The Agile Methods: An Analytical Comparison of Five Agile Methods and an Investigation of Their Target Environment

A thesis presented in partial fulfillment of the requirements for the degree of

Master of Information Sciences

in

Information Systems

at Massey University, Palmerston North, New Zealand

Diane Elizabeth Strode

2005

Abstract

This study defines the systems development methodologies named agile methods and investigates the environmental conditions where agile methods are most suitable.

A definition of agile methods was developed using an analytical comparative framework to investigate five of the earliest published agile methods; Dynamic Systems Development Method, Extreme Programming, Scrum, Adaptive Software Development, and Crystal Methods. The framework decomposed each method into its component parts; philosophy, models, techniques, tools, scope, outputs, practice, and the extent to which the method may be adapted to a situation. Based on this analysis and a literature review, a theoretical model of the target environment for agile methods was developed. This theoretical model is a proposed set of organisation, people, project, technology, and domain factors that relate to the successful use of an agile method.

A mixed method research methodology was used. A qualitative design, consisting of positivist case studies, was used to test the theoretical model. Data was gathered from nine software development projects, both agile and non-agile, using questionnaires and interviews of project leaders. Then cross-case analysis was carried out on each project factor in the theoretical model. The relationship between environmental factors and agile method usage was investigated using non-parametric quantitative data analysis. This led to a revised model of the target environment for agile methods. The empirical data showed that specific organisational culture factors correlate with effective use of an agile method. These include the organisational characteristics of feedback and learning, teamwork, empowerment of people, collaboration, leadership, loyalty, and a results-oriented culture that values entrepreneurship, innovation and risk taking.

This research is significant for method users, those carrying out empirical research into agile methods, and those carrying out studies of systems development methodologies.

Acknowledgements

Many people helped me with this study. I would particularly like to thank the participants, who gave their time to complete a detailed and taxing questionnaire and interview. Their interest and supportive comments helped me to maintain my own interest and to see that this is a study worth undertaking.

I would also like to thank these people from the Massey University Department of Information Systems for their assistance and support: my supervisor Alexei Tretiakov, Lecturer, for guidance and help in finding resources whenever they were needed, my co-supervisor, Roland Kushak, Associate Professor, for targeted critique of the research design, Barbara Crump, Senior Lecturer, for assistance in qualitative research methods and voicing such a positive attitude to the goals and design of this study, and Errol Thompson, Lecturer, for assistance with the pilot study. I am grateful to Erkki Sutinen, Professor of Computer Science at the University of Joensuu, Finland for his suggested improvements.

The following people at Whitireia Community Polytechnic, Faculty of Business and Information Technology provided invaluable help. Pippa Coard, Lecturer, for providing pragmatic, focused and detailed assistance in critiquing the questionnaire, along with positive encouragement and support, Sue Chard and Brenda Lloyd, Senior Lecturers, for helping to locate software developers and with the statistical analysis, and Trish Brimblecombe, Associate Dean, for providing the opportunity, financial support, and the time to devote to this thesis, and assisting to locate software developers. I would also like to thank my colleagues and the senior students, who provided feedback on aspects of this thesis.

My thanks also go to Shane Hastie of Software Education Group, Wellington, for assistance in finding agile method projects.

I would also like to thank Chris Brosnahan for assisting with developing the instrumentation and for his patience.

Table of Contents

Figures

Tables

Chapter 1 INTRODUCTION

Background

Systems development methodologies are perpetually important in the field of information systems. This is because of the affect of innovation and change in business and technology on the process of developing software systems. As business needs and technology change so the methodologies and methods used to produce computerised business systems also change. In 1994 over one thousand brand-named methodologies were reported to be in use around the world (Jayaratna, 1994) and in the decade since then more have been developed to cater to the object-oriented paradigm (Graham, Henderson-Sellers, & Younessi, 1997). During the mid 1990s systems development methodologies called 'agile methods' were developed. The agile methods are the subject of this study.

The agile methods are considered to be a reaction against prescriptive traditional methodologies and a response to the need for quality, rapidly developed systems (J. Highsmith, 2003). This study shows that the agile methods are a refinement and amalgamation of earlier methodology concepts and practices, and have developed in response to a new technology paradigm founded on development for Internet and distributed technologies (Lytinnen & Rose, 2003a).

The agile methods are a set of systems development methodologies that share common characteristics. Agile methods were named in 2001 by a group of method authors (AgileAlliance, 2001). The most well-known agile method at that time was Extreme Programming (Beck, 2000) and the earliest agile method is Dynamic Systems Development Method (Stapleton, 1997) first published in 1995. Agile methods aim to support the software team to achieve effective delivery of software and they focus on team communication and other mechanisms for managing iterative and incremental development. Many books and articles describe specific agile methods, describe agile methods in general, and make statements about their efficacy, when to use them, and how to use them. Much of this documentation lacks a sound basis in theoretical or empirical research. An overall understanding of the body of knowledge about agile

methods is missing from many of these publications, and claims for their efficacy under certain circumstances are untested.

An overview of the field drawn from the body of literature published about agile methods would be a valuable addition to the knowledge about these methods. This in turn would allow the agile methods to be accurately categorised within the existing body of knowledge of systems development methodologies. A full description and classification of agile methods would provide a record for future developers to draw upon when new ideas, technologies, and methods arise to take their place. This study aims to contribute to this body of knowledge.

Research Objectives

The first objective is to find out what an agile method is, if there are characteristics that define agile methods as a group, or if they are all distinct methods and their grouping is arbitrary. I also aim to find out whether agile methods are systems development methodologies, as defined by Avison and Fitzgerald (1995b), or arbitrary combinations of software development techniques. This objective becomes research question 1; what is an agile method?

The second objective is to find out if there are environments where agile methods are most suitable and to investigate what constitutes that environment. Boehm and Turner (2004) provide guidance on project environments where agile methods are more or less suitable but empirical evidence for their ideas is not available. This objective becomes research question 2; for what type of environment are agile methods suitable?

Overview

This study uses a mixed-method research design. The first objective is addressed using an analytical framework to investigate the agile methods. The second objective is addressed using a research methodology suitable for investigating systems development methodologies *in situ*.

The first objective is met by carrying out a detailed analysis of five agile methods using an analytical framework to determine the characteristics of each method. There are

about 13 published agile methods, the exact number is a matter for debate in the literature, and I chose the first five agile methods published for this study. This provides a large enough sample for a detailed analysis and overall conclusions to be drawn, while staying manageable.

The analytical framework developed by Avison and Fitzgerald (2003b) is adapted and applied to each of the five agile methods to decompose each one into its component parts: philosophy, models, techniques, tools, scope, outputs, practice, the extent to which the method may be changed and adapted to a situation (tailorability). The primary source of the method description is, wherever possible, a first edition of the book describing the method. This analysis produces a list of techniques used in each method, a list of common properties of agile methods, a summary of the content of each method in a form suitable for easy comparison, and a description of possible method problems. The analysis also contributes to a theoretical model of the target environment for agile methods. The agile method authors say that their methods are only suitable in certain circumstances but these circumstances are not always clearly described (Beck, 2000; Cockburn, 2002; J. Highsmith, 2002; Stapleton, 1997). I have called these circumstances the 'target environment', which is the set of factors where the method is most effective.

Application of the analytical framework and reviewing the literature on agile methods produces two sets of target environment factors. These factors are amalgamated into a theoretical model of the target environment for agile methods. This theoretical model is a proposed set of organisation, people, project, technology, and domain factors that relate to the successful use of an agile method.

The second objective is met by investigating the target environment for agile methods using a qualitative design consisting of a positivist multi-case study designed to test a theoretical model (Yin, 2003). I carry out nine case studies of software development projects from different organisations and compare the environment of each of these projects with my theoretical model of the target environment for agile methods. A cross-case analysis uses quantitative data analysis methods to show correlation when the sample size and type of data allow for such analysis. This leads to a revised model of the target environment for agile methods based on empirical data.

Significance

This research is significant in three areas. For method users, for those carrying out empirical research into agile methods, and for those carrying out studies of systems development methodologies.

Currently those who want to use an agile method must read a number of books to work out which method will suit their purpose. This study provides a number of useful artefacts for this group; a structured summary of five agile methods and a detailed description of each method organised in a way that allows for easy comparison of methods, an organised summary and critique of the current empirical research into agile methods, and empirical evidence about the target environment for agile methods. This empirical evidence will aid the practitioner in deciding if agile methods will work for them in their particular environment.

Those who want to study agile methods need to know what each method consists of in order to design effective studies of uptake in industry settings, to develop process maturity models, or to amalgamate methods for different purposes. This study provides a list of techniques that are used in each of the five methods. It also provides a survey instrument, parts of which are suitable for assessing the use of an agile method *in situ* and for assessing the environment factors of system development projects.

For those who want to classify agile methods this study provides understanding of what constitutes individual agile methods and agile methods as a group. Based on this classification and comparison, the agile methods can be compared with other systems development methodologies.

Thesis Structure

This thesis has six chapters related to the investigation of the agile methods. This introductory chapter is followed by a literature review that provides a brief history of systems development methodologies and explains the precursor methodologies and movements on which the agile methods are based. The agile methods are described, followed by the current classifications and comparative studies available to date. Method combinations are described and then the empirical literature on agile methods and

individual agile method techniques is organised and critiqued. This chapter concludes with a discussion of the need to understand the target environment where agile methods are considered to be most suitable, and an initial theoretical target model of the environment for agile methods is presented. This leads to a description of the research questions, sub questions and propositions.

Chapter 3 describes the methods used in the study, the mixed qualitative and quantitative nature of the study, and how each of the research questions is addressed. This chapter focuses mainly on the target environment for agile methods which is investigated using a case study method. Validity is addressed, how cases are selected, and the methods for data gathering and analysis are described. The instrumentation, a questionnaire and interview, is described.

Chapter 4 describes the use of analytical frameworks for investigating systems development methodologies, current frameworks in use, and the framework used in this study. Application of the analytical framework provides a detailed analysis of each method and identification of additional environmental factors that contribute to the theoretical model of the target environment for agile methods.

Chapter 5 describes the results for each of the research questions and sub questions. First the results from application of the analytical framework are provided followed by the results of the case studies. The results from the questionnaire and interview are used to address each of the criteria of the theoretical model of the target environment for agile methods. The discussion of these results is integrated with their presentation. A revised version of the theoretical model is then presented based on empirical data.

The final chapter presents the conclusions of the study. The contributions are described followed by a discussion of the limitations of the study along with indications for future research.

Chapter 2 LITERATURE REVIEW

Literature Review Structure

This review first describes what constitutes a systems development methodology (methodology) based on existing definitions. Then the history of systems development methodologies is described in order to explain the background of agile methods and the precursor ideas and practices on which the agile methods are built. The agile methods are introduced with a history of their development. The literature classifying and comparing agile methods with each other and other systems development methodologies follows. Proposed combinations of agile methods and studies of how agile methods can be combined with process improvement methods are described. The literature on agile method adoption in education and industry settings is analysed along with the reported scalability issues. Then the research into individual agile method techniques is described. Studies into the target environment for agile methods are analysed and a theoretical model of the target environment for agile methods developed from the literature review is then introduced. Finally the research questions and their importance are described.

Terminology

The terms 'methodology' and 'method' used in the study of systems development methodologies are not clearly defined (Avison & Fitzgerald, 2003a; Brinkkemper, 1996; Wynekoop & Russo, 1995). Frequently the two terms are used interchangeably (Cockburn, 2003; Graham et al., 1997) and Jayaratna (1994) stated "the term 'methodology' is pragmatically well established within the field of information systems to mean the same as method". I treat these terms interchangeably in this thesis.

