project team include developers and the customer specially invited to be part of the team.

Table E4-2: Stage Two: Methodology Description

Parameter	Parameter Description	Relation to Customer Values and Priorities	Score
Methodology Philosophy	Communication, simplicity, feedback, courage	Supports quality and cost efficiency	4
Methodology Process	See practices specifically	Supports quality and cost efficiency	5
Methodology Techniques and Tools	See practices	Supports quality and cost efficiency	5
Methodology Scope	Limited to construction	Supports quality and cost efficiency	5
Methodology Outputs	User tested code, comprehensive progress metrics	Supports quality and cost efficiency	4
Adoption and Experience	Wide and relatively easy	Supports quality and cost efficiency	4
Methodology Product	?		

Roles and Responsibilities	Testers - develop test cases - test - report on bugs Interaction designer / Analyst - develop stories - keep model up to date Architects - look for and execute large scale refactorings - system level tests that stress architecture - Partitioning Project manager - communication inside team - communication outside team - keeps metrics of progress Technical writer - create closer relationship with users - write manual Users - pick stories - domain decisions Programmers - break stories into tasks - estimate tasks - write tests - write code - implement features - automate tedious development processes - improve design of system - development infrastructure	Supports quality and cost efficiency	5
Support for Distributed Teams		Supports quality and cost efficiency	3

Table E4-3: Stage Two: Methodology Selection Matrix

Parameters	ASD	Crystal	FDD	AMDD	ICONIX	Scrum	DSDM	LD	XP	0 Score
Methodology Philosophy									4	
Methodology Process									5	
Methodology Techniques and Tools									5	
Methodology Scope									5	
Methodology Outputs									4	
Adoption and Experience									4	
Methodology Product										
Roles and Responsibilities									5	
Support for Distributed Teams									3	
Score per methodology										Score per parameter

Table E4-4: Methodology Practices

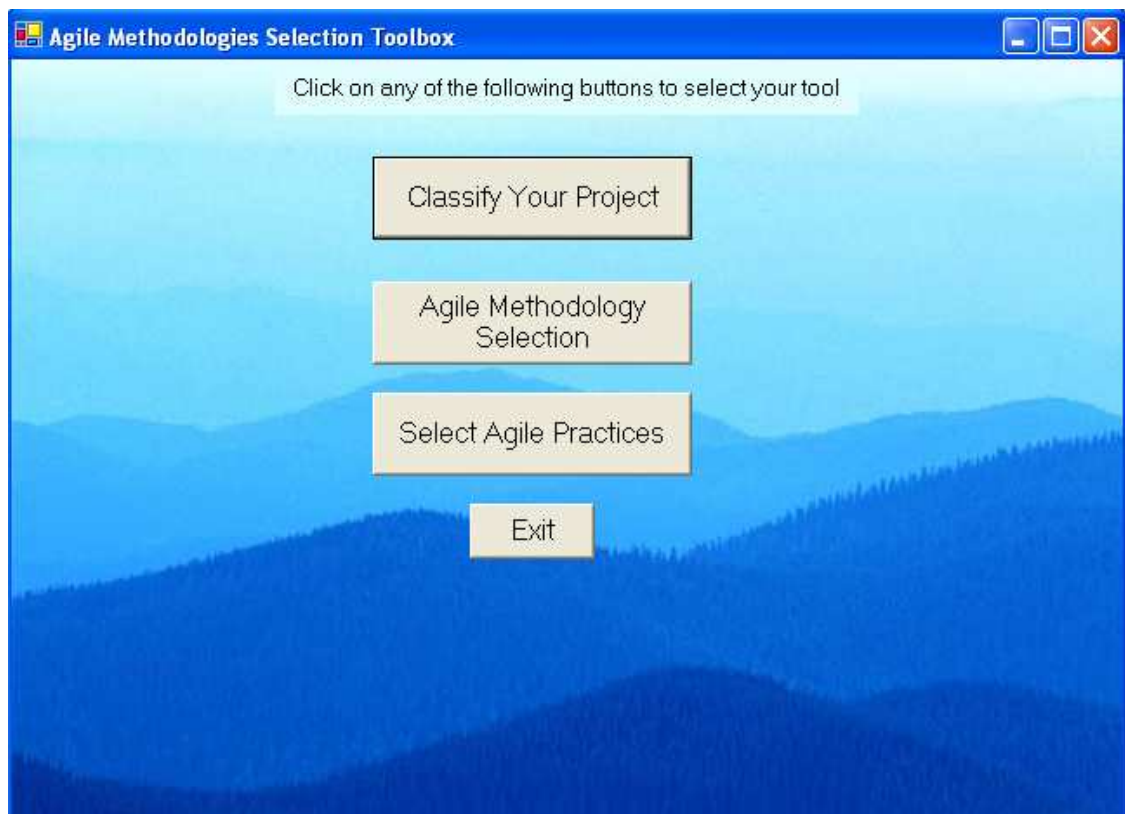
Methodology	List of Practices
XP	Planning game Whole team Informative workspace Weekly cycle Ten minute build Continuous integration Test first programming Incremental design Real customer involvement Shared code Daily deployment
Scrum	
LD	
DSDM	
FDD	
AMDD	
ICONIX	
Crystal	
ASD	

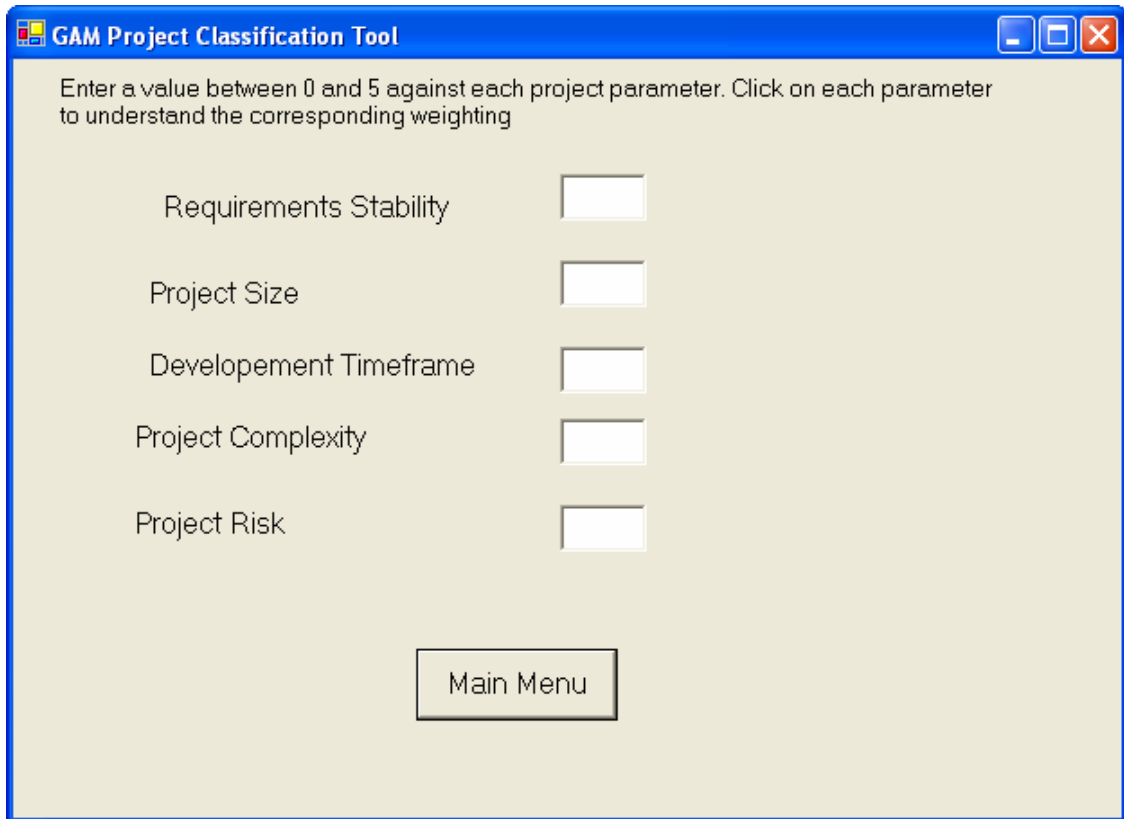
Table E4-5: Values of the Agile Manifesto

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

9.5 Appendix F Screen Dumps for the Generic Agile Methodologies Toolbox

The screen dumps are shown starting with the main menu and ending with a series of screens for each agile methodology's practices.





The screenshot shows a software window titled "GAM Project Classification Tool". Inside the window, there is a text instruction: "Enter a value between 0 and 5 against each project parameter. Click on each parameter to understand the corresponding weighting". Below this instruction, there are five project parameters, each followed by a text input field:

- Requirements Stability
- Project Size
- Development Timeframe
- Project Complexity
- Project Risk

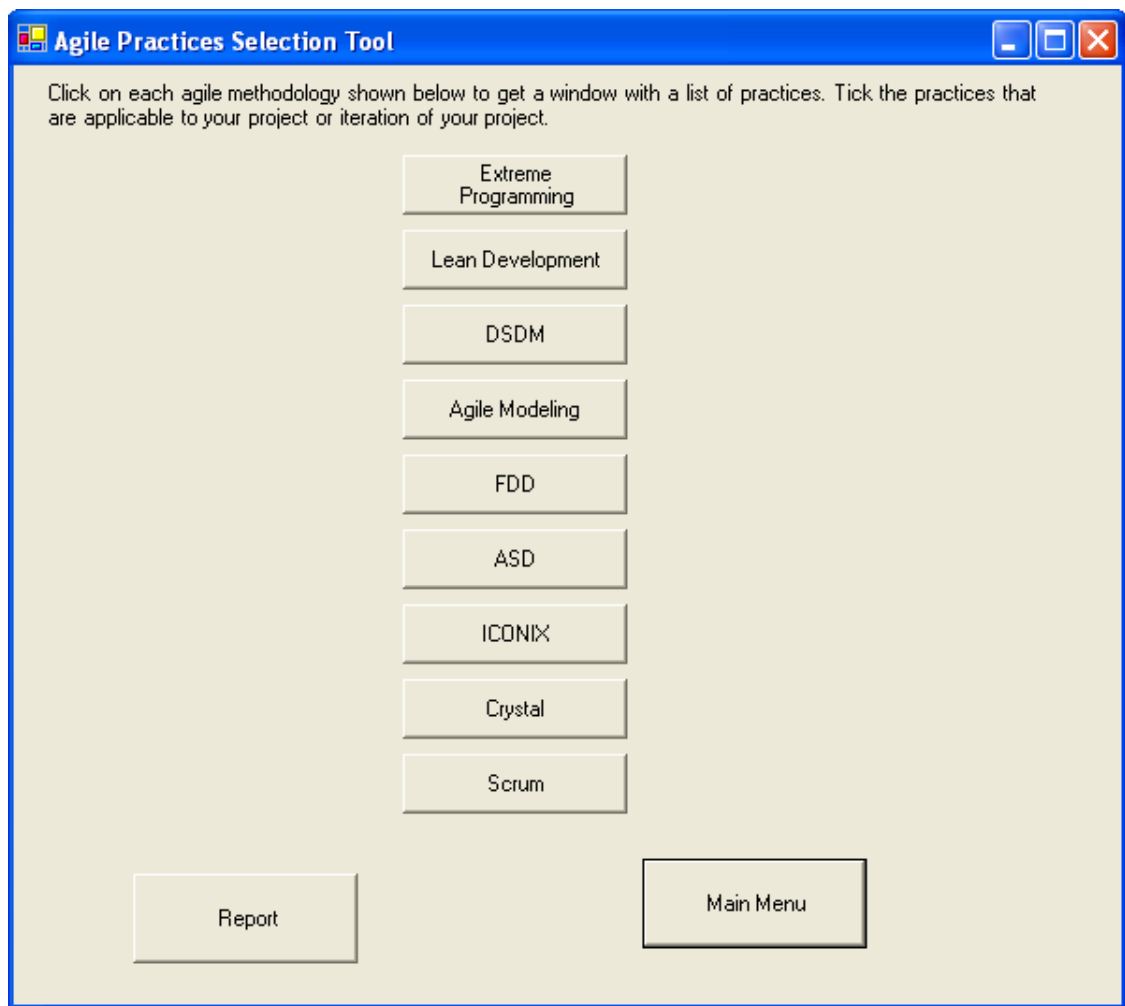
At the bottom center of the window, there is a button labeled "Main Menu".

Methodology Selection Tool

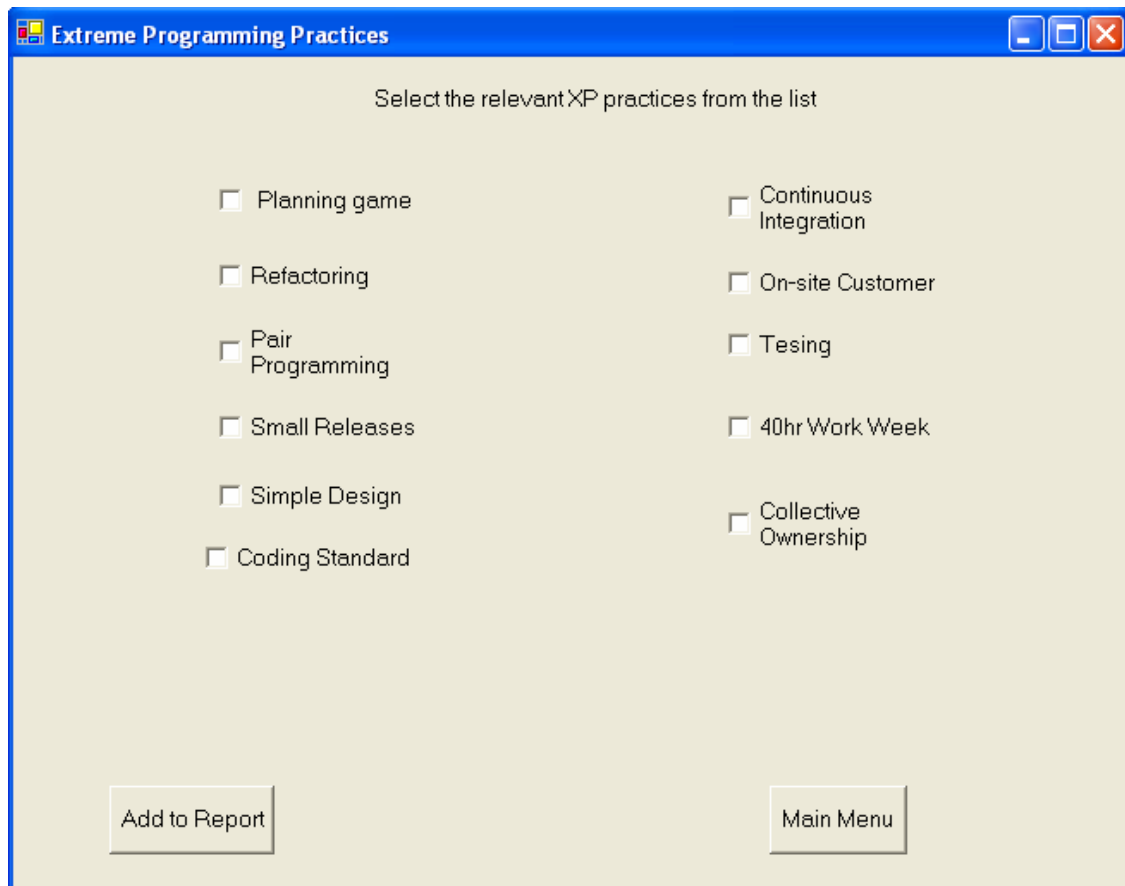
Click on each of the following methodology parameters to access a list of project related attributes per methodology. The window that pops up will allow you to enter a value between 0 and 5 against each methodology. A weighting of 0 means that the attributes are least or not relevant to your project's needs and 5 means that the attributes are most relevant.