The purpose of a systems development methodology is to provide guidance for systems development and many definitions have been published. Maddison ((1983) cited in Avison and Fitzgerald (1995a)) wrote one of the earliest definitions; 'A recommended collection of philosophies, phases, procedures, tools, documentation, management, and training for developers of systems'. Graham et al. (1997, p. 2) defined methodologies for the object-oriented paradigm, which agreed with Maddison but omitted any need for a philosophy and included quality assurance, metrics and reuse. Cockburn (2002, 2003)

agreed with Maddison, but added people, skills, roles and team values to his methodology definition. Though Cockburn (2002) makes no mention of philosophy his list provides clues to one of the philosophies of agile methods. This is that people are an important part of a methodology. This factor was recognised in the work on methodology comparisons by socio-cyberneticists in the early 1980s who noted that systems design methodologies at that time often "do not recognize any contingencies related to human beings" and were not "human-sensitive" (Iivari & Kerola, 1983, p. 129).

A core of common characteristics applies to all methodology definitions and has persisted over the last 20 years. I use the definition of Avison and Fitzgerald (1995b) because it is explicit and comprehensive:

> An information systems development methodology can be defined as a collection of procedures, techniques, tools, and documentation aids which will help the systems developers in their efforts to implement a new information system. A methodology will consist of phases, themselves consisting of sub-phases, which will guide the systems developers in their choice of the techniques that might be appropriate at each stage of the project and also help them plan, manage, control and evaluate information systems projects (Avison & Fitzgerald, 1995b, p. 10).

The authors included the proviso that a methodology is based upon a philosophical view and a set of stated objectives and is not just a recipe to be followed without adaptation to the particular situation in which it is used.

History of Methodologies and Influences for Change

Avison and Fitzgerald (2003c) divide the history of system development methodologies into eras:

1. A pre-methodology era from 1950-1970.
2. An early methodology era from late 1970s to early 1980s.
3. A methodology era.
4. A post-methodology era from the late 1990s until 2004.

During the pre-methodology era, systems developers generally did not use formal development methodologies. Boehm (1988) call this the 'code and fix' stage which had a

number of problems that later stages attempted to remedy. The first systems developed in the 1950s were created by and for scientists and the needs of business played almost no part. When business began to use computers to process data the technology was limited by hardware capability, and technical problems and keeping systems operational were primary concerns. Users had almost no voice in systems development and the business implications of information systems were seldom considered. By the end of the 1960s the National Computing Centre in the United Kingdom recommended formal phased methodologies consisting of:

- "Feasibility study
- System investigation
- Systems analysis
- Systems design
- Implementation
- Review and maintenance". (Avison & Fitzgerald, 1995b, p.20)

This was called the 'the system development lifecycle' (SDLC) or 'waterfall model' and introduced standards and discipline to the process of creating software. At the end of the pre-methodology era, in 1970, Royce (republished by Royce (1987)) published his version of the SDLC with eight sequential phases and two types of iteration in the lifecycle; production of an early pilot version, and the possibility of returning to earlier phases when needed.

The second era saw phased development published widely in the developer community. CRIS (Comparative Review of Information Systems Design Methodologies) conferences were held from 1982 to 1984. Their purpose was to define, document, analyse and compare the systems design methodologies of the period. The first conference described 13 well-developed systems design methodologies and the second analysed the features of 30 methodologies (Olle, Sol, & Tully, 1983; Wasserman, Freeman, & Porcella, 1983). The third conference was designed to improve the practice of using methodologies and the fourth addressed CASE tools and technologies. The conferences were criticised for not achieving their stated goal of methodology convergence (Olle et al., 1983) and for their lack of influence on the use of methodologies by practitioners (Avison & Fitzgerald,

1995a) however, I believe the goal of providing a full feature analysis of a number of methodologies was clearly achieved.

The idea of methodology paradigms also occurred about this time (Wood-Harper & Fitzgerald, 1982). Methodologies belonged to one of two paradigms: science-based or 'hard' methodologies that treated the creation of information systems as engineering problems solvable using engineering principles, and 'soft' or systems methodologies that treated systems as business systems with the software solution a subset of the whole system creation process.

Methodologies proliferated in the methodology era and important new ideas were published. Prototyping, evolutionary development, rapid application development (RAD), iterative and incremental delivery and the development of object-oriented analysis and design techniques occurred during this era. Both throw-away and incremental (or evolutionary) prototyping became accepted techniques (Crinnion, 1992). Boehm (1988) published the spiral software process model which introduced the concept of evolutionary development with incremental prototyping and risk analysis at the beginning of each new increment. J. Martin (J. Martin & Finkelstein, 1981) author of the Information Engineering methodology published Rapid Application Development in 1991 (J. Martin, 1991). This formalised the ideas of RAD which forms the basis of Dynamic Systems Development Method (Stapleton, 1997), an agile method. The philosophy of RAD was to speed up development by using a combination of techniques and tools which were believed to substantially reduce development time. Methodologies from earlier eras continued to be used and refined to incorporate some of these newer ideas.

Crinnion (1992) identified the directions in which the well-established methodologies moved during the years 1982-1992 as follows:

1. "Towards providing greater flexibility and scalability of approach.
2. Towards faster development and delivery of a system.
3. Towards greater involvement of the business user in all stages of the development".

(Crinnion, 1992, p. 3/3)

He noted that typically methodologies were adopted in a cookbook fashion which meant that all systems within an organisation could be developed in the same way. This

provided effective risk management but imposed a resource overhead on each individual project. But the benefits were considered to outweigh the overhead in a climate of high failure rates. As an organisation's experience with a methodology grew and successful systems became the norm, the methodology was tailored and prescriptive elements dropped. In addition as the SDLC methodologies (for example SSADM (Structured Systems Analysis and Design Methodology) and IE (Information Engineering)) incorporated RAD ideas, flexibility and scalability became possible. These methodology changes were brought about by the pressure on business to become flexible to meet market demands and to increase the speed of delivery of systems. The SDLC could not accommodate this faster development pace because its long development time-frame meant that by the time a system was delivered the requirements of the business had often changed.

Another gradual change was to greater user involvement in systems development. Users moved to more active participation involving newer communication techniques such development workshops, structured walkthroughs and prototyping sessions. User-driven development became a possibility during this time (Crinnion, 1992).

Crinnion (1992) also analysed the reasons for, and the effects of, evolutionary development. He believed that evolutionary development is based on meeting the needs of business whereas the SDLC-based methodologies are based on controlling systems development. He observed that as business systems evolve to meet changes in business processes these changes give rise to changes in system requirements. Such changes occur at any time and one of the ways of dealing with this constant change is to involve the business user in all stages of development. Another mechanism for dealing with change is incremental delivery and he proposed that the effect of this type of delivery would be that analysis, design, and construction would all take place simultaneously, blurring the distinction between development and maintenance. He also observed that a side-effect of evolutionary development would be faster development speed. Crinnion (1992) believed that the most important attributes of evolutionary development would be the overlapping of traditional life-cycle stages and incremental delivery.

The most powerful influences on business computing at that time were thought to be the move to object-oriented and knowledge-based systems (Crinnion, 1992). This conjecture has been partially fulfilled. In the 1990s the object-oriented methodologies and

methodologies for component-based systems and object-oriented frameworks evolved rapidly (R. J. Wirfs-Brock & Johnson, 1990). These methods were all developed to provide object-oriented analysis, design and project management techniques for systems built using the object-oriented languages which matured during the 1990s (Graham et al., 1997).

Common characteristics of the object-oriented methods were the use of a specific modelling language, a business concept to drive development, and an evolutionary lifecycle using iterative and incremental development. These methods primarily used a precursor form of the Unified Modeling Language (UML) or UML itself. Various approaches were used to organise development and break it down into manageable blocks; the use-case driven approach (Jacobson, Christerson, Jonsson, & Overgaard, 1992), the responsibility-driven approach (R.J. Wirfs-Brock & Wilkerson, 1989), the model-driven development (Embley, Kurtz, & Woodfield, 1992) or the contract-driven approach (Larman, 2002).

Problems with managing iterative and incremental development were discussed at the OOPSLA[1] workshops in the early 1990s (Coplien, Hutz, & Marykuca, 1992; Coplien, Hutz, & Winder, 1994) and the participants reported that object-oriented development is effectively supported by iterative and incremental development but there is a conflict with project management ideals. The link between iterative and incremental development was explicitly defined at these workshops; to control iteration (so that it does not loop forever) incremental development is used to enable management to watch the progress of functionality as it is developed.

The characteristics of the post-methodology era (in the 2003 edition of their classic text they rename this era the "era of methodology reassessment" (Avison & Fitzgerald, 2003c, p. 542)) are based on the findings of a series of studies carried out by Fitzgerald (1997, 1998, 2000). These studies explored how systems development methodologies are used in practice and involved large scale surveys and interviews with developers. Fitzgerald found that due to disappointing productivity gains and overly complex methodologies requiring extensive training and tools, a 'developer backlash' had occurred where

[1] Annual conferences on Object-Oriented Programming, Systems, Languages and Applications

methodologies were rejected. Conclusions from the 1997 study were based on a small number of interviews at eight different types of company. The conclusions were:

- Development based on a formal methodology is the most common use of methodology. The methodology used is tailored to accommodate the unique attributes of the project in which it is used.
- Experienced developers are more likely to use a methodology and to tailor it for use.
- Developers deliberately omit irrelevant aspects of a methodology during a project because they understand they are not valuable in that particular project.
- Methodologies are often specified at a high level specifying broad activities and objectives and must therefore be tailored for the environment when they are used.

Fitzgerald (1998) then investigated the literature published between 1989 and 1995 into the nature of systems development, the use of systems development methodologies in practice, and the arguments for and against the use of methodologies. He also carried out a rigorous quantitative study involving a survey of 776 individuals responsible for systems development and in-depth interviews with 16 people. Conclusions from this study reinforced the four findings above. A further conclusion was that "there is a need to 'advance the clock' by deriving sensible methodological canons more suited to the needs of the current development climate" (Fitzgerald, 1998, p.111). The development climate at that time was that of customisation of packages, outsourcing and integration. Another telling finding, given the emergence of agile methods with this exact philosophy at about the same time, was that "there may be a sense in which "good enough" systems can be developed in an appropriate time-scale, rather than striving toward delivering optimum solutions in an unreasonable time-scale" (Fitzgerald, 1998, p. 112).

Supporting Crinnion's (1992) ideas, Fitzgerald (2000) found that the business climate in Ireland and Europe required "that organizations act more effectively in shorter time frames". He also noted that the expectation that employees will "plod robotically through standardised checklists, is not valid" (Fitzgerald, 2000, p. 182). He believed that methodologies may need to mix both top-down SDLC approaches and bottom-up prototyping approaches in order to improve flexibility, that methodologies need to be simplified, and that adaptation of methodologies (tailoring to the specific situation) needed to become acceptable. Methodologies should define high-level goals and

deliverables and allow the developer discretion in how a step is achieved. Furthermore if organisations are acting within shorter timeframes in the current business climate, then methodologies must cater for this. Fitzgerald (2000) made two recommendations; further research into the current systems development environment within organisations is needed, and that, 'The next generation of methodologies requires a new set of foundational concepts,' that are based on, 'the "best practice" development situations which prevail at present' (Fitzgerald, 2000, p. 183).

It was in 1999 that Beck first described the fundamentals of Extreme Programming as an answer to the problems of dealing with changing requirements and the difficulty that programmers have in accurately assessing their progress towards completion. The methodology advocated shortened development cycles, negotiated quality, and the adoption of a set of practices based on existing techniques used in software engineering.