	ASD	Crystal	FDD	AMDD	ICONIX	Scrum	DSDM	LD	XP	5 Score
Philosophy										
Process										
Techniques and Tools										
Outputs										
Adoption and Experience										
Roles and Responsibilities										
Distributed Team Support										
Final Choice										

Final Choice Main Menu



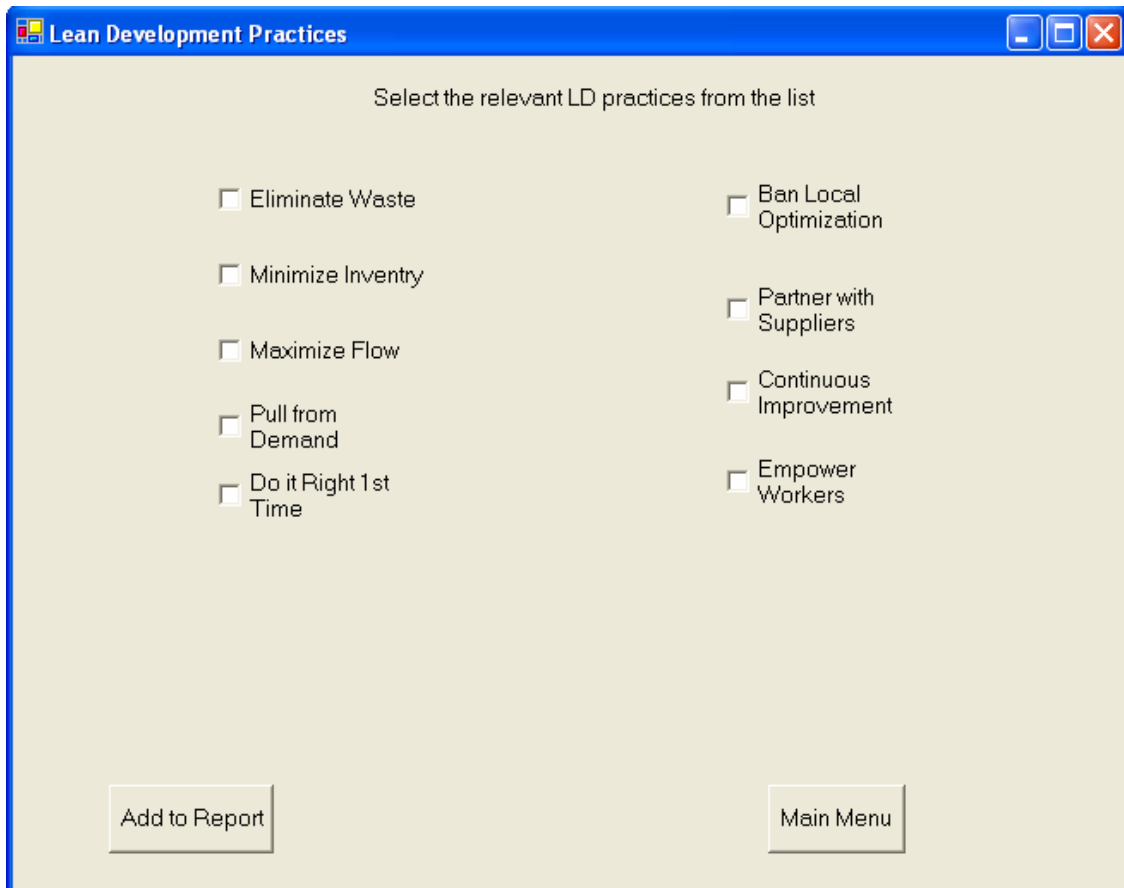
A selection of each agile methodology will open a window showing a list of the most commonly used practices of the selected methodology from which the user may select according to relevance.



Extreme Programming Practices

Select the relevant XP practices from the list

☐ Planning game	☐ Continuous Integration
☐ Refactoring	☐ On-site Customer
☐ Pair Programming	☐ Testing
☐ Small Releases	☐ 40hr Work Week
☐ Simple Design	☐ Collective Ownership
☐ Coding Standard	

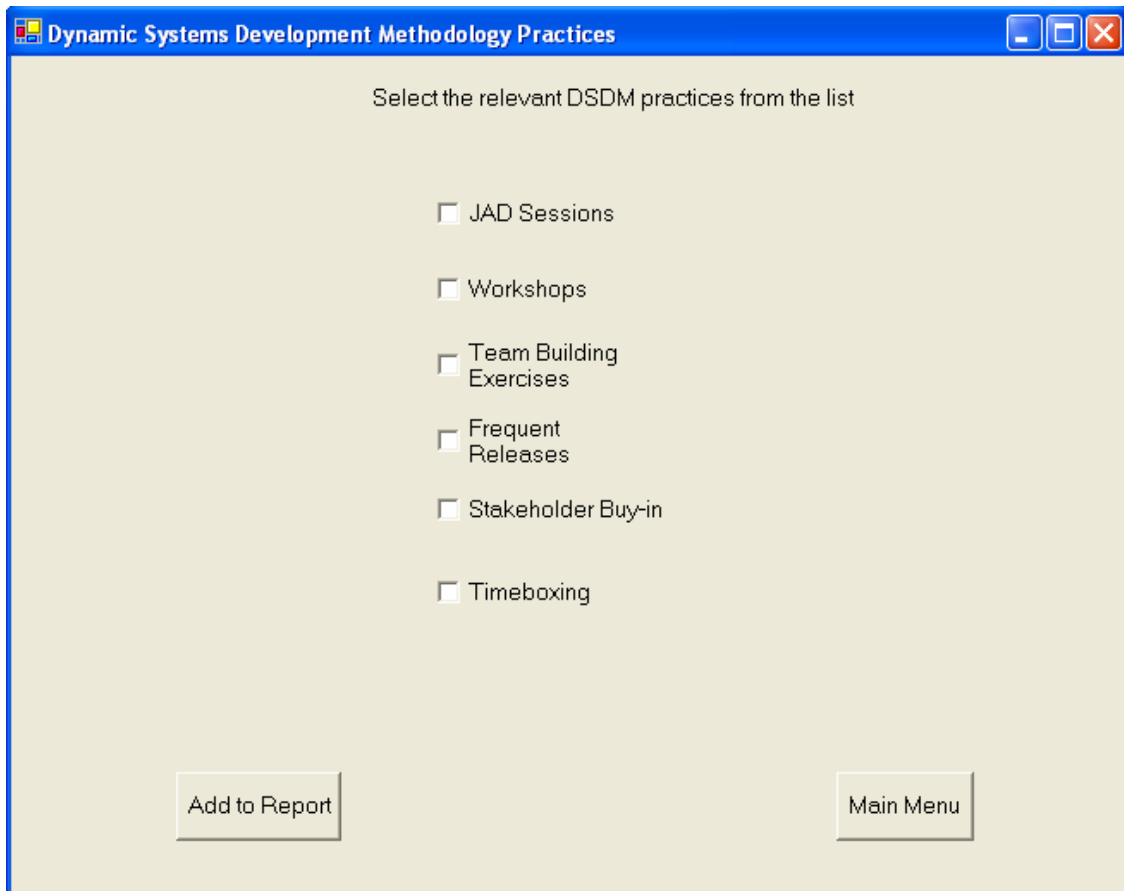


Lean Development Practices

Select the relevant LD practices from the list

☐ Eliminate Waste	☐ Ban Local Optimization
☐ Minimize Inventory	☐ Partner with Suppliers
☐ Maximize Flow	☐ Continuous Improvement
☐ Pull from Demand	☐ Empower Workers
☐ Do it Right 1st Time	

Add to Report Main Menu

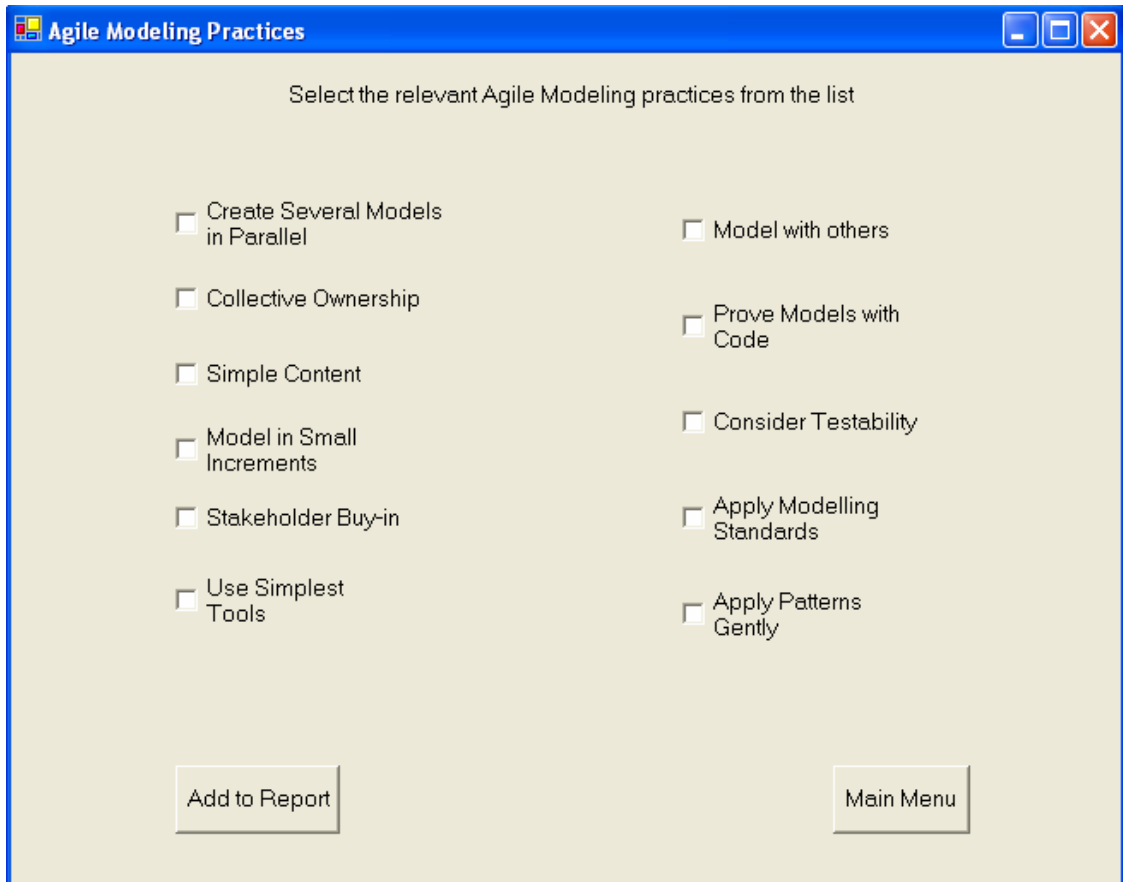


Dynamic Systems Development Methodology Practices

Select the relevant DSDM practices from the list

- ☐ JAD Sessions
- ☐ Workshops
- ☐ Team Building Exercises
- ☐ Frequent Releases
- ☐ Stakeholder Buy-in
- ☐ Timeboxing