In the early 2000s Fitzgerald's call for new systems development methodologies was reiterated (Lytinnen & Rose, 2003a, 2003b; Marchesi & Succi, 2003). Lytinnen and Rose (2003b) showed that Internet computing[2] is a disruptive IS innovation transforming the applications developed, the application services provided and the development practices of organisations. New methodologies were needed to support Internet development because, as Lytinnen and Rose (2003b) found in their study of eight 'leading edge' system development organisations in the USA and Finland, "all companies with rigid methodologies admitted that their established development methodologies did not work" (p. 580) in the new environment and were striving to simplify their processes. The authors developed and tested a 3-set model of disruptive IT innovations (see Figure 1). The model was based on a literature survey and a qualitative case study of companies developing systems based on Internet technologies. Their survey covered the period 1998 to 2002 and was used to find characteristics of system development and services that were observed to change as a result of adopting Internet computing. These characteristics were then investigated using a multi-site case study to test the validity of the model. The qualitative study was based on the findings from interviews of a number of people within

[2] Internet computing is "a holistic concept that draws upon all protocols that enable computing in an open, distributed, and heterogeneous environment running on top of Internet-based transmission protocols." (Lytinnen & Rose, 2003b)

each organisation. Lytinnen and Rose (2003b) concluded that Internet computing has led to significant changes in development processes, in particular it has drastically shortened delivery times, caused methodology changes, and increased the specialisation of development team members when compared to traditional system development. They found that traditional methodologies were not suitable to plan, coordinate and control these projects and more flexible, simple, repeatable, and agile methods, were needed. Specialisation of the development team was occurring requiring expertise in arts/media, telecommunications, business process reengineering, and software development in particular technologies (e.g. J2EE, XML, components, database).

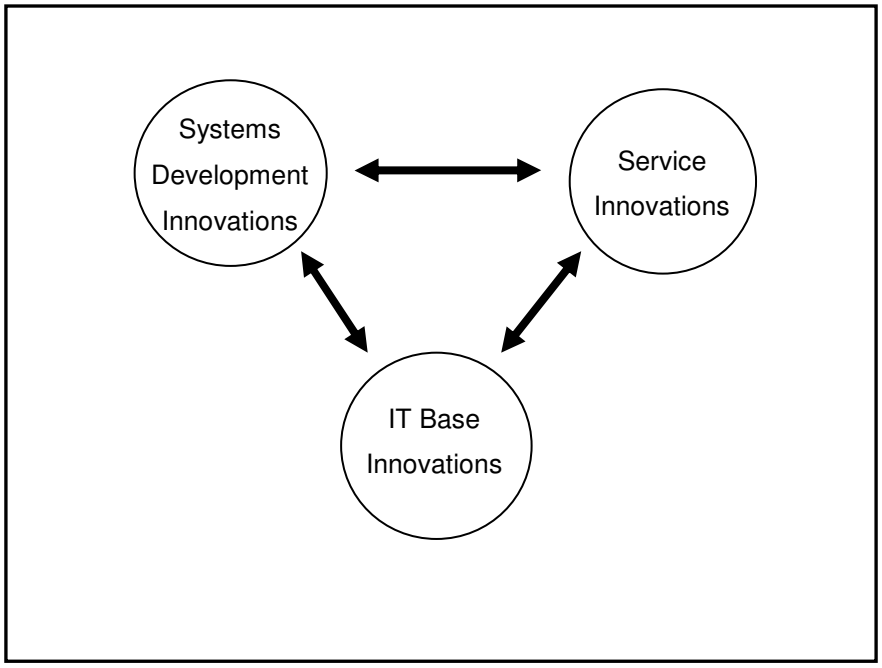

Figure 1: The Three-Set Model of IT Innovation (Lytinnen & Rose, 2003b)

Unchanged in this environment were the traditional needs to control development outcomes, to address risk and uncertainty, and to simultaneously manage cost, quality and speed of delivery. The main limitations of the study, as noted by the authors, were the generalisability of the findings to organisations that are not early adopters, the lack of a control group, and the generalisability of the study to earlier disruptive innovations. The second limitation is overcome to some extent as the organisations studied had both traditional and Internet development occurring within the organisation, and findings were based on interviews with developers who had experienced both environments and had observed the different characteristics of the Internet environment compared with the

traditional environment. The study was only concerned with organisations developing Internet services and not those consuming those services, so the disruptive nature of the Internet computing was only studied from one perspective. Selection of organisations was subject to contextual bias because early adopters are more biased towards showing successful adoption.

Further investigations into the effect of Internet development were carried out by Baskerville and Pries-Heje (2004) who used case studies of software development in companies in the USA and Denmark. The goal of their research was to identify the enduring characteristics of short cycle time Internet development based on findings from organisations who were developing commercial software applications based on Internet connectivity. They found five distinguishing characteristics of short cycle time Internet systems development:

1. The software system is delivered into a business environment that is highly competitive or dependent on rapid technology changes.
2. Systems must be provided quickly, and to achieve this, prototyping, release orientation (delivery of partially completed software product), and parallel development are used.
3. A standardised architecture (normally 3-tier) ensures components work together, controls design conflicts, and allows for scalability and maintainability.
4. The quality of the software product is negotiable; functionality or faster cycle time takes precedence over quality.
5. Expert, experienced and exceptional developers are necessary in this environment.

These characteristics together form a "major new form of systems development" (Baskerville & Pries-Heje, 2004, p. 260) involving a change in development philosophy from that of 'building' systems to that of 'growing' systems whereby traditional maintenance and replacement of systems is subsumed into a "continuous redevelopment of the entire organizational portfolio of systems" (Baskerville & Pries-Heje, 2004, p. 262). I believe that these are the characteristics that have led to the development of agile methods and that the agile methods provide structure and control for the development of software systems in this type of environment.

This history of the main types of methodology and the major influences on methodologies leads to the next section which discusses the agile methods. The agile methods are a group of systems development methodologies that are not analysis methodologies, design methodologies, or object-oriented methodologies, but take parts of many of the methodologies used to date and recombine them in new ways suitable for the development of software-based systems in the business and technical environment of the late 1990s and early 2000s.

The Agile Methods

I have identified twelve agile methods published from 1997 to 2003. This list comes from an analysis of the agile methods investigated by Abrahamsson, Salo, Ronkainen, & Warsta (2002), Cockburn (2002), Highsmith (2002) and Abrahamsson, Warsta, Siponen, & Ronkainen (2003) (see Table 1). I have made a distinction between publication of a journal and a book because the journal is usually published first but the book provides more substantial detail about the method.

Table 1: Agile methods organised by earliest publication date

	Agile Method	**Acronym**	**Primary Source**	
			Journal Article	**Book**
1	Dynamic Systems Development method	DSDM		DSDM ("Dynamic Systems Development Method, Version 2", 1995) Stapleton (1997)
2	Crystal methods	Crystal		Cockburn (1998) Cockburn (2002)
3	RUP (configured)	dX		Martin (1998)
4	Extreme Programming	XP	Beck (1999)	Beck (2000)
5	Adaptive Software Development	ASD		Highsmith (2000)
6	Scrum	Scrum		Beedle, Devos, Sharon, Schwaber, & Sutherland (1999) Schwaber & Beedle (2002)
7	Pragmatic Programming	PP		Hunt and Thomas (2000)
8	Internet Speed Development	ISD	Cusumano & Yoffie (1999)	Baskerville & Pries-Heje (2001)
9	Agile Modeling	AM		Ambler (2002)
10	Feature Driven Development	FDD		Palmer & Felsing (2002)
11	Open Source Software Development	OSS	Sharma, Sugumaran, & Rajagopalan (2002)	
12	Lean Development	LD		Charette (2002) Poppendiek & Poppendiek (2003)

A manifesto for agile software development was published in 2001 (AgileAlliance, 2001) (see Table 2) listing the principles common to all agile methods (see Table 3). The authors of the manifesto were people who had published individual software development methods with similar characteristics; each of these methods is based on practitioner experience and evolutionary development practices with a focus on early delivery of quality software (AgileAlliance, 2001).

The name 'agile methods' came from the field of flexible manufacturing and was considered an improvement on the name lightweight. Initially the methods were called lightweight to distinguish them from heavyweight methodologies. Heaviness is a characteristic of prescriptive methods requiring the production of many non-software artifacts, mainly documentation, during development. Examples of heavyweight methodologies are SSADM (Eva, 1994), Information Engineering (J. Martin & Finkelstein, 1981), Unified Software Development Process (Jacobson, Booch, & Rumbaugh, 1999), and OPEN (Graham et al., 1997).

Table 2: Agile methods manifesto (AgileAlliance, 2001)

Manifesto for Agile Software Development

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.

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

Table 3: Principles of agile methods (AgileAlliance, 2001)

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 good 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 tunes and adjusts its behavior accordingly.

The Evolution of Agile Methods

The agile methods evolved from existing methodologies and techniques in software engineering and information systems. The precursor methods and major influences on each of the agile methods is available in an evolutionary map developed by Abrahamsson et al. (2003). My analysis of their study found four main influences on agile software development: object-orientation, evolutionary development, internet technologies, and methodology engineering. The object-oriented influences were from UML (Rumbaugh, Jacobson, & Booch, 1999) and RUP[3] (Kruchten, 2000). Evolutionary approaches include the influence of Boehm's spiral model (1988), prototyping and RAD. The internet technologies include ideas from Microsoft's sync and stabilise method (M. Cusumano & Selby, 1997) and Internet speed development (Baskerville, Levine, Pries-Heje, Ramash, & Slaughter, 2001; Baskerville & Pries-Heje, 2001; M. A. Cusumano & Yoffie, 1999). The methodology engineering ideas came from Kumar and Welke (1992) and amethodical development (Baskerville, Travis, & Truex, 1992; Truex, Baskerville, & Klein, 1999; Truex, Baskerville, & Travis, 2001). This evolutionary path shows that agile methods are an amalgamation of previous methodologies, ideas and practices from software engineering.

Influences from information systems on the agile methods include the Participatory Design movement (Kuhn & Muller, 1993) and ideas from soft systems methodology (Checkland, 1999). Participatory Design (Kuhn & Muller, 1993) is a movement that began in Scandinavian countries in late 1980s focussing on stakeholder participation in the development of information technology solutions. This movement encompassed a number of techniques and methods to support user participation including ethnographic methods, a variety of prototyping techniques such as collaborative prototyping, video prototyping and storyboard prototyping, future workshops, and card games. Different techniques were designed for different phases of the lifecycle and involved different degrees of participation ranging from users participating in design activities to designers participating in user's workplaces.

[3] Rational Unified Process™

The influence of Checkland's soft systems methodology (Checkland, 1999) and ideas of human activity systems being holistic and evincing emergent properties appear in Crystal methods (Cockburn, 2002) and ASD (J. A. Highsmith, 2000).

Abrahamsson et al. (2003) analysed which of the agile methods contributed to the agile manifesto (see Figure 2).