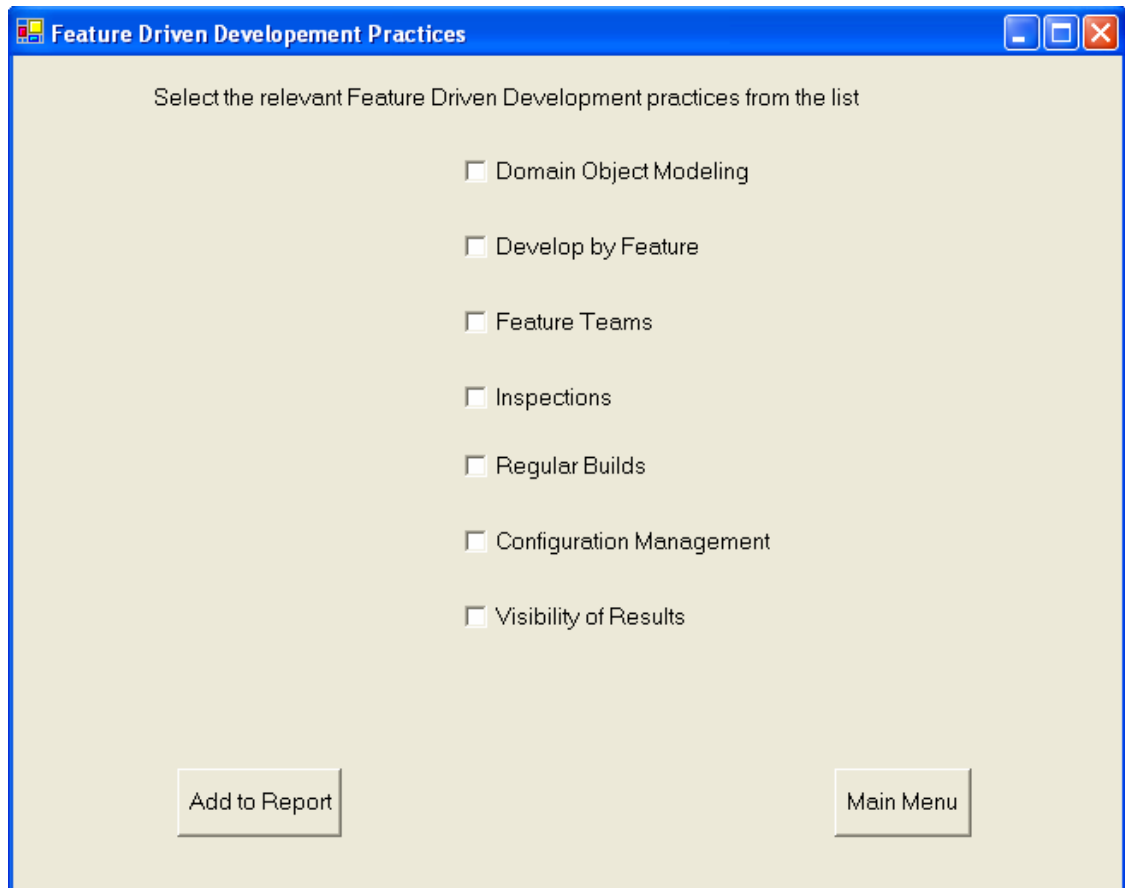
Add to Report Main Menu



Agile Modeling Practices

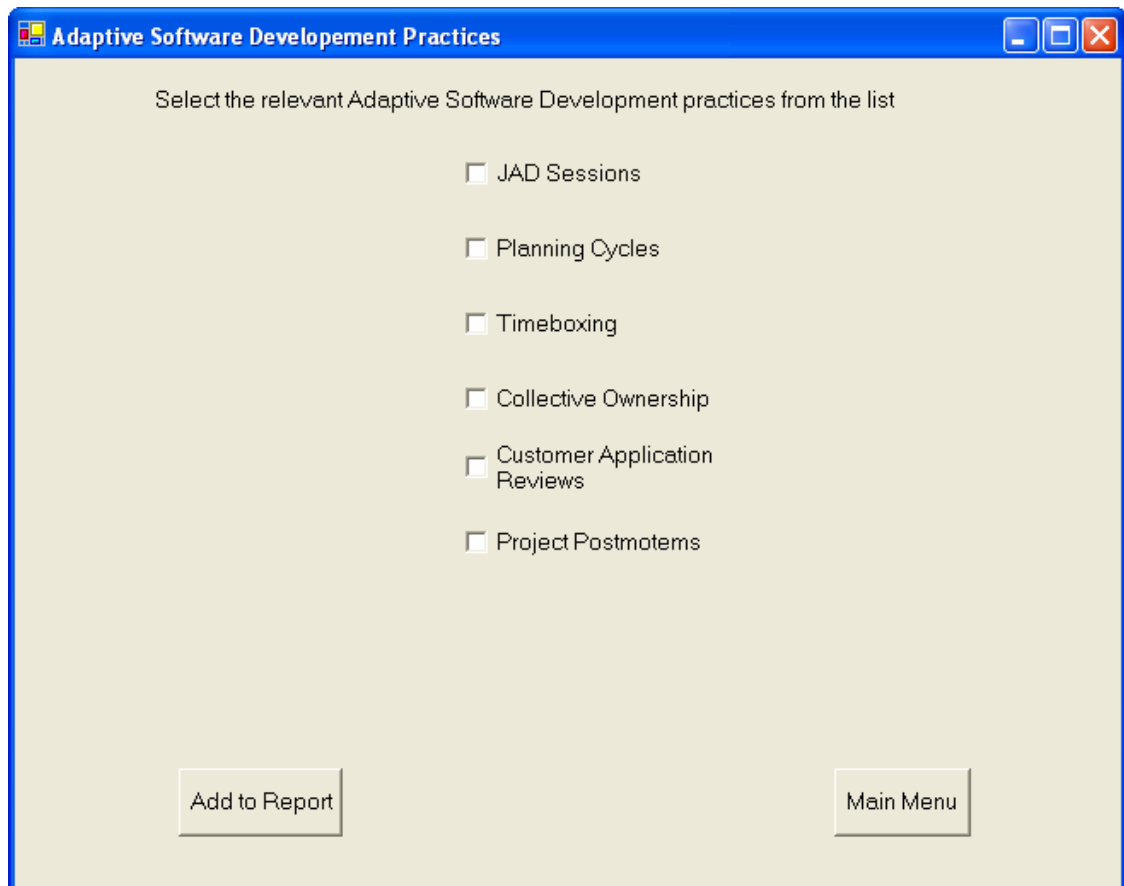
Select the relevant Agile Modeling practices from the list

☐ Create Several Models in Parallel	☐ Model with others
☐ Collective Ownership	☐ Prove Models with Code
☐ Simple Content	☐ Consider Testability
☐ Model in Small Increments	☐ Apply Modelling Standards
☐ Stakeholder Buy-in	☐ Apply Patterns Gently
☐ Use Simplest Tools	



The screenshot shows a window titled "Feature Driven Development Practices" with a blue title bar and standard Windows window controls (minimize, maximize, close). The main area has a light beige background and contains the following elements:

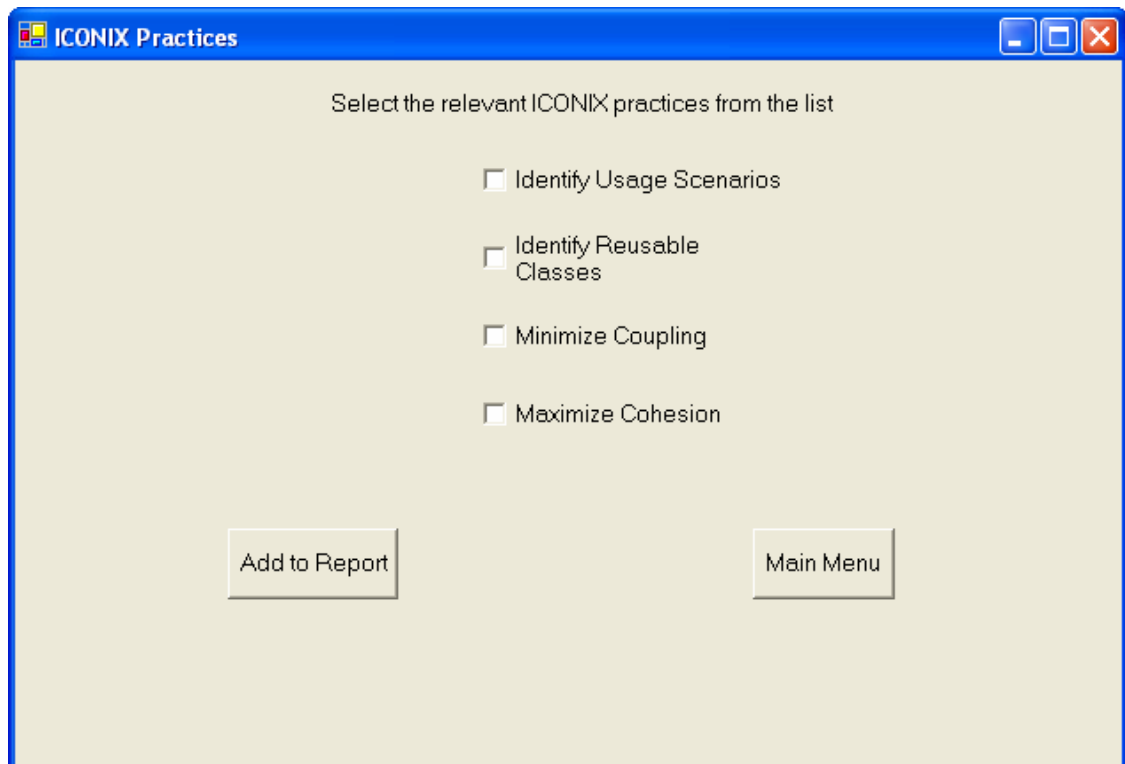
- Instructional text: "Select the relevant Feature Driven Development practices from the list"
- A list of seven practices, each preceded by an unchecked checkbox:
 - ☐ Domain Object Modeling
 - ☐ Develop by Feature
 - ☐ Feature Teams
 - ☐ Inspections
 - ☐ Regular Builds
 - ☐ Configuration Management
 - ☐ Visibility of Results
- Two buttons at the bottom:
 - "Add to Report" on the left
 - "Main Menu" on the right

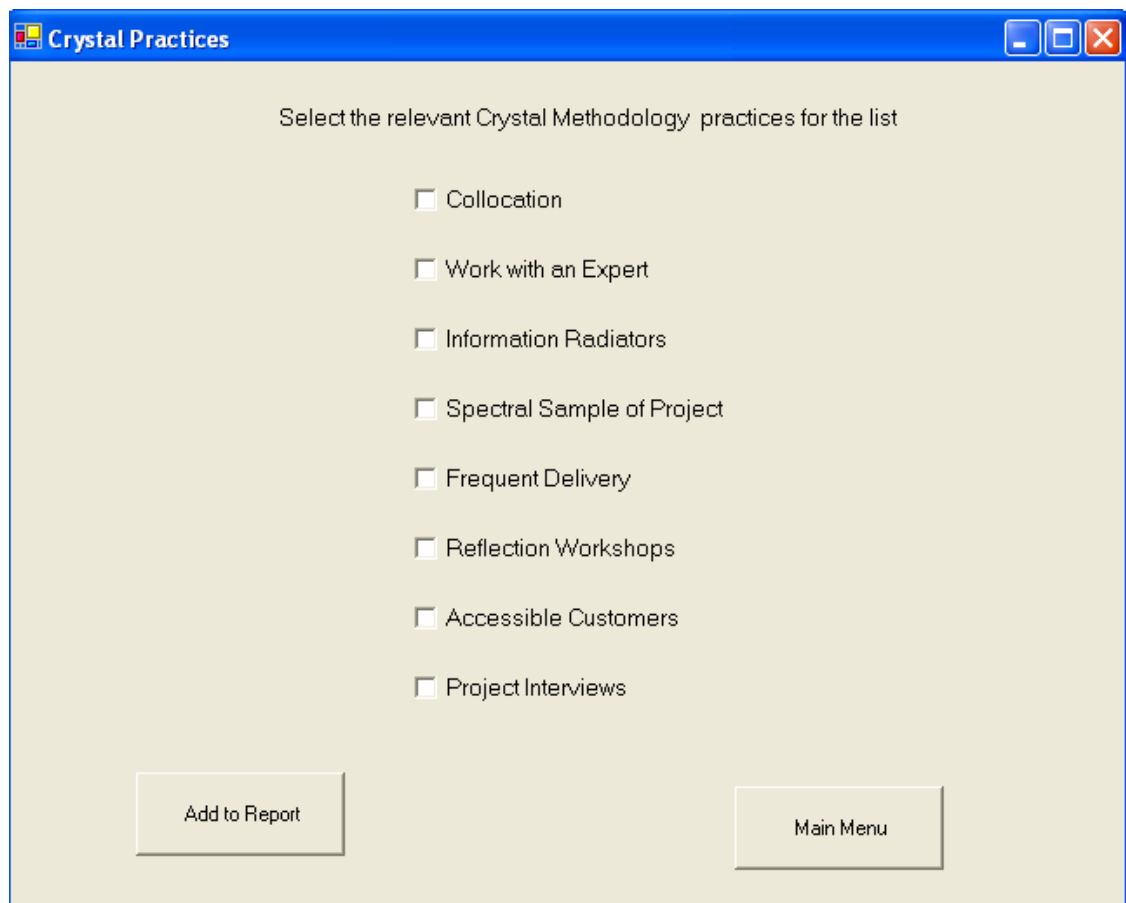


The screenshot shows a window titled "Adaptive Software Development Practices" with a blue title bar and standard Windows window controls (minimize, maximize, close). The main area has a light beige background and contains the instruction "Select the relevant Adaptive Software Development practices from the list". Below this instruction is a list of six practices, each preceded by an unchecked checkbox:

- ☐ JAD Sessions
- ☐ Planning Cycles
- ☐ Timeboxing
- ☐ Collective Ownership
- ☐ Customer Application Reviews
- ☐ Project Postmortems

At the bottom of the window, there are two buttons: "Add to Report" on the left and "Main Menu" on the right.





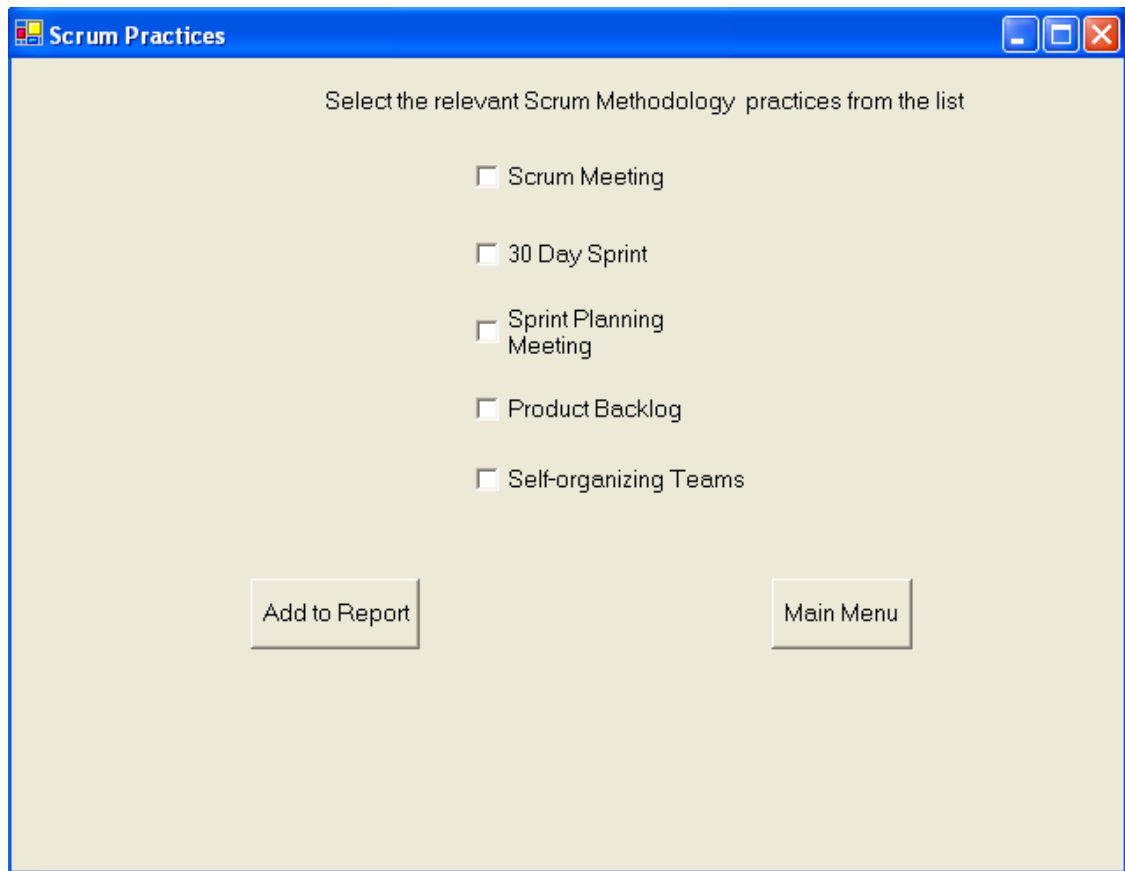
Crystal Practices

Select the relevant Crystal Methodology practices for the list

- ☐ Collocation
- ☐ Work with an Expert
- ☐ Information Radiators
- ☐ Spectral Sample of Project
- ☐ Frequent Delivery
- ☐ Reflection Workshops
- ☐ Accessible Customers
- ☐ Project Interviews

Add to Report

Main Menu



A screenshot of a software window titled "Scrum Practices". The window has a blue title bar with standard Windows window controls (minimize, maximize, close) on the right. The main content area has a light beige background. At the top, it says "Select the relevant Scrum Methodology practices from the list". Below this, there are five items, each with an unchecked checkbox and a label: "Scrum Meeting", "30 Day Sprint", "Sprint Planning Meeting", "Product Backlog", and "Self-organizing Teams". At the bottom of the window, there are two buttons: "Add to Report" on the left and "Main Menu" on the right.

Scrum Practices

Select the relevant Scrum Methodology practices from the list

- ☐ Scrum Meeting
- ☐ 30 Day Sprint
- ☐ Sprint Planning Meeting
- ☐ Product Backlog
- ☐ Self-organizing Teams

Add to Report Main Menu

9.6 Appendix G Methodology Description Tables

Table 9-3: Methodology Description for Extreme Programming

Parameter	Parameter Description	Relation to Customer Values and Priorities	Score
Methodology Philosophy	Technical mastery of programming, aesthetics of code, co-dependency between business and programmers, and building of teams at startup.		
Methodology Process	User writes stories on cards, developer estimates coding time per story, user chooses sets of stories for first iteration, user writes functional tests for each story, code is written.		
Methodology Techniques and Tools	Pair programming, test-driven development, refactoring, on-site customer, and collective ownership of code.		
Methodology Scope	XP is best suited for projects that require collocated teams of small to medium size team (three to twenty).		
Methodology Outputs	Working code for each small release or increment, and high quality programmers as a result of the learning process in pair programming.		
Adoption and Experience	Successfully applied by many companies (Highsmith, 2002a). Experiences of successful experiments with XP are recorded in Williams et al. (2000). XP has also been extensively applied in teaching programming in academic institutions, (Williams et al, 2000).		