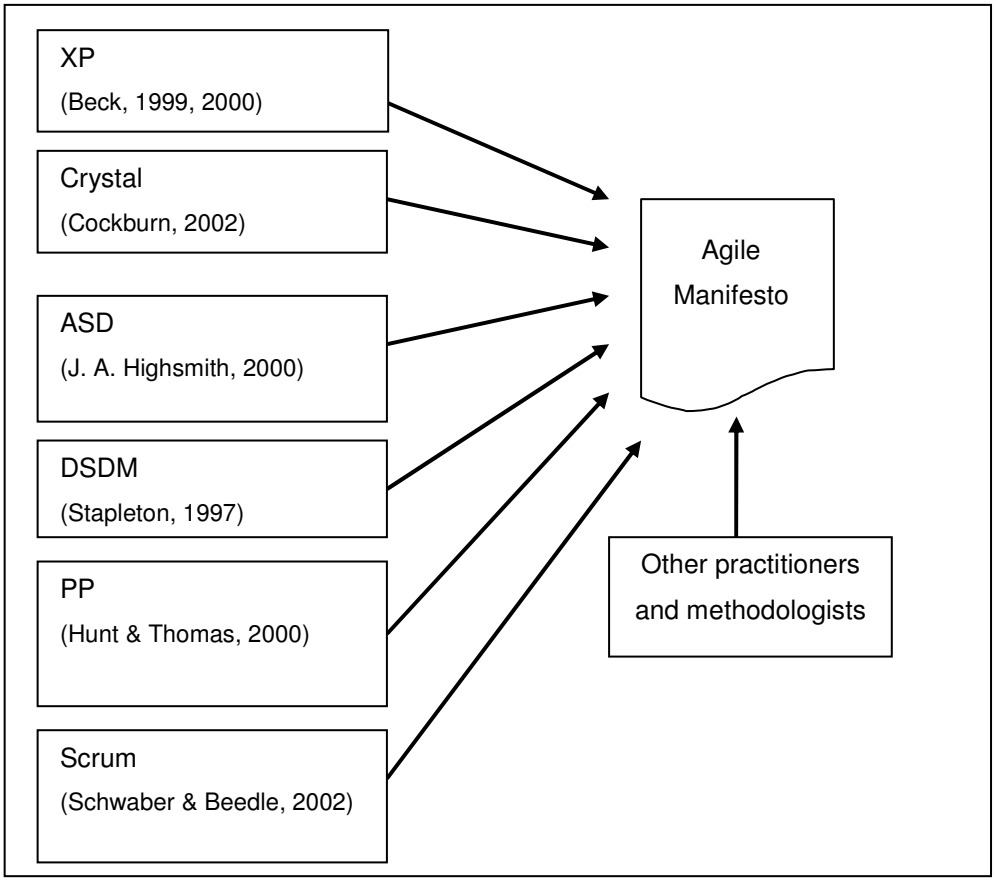

Figure 2: Contributors to the agile manifesto adapted from Abrahamsson et al. (2003)

The evolution of agile methods shows that they are a set of system development methodologies based on practitioner experience and influenced by existing ideas and techniques from the information systems field.

Classification of Agile Methods

There are few attempts to classify agile methods within the body of systems development methodologies. Two are described here. The first is a framework that provides a mechanism for classifying all known methodologies within a hierarchy. The

second takes a different approach by creating a framework for evaluating the quality of agility against which projects and organisations can be measured.

In 2001 Iivari, Hirschheim and Klein published a four-tier framework capable of classifying all known information systems development methodologies (ISDM). They developed this framework in order to organise the knowledge in the many systems development methodologies published up to that time, to clarify the relationships between the different methodologies, and to show how different methodologies belong to different paradigms and have different approaches. Their framework also clarifies which methodologies share development techniques.

The framework divides information systems methodologies into four paradigms. A paradigm is a " fundamental set of assumptions adopted by a professional community that allows its members to share similar perceptions and engage in commonly shared practices" (Hirschheim & Klein, 1989, p. 1201). Each paradigm is associated with a relative degree of order or conflict, objectivism or subjectivism. The paradigms are: functionalism (objective-order), social relativism (subjective-order), radical structuralism (objective-conflict) and neo-humanism (subjective-conflict).

Information system development approaches (ISDA) are the set of related features, principles and fundamental concepts shared by a class of specific ISDM (Iivari, Hirschheim, & Klein, 1999). In 2004 the framework was published describing thirteen approaches (Iivari, Hirschheim, & Klein, 2004). Iivari, Hirschheim and Klein (2004) inserted agile methods into the framework as a separate approach belonging to the functionalist paradigm. This paradigm views methods, people, hardware, software, rules (organisational procedures) as formal, objective entities and assumes that systems development is achieved using formal concepts, planned intervention, and rationalistic tools (Hirschheim & Klein, 1989).

The model published in 2004 (Iivari et al., 2004) showed the four paradigms, thirteen approaches, and a number of methodologies and techniques to illustrate how the model is instantiated.

Using this framework methodologies are classified according to the paradigm and approach to which they belong. The whole classification scheme is dynamic because new techniques, methodologies, approaches and paradigms can be inserted into the model using a semi-formal specification based on the object-oriented concepts of single inheritance, multiple inheritance and instances.

New information systems development approaches are inserted into the model leading to a generalisation (in the object-oriented sense) of the existing approaches. A new methodology is assimilated into the model either by associating it with an existing approach or by redefining the classification structure to insert a new approach to which the methodology can then be assimilated. In some cases a new methodology may be inserted and inherit from more than one parent approach (the multiple inheritance case). An algorithm for the insertion or assimilation is provided but it requires some 'expert' human judgment and is not amenable to automation. The authors describe ten different cases of assimilation and accommodation.

There are two weaknesses of the framework. The first is that agile methods are shown as a completely distinct approach when there is clearly a high degree of overlap between the object-oriented approaches and the agile approaches. For example when comparing the goals, beliefs and principles of the object-oriented approach as determined by Iivari et al. (2001) with those of the agile methods as determined in this thesis (Chapter 4) there are a number of common areas. Both have the goals that "the products meet user requirements", "user requests to modify the system are responded to in a timely fashion", "changes in standards and technology are kept up", and "the project team feels motivated and successful" and both share the principle of "iterative and incremental development" (Iivari et al., 2001, p. 193). This overlap or commonality between approaches is not shown in the model provided by Ivarii et al. (2004). Another criticism is that the definition of agile methods is based on a single analysis of agile methods, that carried out by Abrahamsson et al. (2003). Their comparative analysis is reviewed in Chapter 4.

A different way of classifying the agile methods was proposed by Conboy and Fitzgerald (2004a, 2004b). They classified agile methods by first defining what agility means in information systems development because there is no formal definition of

'agility' in information systems and "no consensus as to what constitutes an agile method, either in academia or in industry" (Conboy & Fitzgerald, 2004a, p. 113). Without a definition of agility any author can state that there method is an agile method. The agile manifesto (AgileAlliance, 2001) is not a suitable set of criteria for defining agility. To be useful as an accurate indicator of agility all of the manifesto principles would need to be met by any method that called itself agile. However each of the commonly acknowledged agile methods fulfils some subset but not all of the principles of the manifesto (Visconti & Cook, 2004), making it, I believe, a poor tool with which to define or assess agility. Conboy and Fitzgerald (2004a) also believed that in order to use an agile method a team should be capable of agility. Based on the literature on agility across different disciplines they developed a conceptual framework for assessing the agility of projects. They noted that the concept of agility arose in mainstream business literature in 1991 but has its origins in earlier business concepts of leanness and flexibility reaching back to the 1950s. They defined agility as, "the continual readiness of an entity to rapidly and inherently, proactively or reactively, embrace change, through high quality, simplistic, economical components and relationships with its environment" (Conboy & Fitzgerald, 2004a, p. 110). Secondly they described a conceptual framework for agility. The authors identified change as a main reason why organisations become agile and they isolated the sources of change to customer, competition, technology, and social factors. Other categories in their framework were the agility strategy, agility capabilities and agility providers. The framework can be used to assess the ability of a team to be agile and if the team is 'agile' then the uptake of an agile method would be worthwhile. There is no actual assessment of the framework against real organisations, projects or teams and the framework assesses projects rather than methods. The assumption in Conboy and Fitzgerald's (2004a, 2004b) study is that a team must be capable of being agile before it should use an agile method. Kahkonen and Abrahamsson (2003) agree with this assumption and apply the idea to whole organisations. I believe this assumption may be incorrect because development teams may adopt an agile method in order to become more agile and capable of adapting to change (Beck, 2000; J. A. Highsmith, 2000; Reifer, 2002a).

Agile methods are classified as a distinct set of methodologies (B. Boehm & Turner, 2003; Iivari et al., 2004; Pressman, 2005; Reifer, 2002a) but they have similarities with

existing object-oriented methods. Classifications within the group of agile methods are now discussed.

Comparative Studies of Agile Methods

A comprehensive comparison of each of the known agile methods was carried out by Abrahamsson, Salo, Ronkainen and Warsta (2002) which integrated the knowledge and terminology in this area. A discussion of the strengths and weaknesses of their analysis is given in Chapter 4 when frameworks are discussed. They compared ten methods which could be considered to belong to the set of agile methods (see Table 4) and concluded the following. Agile methods take a people-centric, developers view of software development. Agile methods are designed for use in small teams of 10 or less developers. They are created from existing software development methods used in novel ways, and are not likely to be suitable for all types of project. Scalability problems are likely to constrain the adoption of agile methods into large organisations. Abrahamsson et al. (2002) reported that there were few empirically validated studies to support the claims that these methods are "effective and suitable for many situations and environments" (ibid, p. 99) as most of the published evidence consists of success stories from practitioners. Finally they called for selection models for practitioners so that they could apply the most suitable method for their needs.

Building on the previous study Abrahamsson, Warsta, Siponen and Ronkainen (2003) investigated and described nine agile methods (see Table 4) with respect to; software development life-cycle coverage, support for project management, whether the method offers abstract principles or concrete guidance, whether the method is universally applicable or situation appropriate, and they tallied the empirical evidence supporting each of the methods. They concluded that life-cycle coverage is not comprehensive because different methods cover different life-cycle phases, all of the agile methods are lacking in project management techniques and, apart from XP and PP, the methods do not offer concrete guidance or explicit techniques only philosophy and values. They found that two of the methods claim to be universally applicable (FDD and Crystal methods) but the others claim to be 'situation specific' or tailorable to different projects. However, these tailorable methods do not describe techniques for identifying and tailoring the method to the project. The authors found that there was little rigorous

evidence to support the efficacy of agile methods as only three of the nine methods studied were supported by empirical studies.

Table 4: Comparative studies of agile methods

	Method	Reviewed by Highsmith 2002	Reviewed by Abrahamsson et al. 2002	Reviewed by Abrahamsson et al. 2003
AM	Agile modeling		+	+
ASD	Adaptive Software Development	+	+	+
Crystal	Crystal methods	+	+	+
DSDM	Dynamic Systems Development Method	+	+	+
dX (RUP)	Rational Unified Process		+	
FDD	Feature Driven Development	+	+	+
ISD	Internet Speed Development			+
LD	Lean Development	+		
OSS	Open Source Software Development		+	
PP	Pragmatic Programming		+	+
Scrum	Scrum	+	+	+
XP	Extreme Programming	+	+	+

I have found that there is little comparative analysis of agile methods; the main work is carried out by Abrahamsson's group (2002, 2003). They found a need for empirical evidence both to support the efficacy of agile methods and to illustrate when they are appropriate or how they should be tailored to fit projects.

Method Combinations

The agile methods have been combined in different ways (see Table 5). With the exception of Visconti and Cook's (2004) study, each of the combinations are proposed by a method author or the combination is of an agile method with a commercial non-agile method (i.e. RUP).

Visconti and Cook (2004) compared XP and Scrum using a framework based on the agile manifesto (AgileAlliance, 2001). These methods were found to meet most, but not all, of the criteria in the manifesto. This led the authors to recommend combining the two methods into a new XP/Scrum method and to add an iteration review activity. This review activity was to provide a feedback mechanism so that the agile method process could be improved in future use. Both XP and Scrum lack such a mechanism. The reasons for combining the two methods were that certain practices used in Scrum

are lacking in XP, and vice versa, and combining the methods creates an improved method. Scrum lacks customer contact and XP provides this by having a customer-on-site for constant consultation and participation. XP lacks a champion role to provide the correct working environment which is provided by the Scrum master in Scrum, and the techniques for testing, coding, and designing are explicitly described in XP but are not addressed in Scrum. However this type of amalgamation assumes that the agile manifesto is a solid basis on which to base the design of a method, and there is no evidence to support this assumption.

Table 5: Proposed method combinations

Combination	Source
XP and Scrum	XP@Scrum ("XP @ Scrum"), Mar and Schwaber (2002), Schwaber & Beedle (2002), Visconti & Cook (2004)
XP and Crystal methods	Cockburn (2002)
XP and ASD	Highsmith (2000)
XP and RUP	Pollice (2001)
XP and DSDM	Simmonds and Fazackerley (2003)
RUP and Scrum	Krebs (2005)

There are further practical reasons for combining methods. Abrahamsson et al. (2003) showed that the XP lacks support for project management, Crystal methods lack detailed advice on the appropriate techniques to use, and Scrum, which is designed for managing iterative and incremental projects, lacks specific practices for the design, code and test portions of the lifecycle. In addition ASD and DSDM are frameworks (J. A. Highsmith, 2000; Stapleton, 1997) offering a philosophy and guidance rather than explicit techniques. Recommendations to combine methods or use techniques from one method in another method have come from a need to address these weaknesses. Only XP offers concrete guidance over their whole lifecycle (Abrahamsson et al., 2003). This explains why XP is the method most often proposed in combination with other agile methods. XP is the most well-studied and frequently used method (Abrahamsson et al., 2003; R. Charette, 2001) and it provides well-described techniques for most of the agile values and philosophies of the agile manifesto. PP is possibly not used in combinations because it is not a method or a process, but a set of programming

practices (Abrahamsson et al., 2002). The amalgamation of XP and DSDM was proposed to improve the scalability of XP and give it greater corporate appeal (Simmonds & Fazackerley, 2003).

Method combination is recommended in some cases by the authors of the methods and there is a single study to show that combination can strengthen weak areas of a method (Visconti & Cook, 2004), but empirical studies of actual projects where this occurs and the benefits of such combinations have not been established.

Agile Methods and Process Improvement

Process improvement in software engineering is based on software process improvement tools such as ISO 9000 and the Software Engineering Institute's Capability Maturity Model (CMM) (Pressman, 2005). Organisations currently using an agile method who want to meet these standards, or organisations currently CMM or ISO 9000 certified, who want to use an agile method need to know if the combination is possible and useful.

SW-CMM is a model against which organisational software engineering capability is assessed. Developed by the Software Engineering Institute at Carnegie Mellon University in the USA it has been widely adopted in the software engineering community. CMM is a 5-level model which describes good software engineering and management practices and prescribes a mechanism for systematic process improvement. Each level of the CMM is made up of Key Process Areas (KPA) or goals that must be met before an organisation is rated at that level.

ISO 9000 is a quality engineering standard and ISO 9001:2000 is the standard for software engineering. These standards are set by the International Standards Organisation and certification is assessed by an external registration body. ISO 9001:2000 has 20 requirements that must be met, for example, management responsibility, quality system, process control, inspection and testing.

"Most leaders in the field agree that XP and SW-CMM are philosophically compatible" (Reifer, 2003, p. 15) and Paulk (2001), a co-author of the CMM, was the first to show

how XP can be used by organisations who aim to meet SW-CMM goals. Although there are a number of areas where XP cannot meet the KPA of the CMM there are other areas where XP techniques can be adopted successfully. XP is able to satisfy each of the KPA at level two of the CMM except software subcontract management. At level three, training programs and integrated software management cannot be achieved, and beyond level three XP addresses few KPA. To achieve higher levels XP must be enhanced with additional practices. XP focuses on technical work and CMM on management issues, but because the two are both rigorous processes they could be used together in a complementary way, according to Paulk (2001). For example XP goals such as good communication and simplicity are also fundamental to CMM organisations. Paulk's recommendations are supported by Bos and Vriens (2004) who describe how XP and Scrum have been used in an organisation to meet CMM level 2 requirements.

Evidence for the compatibility of agile methods and process improvement was found by Reifer (2002a, 2002b) in a study reporting the CMM level of business organisations using agile methods. He found that of 31 projects in 14 firms using agile methods, 23 projects were rated at CMM level 2, and 11 were at level 3 or above. However the respondents self-ranked themselves as to agility and it was found that they did not use specific agile methods but rather any practices which they perceived to be agile. Although this makes the results difficult to compare between organisations (who were possibly all using different agile techniques) it does show that there are instances where organisations using agile techniques were able to achieve a CMM ranking. Problems have been reported when using agile methods while trying to achieve a CMM rating. Incompatibilities in documentation and testing practices between the CMM framework and the agile method were the main problems (Reifer, 2003).

Additional practices are needed when using XP within a CMM framework. Practices such as additional documentation, a requirement engineering phase, and a modified planning mechanism to allow for multiple customer representatives may be necessary to achieve a CMM ranking (Bos & Vriens, 2004; Martinsson, 2003; J. Nawrocki, Jasinski, Walter, & Wojciechowski, 2002a).

Similar additions are needed when combining XP with ISO standardisation. Morkel Theunissen, Kourie and Watson (2003) showed how XP can be used in a standards certified organisation without loosing either the benefits of certification or the benefits of the XP process. They discussed the compatibility of XP and ISO/IEC 12204:1995 which is the most relevant standard in regard to software development. The mechanism was to tailor the ISO standard to a given project in consultation with stakeholders, then to treat the detailed conformance to the standard as a system requirement specification for the XP process. These authors also describe in detail exactly how specific XP practices and techniques meet the various sub clauses of the 12204:1995 standard. Studies undertaken to use XP within ISO 9000 certified organisations report that modifications are needed to achieve this including the need to make documentation, process monitoring, and audit trail requirements for certification, an output of the XP process (J. Nawrocki et al., 2002a; J. Nawrocki, Jasinski, Walter, & Wojciechowski, 2002b; Wright, 2003).

Analytical comparisons and field reports show that, in theory, agile methods, primarily XP, are compatible with CMM and ISO certification. However, in practice the XP must be enhanced with additional techniques to be fully compatible.

Adoption of Agile Methods

From 2001 to 2005 at least fifty articles have been published in books and journals that describe adoption of agile methods. These articles are published in the proceedings of annual conferences in North America and Europe that are dedicated to exploring agile methods (e.g., "Agile Development Conference", 2005; e.g., "Agile Open", "International Conference on eXtreme Programming and Agile Processes in Software Engineering", 2005, "XP @ Scrum", "XP Agile Universe", 2005), and in other forums such as those dedicated to computer science education (e.g. SIGCSE[4], ITiCSE[5] and FIE[6]). However most articles on adoption are experience reports which “take the form of anecdotal success stories or lessons-learned from organizations that have adopted XP for a project” (Layman, Williams, & Cunninghan, 2004, p. 1). These reports often lack

[4] Association of Computing Machinery’s Special Interest Group on Computer Science Education

[5] Innovation and Technology in Computer Science Education

[6] Frontiers in Education Conference

qualitative or quantitative measurement of stated benefits, a research goal, control groups or stated findings (Ambu & Gianneschi, 2003; Bedoll, 2003; Broza, 2004; Fuqua & Hammer, 2003; Howard, 2003; Rising & Janoff, 2000; Spayd, 2003; Vanhanen, Jartti, & Kahkonen, 2003). Empirical evidence for the efficacy, benefits, problems and most appropriate project environment for agile methods is scarce.

The bulk of adoption studies describe the adoption of XP. "XP is the most commonly used agile method today because it is the best documented and thus the easiest to implement" (Lindvall, Muthig, Dagnino, Wallin, Stupperich, Kiefer, May, & Kahkonen, 2004, p. 28). This is supported by Charette (2001) who surveyed the frequency of use of a number of agile methods. He found from a survey of 200 international organisations, that of the organisations using agile methods, XP at 38% was the most popular. This was followed by feature-driven development at 23%. Scrum was the least adopted at 3%. Of the non-agile methods RUP had the largest uptake at 51%. A number of respondents indicated that they were using their own brand of agile method rather than any published method. This could indicate that they are using any development method from *ad hoc* development to a formalised in-house method which makes the survey results tenuous. In addition the survey is not based on a large enough sample to draw firm conclusions because the proportion of respondents using agile methods was not reported. Another survey carried out on-line from Australia ("Agile methodologies: Survey results", 2003) with 131 responses showed similar results with Extreme Programming (59%) the most commonly adopted method followed by Scrum. Both of these on-line surveys however can only act as indicators because the respondents were self-selected.

To address the problem of assessing whether projects are using the agile method XP, ad hoc development with no documentation, or some intermediary method, Nawrocki, Bartosz and Wojciechowski (2001) developed a four-level maturity model. Their model is based on the KPAs of the CMMI[7]. Level 1 indicates the project performance of an organisation is not compliant with XP at all, level 2 is at an initial level, level 3 is advanced and level 4 is mature project performance. XP techniques are assigned to

[7] CMMI – Capability Maturity Model Integration

each level based on the author's assessment of their match to the relevant CMMI KPA. Their maturity model is as yet untested.

Among the agile methods, XP is the most popular in practice and there is now a documented method for assessing the maturity of its use. There are two main areas where studies of agile methods are carried out; the software development industry, and software engineering education and there are now a large number of adoption reports from these two areas.

Adoption in Education

Agile methods have been used in software engineering courses since about 2001 and I have found that most reports (published in books and journals) are about adoption of XP. In some cases the whole of XP is adopted and in other cases individual techniques are adopted and experiments are carried out to study the efficacy of these techniques. In the cases of whole adoption the method is frequently tailored (i.e. some techniques are not used or are changed) in order to accommodate the educational environment of 'pretend' customers, fragmented teaching blocks and the need for assessment. The excluded practices include, metaphor development, customer-on-site, and 40 hour weeks (Johnson & Caristi, 2003; R Mugridge, MacDonald, Roop, & Tempero, 2003b; Noble, Marshell, Marshall, & Biddle, 2004), automated testing and user stories (McKinney, Froeseth, Robertson, Denton, & Ensminger, 2004), and the planning game (Johnson & Caristi, 2003). XP has been trialled mainly in more senior student groups with a consensus that the whole method is too large and complex to adopt at once and students need to be trained in some of the more complex techniques such as unit testing and pair programming before they attempt to use all of the techniques together (Astrachan, Duvall, & Wallingford, 2003; Johnson & Caristi, 2003; R Mugridge et al., 2003b). In addition some level of experience in programming is needed before learning to use the method (Holcombe, Gheorge, & Macias, 2003; Sanders, 2003). Background knowledge of traditional techniques and methodologies is also considered beneficial before attempting XP because it gives the student a greater appreciation of the purpose of XP and its process (Bunse, Feldmann, & Dorr, 2004; Sanders, 2003). There are positive reports advocating XP in teaching (Becker-Pechau, Breitling, Lippert, & Schmolitzky, 2003; Holcombe et al., 2003; McKinney et al., 2004; Smrtic & Grinstein,

2004) and negative reports advising against using XP for software engineering education (Bunse et al., 2004; Sanders, 2003; Schneider & Johnston, 2003) and mixed reports of both benefits and problems (Melnik & Maurer, 2005; Noble et al., 2004). The articles on these topics are primarily experience reports or discussion papers. Conclusions are usually drawn from a single-lecturer perspective and lack control groups against which the results can be compared. One rigorous survey-based study by Melnik and Maurer (2005) found that student perceptions of XP as a whole and of the XP practices of pair programming, test driven development and planning game, were positive and statistically significant across all experience levels and ages of computer science students. They concluded that working in an agile team promotes the development of skills in communication, commitment, cooperation and adaptability.

Adoption of XP in an education setting depends on the goals of the course offered. If the goal is to teach students how to deal with large systems while meeting the criteria of a body of engineering knowledge then XP is not a good choice as it cannot satisfy many of the criteria (Schneider & Johnston, 2003). However if the goal is to teach students how to work well in small teams, to improve their testing capabilities, and deliver functioning software in a short time frame (about 12 weeks for many of the capstone courses in these reports) then these studies indicate XP is a useful choice.