Methodology Product	There are many books on XP, (Highsmith, 2002a), mentions some. Training can also be provided and there is online help on the XP website.		
Roles and Responsibilities	Programmer, customer, tester, tracker, coach, and manager.		
Support for Distributed Teams	XP has been used in distributed environments with tailoring o practice like pair programming and standup meetings.		

Table 9-4: Methodology Description for Adaptive Software Development

Parameter	Parameter Description	Relation to Customer Values and Priorities	Score
Methodology Philosophy	Based on these four goals: <i>adaptive culture</i> , which views organizations as complex adaptive systems. <i>Adaptive frameworks</i> , which offers a series of frameworks or models to help an organization employ adaptive principles. <i>Adaptive collaboration</i> , which establishes the interaction of people with similar and sometimes dissimilar interests to jointly create and innovate. <i>Adaptive scale</i> , which provides a path for organizations to adapt the approach on larger projects. <i>Adaptive management</i> , which replaces the culture of command–control management with a flexible management style that provides for wider participation in decision–making.		

Methodology Process	Based on the adaptive development life cycle, it is a model that supports the concepts and provides a practical way of moving from the conceptual to the actionable.		
Methodology Techniques and Tools	ASD applies the old technique of Joint Application Development (JAD) sessions. Planning is based on results it is also known as component-based planning, which plans based on a group of features.		
Methodology Scope	ASD is essentially a management philosophy for agile projects and therefore limited to project management activities.		
Methodology Outputs	The speculate phase delivers a list of features which are a result of intense planning with estimates on resources. The collaborate phase delivers working components for the quality review.		
Adoption and Experience	There have been more than one hundred projects that were using some form of ASD over the last decade. These projects span the range of internal IT applications, to commercial software products, to embedded software in medical instruments.		
Methodology Product	Books on ASD and training seminars provided by methodology author and other organizations.		
Roles and Responsibilities	Executive sponsor: as person with overall responsibility for the product. Participants in joint application development sessions. Other roles include: the developer, and the customer.		

Support for Distributed Teams	Not mentioned, but depends on situation.		
-------------------------------	--	--	--

Table 9-5: Methodology Description Table for Crystal

Parameter	Parameter Description	Relation to Customer Values and Priorities	Score
Methodology Philosophy	The crystal philosophy can be summed up in three fundamental principles; the first is a human-powered and adaptive principle, which means that the project success is based on empowering the people involved to make decisions and enhance their work The second is ultra-light, which means that for whatever project size and priorities, documentation and any paperwork must be kept minimal. The third principle is the ‘stretch –to–fit’ which means that it is better to start with something too light and add as needed.		
Methodology Process	Crystal Orange’s process is illustrated by the activities of one increment which are; starting from a requirements document, followed by staging which is the planning of the next increment and scheduling, next is the revision and review of each of the several iterations (each iteration has the following activities: construction, demonstration and review) in an increment.		

Methodology Techniques and Tools	The Crystal family in general does not prescribe particular techniques but emphasizes use of any techniques from the team's experience.		
Methodology Scope	The Crystal family of methodologies is essentially a project management philosophy that defines projects according to team sizes. It is rather difficult to spell out the scope of Crystal because the methodology provides a basis for selecting and tuning other methodologies.		
Methodology Outputs	Crystal Clear and Crystal Orange include the following outputs; release sequence, common object models, user manuals, test cases and migration code.		
Adoption and Experience	Crystal Orange principles have been used in one documented project, a two-year project called 'Winifred' and many others see section 3.6.2		
Methodology Product	The main product that can be purchased is the Crystal book Cockburn (2004b) and the other support texts are: (Cockburn, 1997 and 2002 and Highsmith, 2002a). There is a website that provides online help (http://alistair.cockburn.us/) and some help can also be obtained by subscribing to the crystalclear@yahoogroups.com group.		

Roles and Responsibilities	Crystal Clear has only one team and Crystal Orange has several teams. The roles are: <i>sponsor</i> finances the project and delivers the mission statement, <i>senior designer</i> maintains team structure, implements methodology, designs the system, <i>designer-programmer</i> creates screen drafts, design sketches and notes, common object model, source code, packaged system, migration code, and test cases, and <i>user</i> helps with use case and screen drafts.		
Support for Distributed Teams	Has not been used in distributed teams.		

Table 9-6: Methodology Description Table for Feature Driven Development

Parameter	Parameter Description	Relation to Customer Values and Priorities	Score
Methodology Philosophy	Centered on Object Oriented (OO) modeling of a project to represent it in the simplest possible model that graphically captures the realities of the project.		
Methodology Process	Starts from an overall model (domain object model) from which a features list is built through analysis of client valued functions for each domain area. Then the planning is based on these features (UML sequence diagrams). The final phase is the iterative design and build by feature (Detailed UML class diagrams) which results in the main build.		

Methodology Techniques and Tools	The strength of FDD is in its modeling techniques that are based on Object Oriented Analysis and Design. Detailed UML class diagrams and UML sequence diagrams combine with short iterative agile techniques to provide system development advantage to the development team.		
Methodology Scope	FDD is limited to small to medium sized teams (four to twenty people).		
Methodology Outputs	The initial output of the methodology is a comprehensive feature list that is produced from the analysis of object models. From the feature list an overall model is then produced during the planning by feature phase, which is given to class owners (individual developers). Design and build by feature results in the final code.		
Adoption and Experience	First used for a large complex banking project, Palmer and Felsing (2002).		
Methodology Product	The only available product for the methodology is the FDD book.		

Roles and Responsibilities	FDD has many roles here is just a few: Three groups of roles: key, supporting and additional. Project Manager is the administrator and financial leader. Chief Architect is a chief designer and runs workshops on design sessions. Developer Manager runs daily development activities and solves problems in development. Chief Programmer is an experienced developer who participates in requirements, design and analysis. Class Owner design, code tests and documents the system under supervision of the Chief Programmer.		
Support for Distributed Teams	Not mentioned.		

Table 9-7: Methodology Description Table for Agile Modeling

Parameter	Parameter Description	Relation to Customer Values and Priorities	Score
Methodology Philosophy	Focuses on working software as the main goal of modeling. The underlying principle is to keep the amount of models and documentation minimal but advanced enough to support meticulous design.		
Methodology Process	AMDD is not a complete methodology but a modeling framework hence it does not prescribe a process but works well with other agile methodologies.		
Methodology Techniques and Tools	No specific tools are mentioned except that there must be use of simplest available tools.		
Methodology Scope	No specific team size is mentioned but the methodology aims for small teams.		
Methodology Outputs	The methodology delivers minimal UML use case models and class diagrams.		
Adoption and Experience	It is relatively new (introduced in 2002), there is a lot that is not yet clear about the methodology.		
Methodology Product	There is an agile modeling book, Ambler (2002a) and a lot of online papers at the AMDD website (www.agilemodeling.com).		
Roles and Responsibilities	AMDD teams are expected to come from developers and project stakeholders.		

Support for Distributed Teams	Has not been used in distributed teams.		
-------------------------------	---	--	--

Table 9-8: Methodology Description Table for ICONIX

Parameter	Parameter Description	Relation to Customer Values and Priorities	Score
Methodology Philosophy	The ICONIX Unified Modeling approach's philosophy was derived from a combination of the most commonly used Object Oriented modeling techniques; Booch, Object Modeling Technique (OMT) and Object Oriented Software Engineering (OOSE). It centers on building fine object models through streamlining UML diagrams to a minimal but sufficient set.		
Methodology Process	The ICONIX process starts from prototypes or line diagrams of the screens, after user assurance of the prototype, followed by identification of use cases on use case diagrams, then comes the writing of use case text, next comes the refinement of text in robustness analysis, which leads to detailed design on sequence diagrams, followed by breakdown of the system along use case boundaries to derive a detailed object model.		