Adoption in Industry

Adoption studies of agile methods in organisations show a generally positive improvement in quality and productivity and a positive response from method users, although results from these studies are usually based on small numbers of participants.

An investigation of the agile practices used by early adopters, the costs and benefits of using agile practices and the type of industries using agile practices was carried out by Reifer (2002a, 2002b). He found that early adopters are using whatever techniques they perceived as agile rather than a published methodology. Survey respondents indicated that projects are small (10 participants or less) pilot projects and most were web-based quick-to-market applications. His study involved 31 projects, 14 that claimed to use agile methods. Cost and benefits were based on a small number of measurements taken in some projects and by qualitative assessment. The measurements showed productivity

improvements of 15 to 23% and cost reduction of 5 to 7%. Qualitative assessment consisted of stakeholder questionnaires in which all "argued passionately for continued use of agile methods" (Reifer, 2002b, p. 18). Weaknesses of the study included the small sample size (14 of 29 respondents used agile methods), and the self-selection of respondents. Respondents were all working on small pilot projects in low-risk environments so the findings may not scale to larger projects or higher-risk projects (Reifer, 2002a). The results may also have suffered from the early-adopter effect. An early adopter may be more positive about a new process just because it is new (Lytinnen & Rose, 2003b).

Two case studies of XP adoption assessed using an XP Effectiveness Framework (discussed fully in chapter 4) found that "over time XP leads to increased productivity and quality" (Layman et al., 2004; Williams, Layman, Krebs, & Anton, 2004c). In both cases pre-release quality of software (case 1: 65% increase, case 2: a two-fold increase), post release quality of software (case 1: 30% increase, case 2: 4-fold increase) and programmer productivity increased (case 1: 50% increase, case 2: qualitative measure showing a positive response).

Beck (2000), the XP author, said that XP is unproven and probably not suitable for mission-critical projects, but there are reports of XP use in these environments. Paige, Chivers, McDermid and Stephenson (2005) advise on the use of XP in high integrity systems such as avionics, marine, energy and defense domains where system criticality is high (i.e. at the level of loss-of-life when a system fails). They report the XP practices of planning game, small releases, refactoring, metaphor, pair programming, on-site customer and 40 hour week are likely to be only partially applicable due to the constraints of developing systems in these environments where process and quality certification is required. The problem with 40 hour week and customer-on-site was confirmed by Wood and Kleb (2003) who reported conflicts with the existing processes when using XP to develop a software test bed for NASA. They concluded that XP is suitable for mission critical software development because of the large number of practices they were able to use successfully in that environment.

Four mission-critical projects using XP were studied by Drobka, Noftz and Raghu (2004). They found the practices were followed 80% of the time, a customer proxy was

necessary as there were too many potential customers for the project, the developers' enjoyment of XP (85%) was high, and compared to traditional development, both productivity (KAELOC[8]/total staff effort) and test coverage was improved (above 73% on all projects).

There are numerous experience reports of XP, but reports on the other agile methods are scant. An improvement in quality and productivity with XP is reported in a small number of cases, and further cases report XP can be used successfully in mission-critical projects, but not without adjustments to the method.

Scalability of Agile Methods

Scalability of a method is its capacity to be effective in situations beyond its specified environment. Agile methods are said to be suitable for small collocated teams of 2-10 developers (Beck, 1999; Stapleton, 1997) and small projects or projects that can be broken down effectively into independent sub-projects suitable for small teams (Schwaber & Beedle, 2002).

Reifer, Maurer and Erdogmus (2003) identified the scalability issues as the need for architecture definition, testing large interdependent systems, selecting the on-site customer from a large customer base, enabling communication between many small teams and geographically distributed teams, accommodating existing legacy systems and existing components, and the need for requirements engineering.

With large teams and geographically distributed teams XP can be successful (Grossman, Bergin, Leip, Merrit, & Gotel, 2004; Schalliol, 2003; Woit, 2005), but the method must be tailored to the project environment. Corporate culture, the interface of the team with the existing organisation, its processes and structure must be accommodated by tailoring the method. Tailoring involves either dropping or adjusting practices, or adding extra practices and support tools (Lindvall et al., 2004). For example a requirements management process was added to Woit's (2005) project in order to manage the requirements dependencies in a large complex project. Woit was using XP to produce a complex collaboration system tool with high security

[8] Thousands of assembler-equivalent lines of code

requirements. Support tools to enable effective communication, collaborative coding, project awareness, and coordination within a non-collocated XP team have also been developed and used (Maurer, 2002; Reeves & Zhu, 2004; Schummer & Schummer, 2001).

Schalliol (2003) described two main problems encountered on a 50-person XP team consisting of developers, quality assurance testers and analysts. Achieving a holistic picture of the application for the use of the whole team could not be achieved using the system metaphor practice, and story cards were problematic when dividing up the project appropriately. Another problem was selecting appropriate customer representatives from a large and diverse customer base with the expectation that the customer will understand their role and participate appropriately. Grossman et al. (2004) reported difficulties in getting the customer to set release dates and carry out acceptance testing, and found a culture clash with the existing governance process of the organisation for another large XP project.

Geographically distributed teams are another scalability issue in using agile methods but, apart from anecdotal reports, there are no research findings in this area. Domino, Hevner and Collins (2002) propose an experimental study of the differences between face-to-face and global virtual software development environments but results are not yet available.

Agile methods and XP in particular, have a particular target environment; small teams, collocated programmers; and projects that can be addressed by teams of this size. When these conditions are not met then the method is adjusted. This is supported by reports of problems, method tailoring, and the addition of practices to XP so that it can be used in large projects and geographically distributed teams. Research beyond single case reports is not yet available.

Individual Agile Method Techniques

XP is the method of choice for studies of individual techniques. There are research streams on pair programming, customer-on-site, refactoring, test-driven development,

communication and teamwork and the effect of organisational culture on the adoption of XP. I have also included a brief discussion of the XP metaphor practice in this section.

Pair Programming

Pair programming is a technique where two programmers sit together at one computer to write code. The 'driver' enters the code and takes a tactical role, the 'navigator' takes a strategic role critiquing the code as it is written (Aiken, 2004). Beck (2000) described the technique as a way of increasing communication within a project team, improving the quality of code, supporting shared knowledge about the developed code as pairs switch partners, and training and sharing knowledge about programming standards when experienced programmers pair with less experienced programmers.

Pair programming is studied in educational and industrial settings. The first empirical study was carried out in 1998 (Hulkko & Abrahamsson, 2005) and to date reports are inconclusive as to the improved quality of software developed and the improved productivity of programmers using pair programming. Muller (2004) found 75% (15 subjects) of programmers thought it was more fun than single programming.

When calculated using system dynamics simulation pair programming gives a 4.4% decrease in development effort (Kuppuswami, Vivekanandan, Ramaswamy, & Rodrigues, 2003). When calculated using net present value ratios pair programming provides a gain in economic efficiency under conditions of strong market pressure, as long as the defect rate is reduced and the pairs work faster than single programmers. This is true even though pair programming doubles the personnel cost of development (Muller & Padberg, 2003). In a controlled experiment when code quality produced during pair programming was compared with that of single programmers undergoing code reviews, Muller (2004) found that pair programming does not produce more reliable code than code reviews. Reviews produced the same code quality as pair programming with nearly the same cost, but pairs were found to be more than two times faster than singletons using code reviews. Small sample size is a problem with this result.

Two academically equivalent groups were compared (using and not using pair programming) by Williams and Kessler (2001). The pair programming students passed

15% more of the provided test cases than non-pairs (statistically significant to $p<0.01$) and qualitative responses showed a high level of positive feeling about the technique. Williams and Kessler (2001) concluded that pair programming in student teams leads to improved morale, productivity, learning and code quality. However there are a number of problems in this study; it is not clear that all of the 28 students in the class responded to the questionnaire, it is not clear that all of the questionnaires were confidential, early adopters can be more positive about an experience, and because the participants are students they may want to please the researcher (who was also their lecturer). However supporting evidence for these results is provided by Sanders (2003) who reported that pair programming is favoured by 70% of experienced students (14/20 students) and is problematic for some (6/20 students).

Hulkko and Abrahamsson (2005) reviewed empirical research into the affect of pair programming on code quality in an industry setting. They found that pairing tends to improve quality but warned that "generalizability and significance of the reported findings remains questionable" because the studies either do not define the metric they report, or there is no direct relationship between the metric and what it purports to measure (e.g. shorter code means higher quality code). They summarised the results of existing studies and concluded that pair programming produces:

1. higher productivity than solo developers;
2. code with higher quality (e.g. better readability);
3. code with lower defect rates;
4. greater enjoyment for developers than solo programming

and is;

5. more useful in complex tasks than simple, routine tasks;
6. useful for training a new person;
7. and the cost-benefit ratio of pair programming (quality vs effort) is unknown.

Hulkko & Abrahamsson (2005) also reported on the results of four cases from close-to-industry settings using the controlled case study method of Salo & Abrahamson (2004). They found evidence to support earlier studies that pair programming is most useful for learning and complex tasks, that code produced by pair programming has higher comment ratio than solo code and higher adherence to coding standards. A new finding was that the effort spent on pair programming is highest in the beginning of a project

and in the final iteration. They also found no evidence to support the earlier studies reporting reduced effort, greater productivity, and code with fewer defects. This means that questions about the benefits of pair programming still remain.

User Participation

Customer involvement in systems development is a major subfield of information systems research were it is more commonly called user participation. User participation is important because it is assumed to improve communication and collaboration between the development team members and the customer, and enables the developers to create a system that will satisfy the customer.

Customer-on-site is an XP practice (Beck, 2000). In XP the customer is expected to 'sit with the team full-time' (Beck, 1999 p. 71) whereas the agile manifesto (AgileAlliance, 2001) principles are less stringent and the advice is for 'customer collaboration' and that 'business people and developers must work together daily throughout the project'. These practices support the communication principle of agile methods.

The customer role is not well-defined in either the agile manifesto (AgileAlliance, 2001) or in the literature on participation. The XP author Beck (2000, p. 143) says 'the best customers are those who will actually use the system being developed' indicating the customer should be an end-user. However the customer may be the client (who pays for the system), sponsor (who sees the need for the system), or end-user (who uses the system). The term 'user participation' is commonly used in the literature to describe any stakeholder who takes part in system development alongside the developer team.

Typical reasons given for user participation in development are:

- To enhance the quality of the system being developed by improving the understanding of the solution and avoiding the development of an unacceptable solution (Roberts, Leigh, & Purvis, 2000).
- To provide more complete and accurate requirements (Roberts et al., 2000).
- To avoid unrealistic customer expectations (Hevner, Collins, & Garfield, 2002).
- To increase user acceptance of systems by reducing user resistance to change (Butler & Fitzgerald, 2001).

- To provide an arena for bargaining and conflict resolution about design issues (Roberts et al., 2000).
- To lead to system ownership by users (Roberts et al., 2000).
- To commit users to the system (Roberts et al., 2000).
- To provide users with an understanding of any changes to work processes and the organisation caused by the new system (Roberts et al., 2000).
- To enhance system design and utilisation by incorporating the creative contributions of diverse stakeholders (Doll & Deng, 2001).
- To increase customer satisfaction (McKeen, Guimaraes, & Wetherbe, 1994).

One of the main reasons for using a systems development methodology has traditionally been to support and enhance communication among stakeholders during all of the phases of development (Wasserman et al., 1983). Royce (1987) provided an explicit plan to involve the customer in development when using a traditional systems development lifecycle. He described five steps for guiding a risky development process to success and one of the steps was, "Involve the customer", because the, "insight, judgement, and commitment of the customer can bolster the development effort" (*ibid*, p 337). Royce believed that the "involvement should be formal, in-depth, and continuing" (*ibid*, p 337) and that the customer should be involved at key points in development.