Methodology Techniques and Tools	UML diagrams, Use cases, Object Oriented Analysis and Design (OOAD) Techniques, Prototyping tools and techniques, Iterative/incremental development, Training tutorials.		
Methodology Scope	ICONIX works better with small to medium sized teams (four to twenty people) but it can be scaled to larger teams.		
Methodology Outputs	The code delivered is in essence a prototype of the proposed system. A high level version of the given use case model.		
Adoption and Experience	Based on UML and other OOAD methodologies that have been in use since the early 1990s. In many organizations where ICONIX is in use it is applied as tool for quickly changing use cases to code.		
Methodology Product	The methodology is sold in the form of CD-ROMs. Online help is also available including tutorials and training and the book, (Rosenberg and Scott, 2000 and Rosenberg, Collins-Cope, and Stephens, 2005).		

Roles and Responsibilities	Programmer writes the code. Trainer trains the project team on OOAD and ICONIX. Mentor is a fulltime onsite consultant. Analyst writes use cases. Designer changes use cases into design diagrams. User provides the requirements and reviews the design		
Support for Distributed Teams	Applicable to distributed teams.		

Table 9-9: Methodology Description Table for Scrum

Parameter	Parameter Description	Relation to Customer Values and Priorities	Score
Methodology Philosophy	The Scrum philosophy is based on the empirical model of industrial process control used to control complex and unpredictable systems. The fundamental notion of Scrum is that the set of variables that constitute software development such as people, requirements, resources, technology etc are not predictable and hence the use of rules for a defined process in software development would not be appropriated.		

Methodology Process	Starts from a product backlog (similar to use cases) list i.e. high-level design. Next is the game phase where the sprint backlog lists are used for analysis and design, this is the black box and unpredictable part of Scrum. Finally is the post-game phase where testing is done and closure of the release.		
Methodology Techniques and Tools	Planning game is a useful technique that results in a list of prioritized requirements called a Backlog list. Sprint is a technique where the prioritized list of requirements are developed into an executable product without allowing further changes to requirements within the thirty-day iteration, (Schwaber and Beedle, 2002). Daily Scrum is a great project monitoring technique that gives the management control of the project as they know what happens on a daily basis and have to solve the impediments. There are no specific tools mentioned but most project management tools would be useful.		
Methodology Scope	Teams should be small comprising about seven to ten people. It can be scaled to larger numbers. The methodology is aimed at project management in changing environments.		

Methodology Outputs	The methodology delivers three main deliverables: the Product backlog, which a document containing a list of all the features of the proposed system. Sprint backlog, this is a document that contains a prioritized list of the features that can be developed during a sprint. At the end of each sprint or several sprints if they are developed concurrently, a working product increment is delivered.		
Adoption and Experience	Provides project management framework that can be supported by any software development practices. Its success with specific projects is mentioned in (Abrahamsson et al, 2002 and Schwaber and Beedle, 2002).		
Methodology Product	The Scrum books, (Schwaber and Beedle, 2002 and Schwaber, 2004a) and there is some online help from the Scrum website [www.mountangoatsoftware.com/scrum/] including seminars and training for the various roles.		

Roles and Responsibilities	Scrum Master ensures Scrum practices and values are followed up to the end of the project. Product Owner manages the project and controls and makes visible the Product Backlog list. Scrum Team has authority to decide actions and is self-organizing so as to complete a Sprint. Customer participates in product backlog items, (Schwaber and Beedle, 2002). Management makes final decision and participates in setting of goals and requirements.		
Support for Distributed Teams	Originally meant for collocated teams.		

Table 9-10: Methodology Description Table for Dynamic System Development Methodology

Parameter	Parameter Description	Relation to Customer Values and Priorities	Score
-----------	-----------------------	--	-------

Methodology Philosophy	DSDM philosophy can be summed up as follows, (Stapleton, 2003): • Development is a team effort, which combines the customers' knowledge of business requirements with the technical skills of IT professionals. • High quality demands fitness for purpose and technical robustness. • Development can be incremental, i.e., not everything has to be delivered at once, and delivering something earlier is often more valuable than delivering everything later. • The law of diminishing returns applies, which means that resources must be spent developing features of most value to the business.		
---------------------------	---	--	--

Methodology Process	The model is centered on functional prototype and design prototype. There are five phases in the development process. The process starts with feasibility study, the next phase is the business study, the third phase is the functional model, the fourth phase is the design-and-build iteration. Testing is done within the functional model iteration and the design and build iteration. The last phase is the implementation, which mainly involves training the users and handing over the system to them.		
Methodology Techniques and Tools	: Business study workshop is a technique that brings together all stakeholders and elicits business requirements. DSDM training is often recommended in order to effectively introduce it in an organization. DSDM does not specify tools but provides an ideal support environment, which spells out a need for requirement analysis tools, system prototyping tools, design tools, construction tools, testing tools, and reverse engineering tools.		
Methodology Scope	Team size varies from two to six people but there may be many teams in a project. DSDM has been applied to small and large projects. DSDM is mainly applicable to the development of user interface intensive systems and complex business applications, but may also be used for non-IT projects.		

Methodology Outputs	Feasibility report, outline for development or prototype, functional model prototype, functional prototyping review documents, risk analysis for further development document, design prototype, user manual, Project review report, and final tested system.		
Adoption and Experience	DSDM has been in use in Europe since the mid 1990s. It has proved to be a fine alternative for RAD. The DSDM manual spells out the adoption procedure.		
Methodology Product	When you join the DSDM consortium you get the following; online DSDM manual, UK government white papers and templates, a wide support network of other members, and receive regular updates on the latest news and events from the consortium.		
Roles and Responsibilities	Developers and senior developers are involved in development. Technical coordinator defines system architecture and ensures technical quality in the project. Ambassador user brings the knowledge of users into the project and reports project progress to other users. Visionary has accurate view of business objectives of the system and project. Executive Sponsor has the financial authority and responsibility		
Support for Distributed Teams	Not mentioned.		

Table 9-11:Methodology Description Table for Lean Development

Parameter	Parameter Description	Relation to Customer Values and Priorities	Score
Methodology Philosophy	LD's philosophy is based on the adoption of a radical change tolerant approach to risk leadership in software development. The fundamental principle is to create change tolerant software with one-third the human effort, one-third the development hours, one-third the investment in tools and methods and one-third the effort to adapt to a new market environment.		
Methodology Process	LD's view of reality is: creation of visible customer value rapidly, building change-tolerant software, creating only necessary functionality and no more, and aggressiveness, stubbornness, and belief in meeting LD's overall goals.		

Methodology Techniques and Tools	Value analysis process is used to prove the value of every artifact. Rapid delivery of product is another technique that races with changing requirements, the faster you deliver the least the change in requirements.		
Methodology Scope	Any software development project where there is need for radical change. Focused at company CEOs. No team size specifications because LD is more of a software development management philosophy than a methodology.		
Methodology Outputs	Artifacts can be released incrementally as the customer requests before the final product.		
Adoption and Experience	Based on Lean Manufacturing with origins as far back as 1935. Poppendieck and Poppendieck (2003) mention many projects where LD has been applied.		
Methodology Product	The LD book, (Poppendieck and Poppendieck, 2003), and some online help and tutorials (www.poppendieck.com). Tom and Mary (the authors of LD) also provide a lot of in house training in organizations that need to adopt LD.		
Roles and Responsibilities	No specific mention of roles and responsibilities except that LD is aimed at CEOs before it can be implemented in the organization. Project manager and customer are also mentioned in (Highsmith, 2002a).		
Support for Distributed Teams	Can be adapted for distributed teams.		