The importance of user participation is reflected in the Participatory Design (Kuhn & Muller, 1993) movement in European countries that involved users participating in design activities or designers participating in the user's workplace.

In a literature review on user participation in information systems development published from 1982-1992 Cavaye (1995) noted that "For many years it was considered axiomatic that user participation has a significant positive influence on the eventual success of the system" (ibid, 1995, p. 311). She found that participation studies had been undertaken since 1959 but the empirical research findings were inconsistent and contradictory. Cavaye (1995) could find no direct causal link between user participation and system success. The relationship is moderated by contingency factors and other variables; the user's perception of their control of the implementation, the user's desired level of participation, and how important and relevant the system is to

the users. Control is determined by the power relationships between developers and users. When developers have 'sanctionary power' over users then this interferes with true communication and mutual agreement in the participation process (Gallivan & Keil, 2003; Markus & Bjorn-Anderson, 1987). XP acknowledges this; one of the basic principles of XP is "open, honest communication"(Beck, 2000, p. 40) between team members and the team and the customer.

Projects may be unsuccessful and yet involve comprehensive user participation (Butler & Fitzgerald, 1997; Gallivan & Keil, 2003) and satisfied customers (Cavaye, 1995), leading to the conclusion that user participation is not sufficient for project success (Cavaye, 1995). If user participation is necessary for systems success has not been definitively answered because there are instances of successful projects with no user participation (Cavaye, 1995). Numerous studies show a positive relationship between the two, other studies do not concur. Cavaye (1995) attributes this to a lack of consistent measurement instruments, poor definition of variables in participation studies, the reliance on quantitative bivariate measures and a lack of attention to important contingencies such as the type of system studied and the development method employed.

However there is still a strong belief among experts that having users (both managers and staff) included in the development team is one of the important ways that stakeholders can participate in development (Roberts et al., 2000). However user participation is not better in all cases, and user satisfaction can depend on many other factors (McKeen et al., 1994). Too much participation can be ineffective or dysfunctional when it involves technical issues as there is a negative correlation between participation and productivity when the user does not possess the unique skills or training needed to participate effectively (Doll & Deng, 2001).

Until the 1980s customers were involved in development as domain experts and to specify requirements in the analysis phase of development. Then during the 1980s and early 1990s they were formally included to carry out specific support tasks such as user acceptance tests and to participate in design meetings. From about the mid-1990s the user moved into the team to participate in a more inclusive way. At this time the development

team became more heterogeneous. A team could consist of programmers, analysts, end-user representatives, graphic designers, database designers, network engineers, interface design experts and other specialists working together to develop a system. This change in team composition is attributed to the changing scope of development brought about by the development of enterprise wide and web-based systems (Trimmer, Collins, Will, & Blanton, 2000). The agile methods, particularly XP, have formalised this close participation of the user within the team structure.

Communication

Communication research in information systems shows that communication is another important factor in developing systems and direct communication links are more effective in promoting successful projects than indirect links. Direct links are face-to-face communication channels and indirect links are channels where the customer and developer interact through intermediaries or customer surrogates (Keil & Carmel, 1995). Lengel and Daft (1988) who developed media richness theory, a hierarchy of media based on the capacity of a medium to convey information, found that face-to-face communication is the richest medium for managers.

The communication process itself must be effective for the user participation to be effective and developers must "create an environment where users feel free to share their concerns no matter how critical or sensitive those judgements may be" (Gallivan & Keil, 2003, p. 38). These authors identified a common theme in the literature on software project team communication; informal, interpersonal communication is most important in achieving successful results.

Research into user participation and communication supports the values of communication, collaboration and customer participation of both XP (Beck, 2000) and the agile manifesto (see page 18). However empirical research on the on-site customer role in XP shows that it is frequently compromised and problematic. This is because of the sustainability of the role on large and complex projects, expectations that the customer will be capable and willing to carry out the designated tasks, and the difficulty in effectively representing a large diverse user-base and other stakeholders (Beck & Fowler, 2001; Drobka et al., 2004; Koskela & Abrahamson, 2004; A. Martin, Biddle, & Noble, 2004; A. Martin, Noble, & Biddle, 2003; Reifer, 2002b; Schalliol, 2003).

CRACK (Collaborative, Authorised, Representative, Committed and Knowledgeable) customer representatives are recommended by Boehm and Turner (2004) and Beck (2000) advises that the role is not easy.

Beck (2000, p. xv) said "XP is a light-weight methodology for small-to-medium-sized teams developing software in the face of rapidly changing requirements". This method was not originally intended for developing large and complex systems and the reported results on problems with the customer role are all for XP projects which have a large number of possible customers and other system stakeholders. I believe this explains the problems with the role and that a large, concerned customer base may be a contraindication for using XP in full.

Test Driven Development

Test driven development (TDD), or test-first development (Beck, 2000) is an XP technique that has been used sporadically for decades (Fraser, Astels, Beck, Boehm, McGregor, Newkirk, & Poole, 2003; George & Williams, 2004). Individual agile methods include various kinds of testing although it is not mentioned explicitly in the agile manifesto (AgileAlliance, 2001). TDD requires that code is developed by first writing a test then writing the code to meet the test. An automated test tool is required to store the tests and indicate if the test passes or fails. The purpose of this style of unit testing is to improve the quality of each section of code as it is written and to reduce the risk of untested or poorly tested code being released due to time pressure. Test-first development is designed to be used in conjunction with the XP practices of refactoring of code and continuous integration. Anecdotally TDD increases system quality and saves development time (Fraser et al., 2003) but empirical case studies are contradictory (George & Williams, 2004; Maximilien & Williams, 2003; Muller & Hagner, 2002). Shortcomings of TDD include a lack of documented design which limits the approach to the capacity of the programmers to keep the design in their minds, the lack of a high level or macro view of the software under development, the difficulty of using TDD for some kinds of code (e.g. Graphical User Interfaces), the difficulty of writing the mock test objects, a reliance on refactoring to reduce code complexity, and considerable programmer skill and determination to write test cases for hard-to-test code and to maintain the test assets. Benefits include improved code comprehension by programmers, up to date test cases, and faster identification of program faults, increased

testability of code due to embedded test parameters, and an automated test suite available for regression testing when code is enhanced or maintained (George & Williams, 2004). Other research into TDD consists of guidance and techniques for carrying out the practice (Nickell & Smith, 2003; Yuan, Holcombe, & Gheorghe, 2003) and a description of tools for supporting TDD acceptance testing (Ibba & Ohlemeyer, 2003; R. Mugridge, MacDonald, & Roop, 2003a).

Refactoring

Refactoring is a process carried out in object-oriented software systems to improve the internal structure of the programming code in a way that does not alter the external behaviour of the code (e.g. an operation or method body may be changed but the operation signature is left unchanged with no or minimal impact on the interactions of the associated class) (Fowler, 1999). Refactoring is the object-oriented version of restructuring first described in 1990 and is considered one of the cornerstones of XP (Mens & Tourwe, 2004). An overview of existing research in this topic was carried out by Mens and Tourwe (2004) who classified the research by refactoring activities, specific techniques used, the software artefacts that are typically refactored, the characteristics that need to be accounted for in building refactoring tools, and the effect of refactoring on the software development process. I could find no specific research into the effect of refactoring on the efficacy or quality of XP products, but there is a large body of research into this practice in the area of maintenance of object-oriented systems.

System Metaphor

System metaphor is an XP practice used to describe the high level architecture of a system. This is a technique which is difficult for people to understand (Wake & Wake, 2003) and it is not clearly explained by the XP author Beck (1999). The practice is usually dropped or substituted for some other more useful practice by practitioners (Elssamadisy, 2003; Murru, Deias, & Mugheddu, 2003; Paige et al., 2005; Schalliol, 2003; Williams et al., 2004c). Other agile methods have techniques for developing an understanding of the high level architecture of the system under development. For example Crystal methods uses use cases (Cockburn, 2002), Adaptive Software Development uses component planning (J. A. Highsmith, 2000) and DSDM uses a

person who is the 'visionary' on the project and both a feasibility and business study (Stapleton, 1997).

The Target Environment for Agile Methods

The preceding discussion shows that there is research into the effectiveness of agile methods, typically assessed by qualitative studies of programmer perceptions, and there is qualitative and quantitative research into the effectiveness of some agile techniques. However, I could find no research that addresses the question of when it is appropriate to use an agile method. The authors of the agile methods state that agile methods are not suitable in all situations or project environments (Cockburn, 2002; Salo & Abrahamson, 2004) and, "Exactly in which environments and under what conditions agile methods works remains unclear" (Lindvall et al., 2004, p. 26). It is important that method users know when to use a particular methodology and they have asked for this information: "We need some better advice on how and when to use methodologies" (Glass, 2004, p. 19).

I have summarised the situations where agile methods are believed to be most effective based on experience reports, the statements of the method authors, and an empirical study by Reifer (2002a, 2002b) (see Table 6). To organise this information I have divided the target environmental conditions for agile methods into organisational, application domain, people, project, and technology factors to form an initial theoretical model of the target environment for agile methods (see Table 7). There are two contradictory factors. Projects with stable requirements are reported as suitable for agile methods, as are projects with unstable requirements. Projects with precedented systems and well-established architecture, and new developments are also reported as suitable for agile methods. These contradictions arise because the method authors have made a statement about their method that is contradicted by empirical research. Reifer (2002a, 2003) found that precedented systems, well-established architectures and projects with stable environments are those where agile methods are adopted most successfully. However his results are based on a small group of early adopters so further research is needed to clarify this contradiction.

This review has shown that there are many agile method adoption reports. Such studies are another possible source of evidence to support the method author's statements about the target environment for agile methods. However, the problem with using adoption studies to investigate the target environment is that they primarily report on the adoption of XP, and each case describes only some of the possible situational factors (e.g., size of company, application domain, technology used, customer base, team size, or complexity of the product). Two exceptions are the reports from structured case studies used to gather project data, but full results from these cases have not yet been published (Visconti & Cook, 2004; Williams, Krebs, Layman, & Anton, 2004b).

Table 6: A summary of the target environment for agile methods by publication

Situations where agile methods are most effective
Source: Rising and Janoff (2000) • Based on Scrum only • Not for large complex team structures
Source: Cockburn (2002) • Based on Crystal clear, XP and Scrum • Two to eight people in one room • Onsite usage experts • Monthly increments • Fully automated regression (unit and function) tests • Experienced developers
Source: Turk and France and Rumpe (2002) • Based on XP, Scrum, Agile Unified process, Agile Modeling • Internet application domains • Significant time-to-market pressure • Cost of upgrading to the next release is minimal • Not suitable for long-lasting, large, complex systems
Source: Wendorff (2002) An organisational culture where: • Social interaction is assumed to be trustful, collaborative, and competent interaction • Social interaction leads to consensus • Social interaction sychronises activities • The workspace is a social venue • Happy people do good work • Action makes a project work • Leaders influence followers
Source: Reifer et al. (2003), Reifer (2002b) • Small, self-organizing, collocated teams • Teams of less than 10 developers

Situations where agile methods are most effective
• One or more on-site customers • Variable-scope applications with unstable or emergent requirements • Face-to-face communication • Precedented (not new) systems • Stable requirements • Well established architectures • Organizational culture where process change is acceptable (before adopting)
Source: Highsmith (2003) • Small team size • High change environment • Time pressured environment • New development
Source: Boehm and Turner (2004) • Small teams • Small projects • High change environments • Qualitative project control • CRACK customers • Developers consist of 30% full time Cockburn level 2 and 3 experts (see Appendix K Application of the analytical framework for detail on these categories) • No level 1B or -1 personnel (personnel capable of carrying out simple tasks only) • Organisational culture where people feel comfortable and empowered by having many degrees of freedom • Non-critical projects
Source: Nerur, Mahapatra and Mangalaraj (2005) • A management style of leadership and collaboration • Project manager acts as facilitator and coordinator • Teams capable of self-organising • Developers are competent, above-average people • CRACK (Competent, Representative, Authorized, Committed, and Knowledgeable) customers • Small teams • High change environments • Where communication is informal • Where the customer is prepared to accept a critical role in development • Organisational structure is flexible, participative and encourages cooperative social action (rather than bureaucratic and formal) • Object-oriented technology • Suitable tool support is available (e.g for TDD, regression testing) • High value products (to the customer)

The situations where agile methods are not applicable are described (Turk, France, & Rumpe, 2002) for XP, Scrum, Agile Unifed Process and Agile Modeling as:

1. Distributed development environments.

2. Development which is structured around a fixed scope contract.
3. Development to provide reusable artifacts (e.g. agile methods do not support generalized solutions such as design frameworks).
4. Large teams.
5. Development of safety-critical software.
6. Development of large, complex software.

These authors concluded, "Empirical data comparing the effectiveness and limitations of agile and non-agile approaches would greatly enhance our understanding of the true benefits and limitations of agile processes" (Turk et al., 2002, p. 46). They say this because their list was taken from literature published by the method authors or experience reports, rather than empirical studies.

Boehm and Turner (2004) brought these ideas together when they distinguished two types of environments at either end of a continuum; those most suitable for agile methods and those most suitable for 'plan-driven' methods such as SW-CMM, ISO, and USA Department of Defence standards. They believe that "agile and plan-driven methods have home grounds where one clearly dominates the other" (B. Boehm & Turner, 2004, p. 148) and developed a map to determine where a project is situated with respect to these home grounds. The map uses project size (number of personnel), criticality (loss due to impact of defects), personnel skill level, dynamism (requirements change per month) and organisation culture (thrives on chaos versus order) to determine the most suitable type of method to use for a particular project. The map is used to create a graph for a project. The shape of the graph indicates if the project should be developed using agile methods or plan-driven methods. Projects with ratings near the centre are more suitable for agile processes. The graph provides only a rough selection guide because the shape is assessed manually and is not quantifiable. I believe that another problem with this map is that it does not take into account the application domain, the size of the user base (which affects the customer role, as discussed in the section: Individual Agile Method Techniques, p. 36), the technology used, the technology developed, or the support tools provided.

Table 7: The target environment for agile methods by category

Organisation factors	Social interaction is trustful Social interaction is collaborative Social interaction is competent Face-to-face communication Communication is informal Organisation enables empowerment of people Management style of leadership and collaboration Organisation is flexible and participative and encourages social interaction
Domain factors	Internet application domains
Technological factors	Automated testing Object-oriented technology
Project factors	High time pressure Unstable requirements Stable requirements Precedented systems Well-established architectures New development Qualitative project control Non-critical project Monthly increments Small teams Collocated teams Project manager acts as a facilitator Product is of high value to the customer
People factors	CRACK customers Experienced developers On-site usage experts Teams capable of self-organising

Glass (2004) discussed the importance of identifying the application domain and then choosing an appropriate systems development methodology. He believes that it is important to have well defined methodologies and that each methodology has a clear domain-of-fit. His solution to this problem was to create taxonomies of available methodologies and of application domains (Glass & Vessey, 1995) and then map the methodologies to the domains. In order to place the agile methods into such taxonomy first a definition of what characterises an agile method is needed followed by a definition of the application domain where they should be used.

In summary this literature review shows that while there is a substantial amount of literature on agile methods empirical evidence based on rigorous research is scarce. Most research is into XP and Scrum and there are a number of other published agile methods with no supporting research. XP is a key method. Almost all research focuses on XP and its techniques, and empirical research confirms the improved productivity and quality of systems developed using XP. A problem with studies into the agile methods remains. There is no clear boundary defining which methods legitimately belong to the set of agile methods. It is not clear from the existing literature and cannot be clear until there is an answer to the question; what is an agile method? To answer this question a set of conditions must be constructed and any new method must be benchmarked against these conditions to determine whether it belongs to the set of agile methods or not. The agile manifesto is inadequate as a benchmark as discussed in the section: Classification of Agile Methods (p. 21).

Another question posed, but not investigated empirically is; what is the target environment for agile methods? The need for this research is stated by Abrahamsson, Warsta, Siponen and Ronkainen (2003): "Empirical works that study the effects of particular methods, their ease of use, costs and possible negative implications for different sizes and lines of business are needed in particular. The empirical work should use both qualitative and quantitative research methods to study these issues."(p. 252).

Research Questions

Two questions arise from this review of the literature of systems development methodologies and agile methods:

1. What is an agile method?
2. For what type of environment are agile methods most suitable?

The second research question is related to the first because the environment of an agile method cannot be studied unless there is first some definition of an agile method.

These questions are important for a number of reasons. A common theme or set of characteristics of agile methods has not been substantially identified. Reifer (2002a, p. 18) stated that it is necessary to 'clearly identify what "agile methods"' means because, after questioning software engineers, he found that there is no common meaning of the

term. A definition is needed so that the method users and those who want to study, evaluate, and compare methods have a common understanding of what an agile method is and which methods legitimately belong to the set of agile methods. Currently when new system development methods claiming to belong to the set of agile methods are published there is no way to show that the claim is legitimate. One method for checking the claim would be to benchmark the new method against the distinct properties of agile methods. However these properties must be determined before making this type of classification. Another outcome of having a definition for 'agile method' is to enable agile methods as a whole to be compared with other classes of methodology as was attempted by Iivari, Hirschheim, and Klein (2001) and discussed in the section: Classification of Agile Methods, page 21.

Few comparative studies of agile methods are available as previously discussed. Currently in order to compare methods and select the most suitable for use in a project it is necessary to read a book on each method. This is a challenge for any project manager who needs to select an appropriate agile method from those currently available (Nerur et al., 2005). Those who want to select a suitable method for their project or who want to study the techniques (e.g. pair programming) of the methods have very few unbiased studies to choose from that define, describe and compare the published agile methods. This thesis aims to provide some of this information by defining and comparing five of the earliest published agile methods in order to answer the first research question; what is an agile method? Research question one is broken down into sub questions (see Table 8). Answering these questions provides an explicit definition of the properties, common characteristics and differences between each agile method, and possible combinations of agile methods. The published problems or weaknesses of the methods are included as it is important to know in what environments agile methods are not suitable as specified by the method author.

Table 8: Research question 1, what is an agile method?

	Secondary questions	Purpose of the question
1.1	What are the properties of individual agile methods?	To explicitly define the properties of individual agile methods.
1.2	What are the properties common to all agile methods?	To determine the characteristics common to all agile methods. To enable accurate classification of new 'agile' methods in the future.
1.3	What are the differences between agile methods?	To enable appropriate selection of an agile method based on different needs.
1.4	What properties are unique to particular agile methods within the set of agile methods?	To determine if agile methods are different to one another or are all a version of one basic method.
1.5	What combinations of agile methods are possible?	To determine how agile methods can be combined to provide a more useful or encompassing method capable of solving a wider range of problems more effectively.
1.6	What are the published problems with agile methods?	To know in what environments agile methods are not suitable according to the method authors.

The second research question: for what type of environment are agile methods most suitable, is broken down into sub-questions (see Table 9) about the method environment, how the method is tailored to accommodate the environment, and what the perceived benefits and problems of agile methods are within an organisation.

Table 9: Research question 2, for what type of environment are agile methods most suitable?

	Secondary questions	Purpose
2.1	How are agile methods used in projects and how does the project environment affect the use of an agile method?	To determine how agile methods are tailored for use. To determine why agile methods are tailored for use. To determine the type of environment in which agile methods are used. To determine if there is a relationship between the project environment and how the method is tailored for use. To support or refute the stated target environment in which agile methods are most suitable
2.2	What are the perceived benefits of agile methods?	To determine any perceived benefits of agile methods To determine if there is any relationship between the method environment and the perceived benefits.
2.3	What are the problems with agile methods?	To determine the perceived weaknesses in the published method To determine if there is any relationship between the method environment and the perceived problems.

It is important to isolate the type of environment best suited to agile methods because method users need to know this information before selecting a method. They also need to know of the problems inherent in the method if it is used in particular environments. Published methods seldom include discussion of the weaknesses of the method so this research aims to provide some evidence of problems or weaknesses with agile methods.

The extent to which a method may be changed or extended (its tailorability) and yet remain useful is important information for method users. Tailoring is a common procedure carried out on methods to suit the environment (Avison & Fitzgerald, 2003d; Goulielmos, 2004). A complete lack of tailoring could indicate that a method is perfect for the environment in which it is used. Some tailoring to accommodate certain environmental situations could indicate weaknesses in the published method or that the method is not useful in certain situations. Tailoring is also needed when a method is used in projects larger in scope or complexity than that for which it was designed, thus tailoring is related to scalability.

The use of a theoretical model is recommended for case study research (Yin, 2003) and the theoretical model on which research question 2 is based is shown in Figure 3.

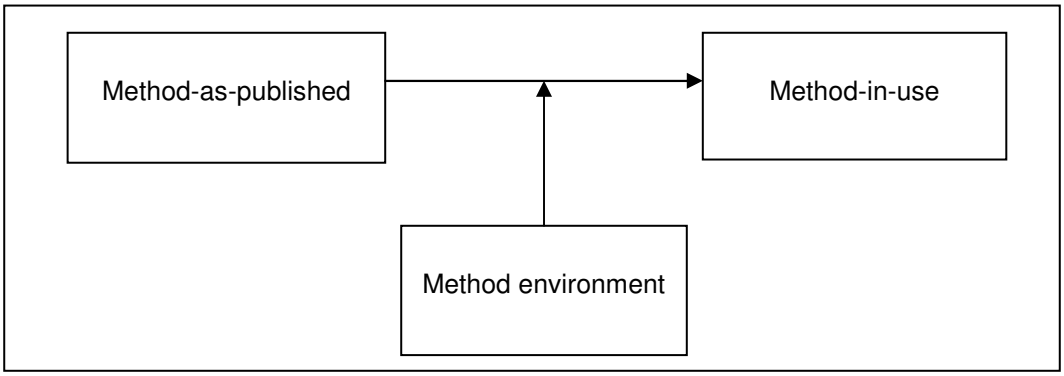

Figure 3: Basic theoretical model for research question two

The model shows that the method as it is used on a real project (method-in-use) differs from the published method (method-as-published) because it is affected by the method environment (the situation) in which it is used. There is an underlying assumption in this model that the method-as-published has a unique target environment (the method environment) and when the method-as-published is used in its target environment then it will not be tailored.

Summary

This review has provided a history of the ideas leading to the development of agile methods, described which methods are agile methods, and summarised the research into agile methods and individual agile method techniques. There are numerous case studies of agile method adoption but conclusive research findings are few. Studies of individual techniques focus on those of XP and show improved quality and productivity in most cases. Based on the findings from this review I have constructed an initial model of the target environment for agile methods. The review has described the research questions and sub questions designed to meet the objectives of this study and emphasised why they are important. The following chapter describes the methods used to address the research questions.

Chapter 3 RESEARCH METHOD

Introduction

There are two research questions in this study. The first: *what is an agile method?* is addressed using an analytical framework to describe the characteristics of five agile methods. Details of the analytical framework and its application are described in chapter 4. The second: *for what type of environment are agile methods most suitable?* is addressed using a multi-case study to investigate the way that agile methods are used and the environment in which they are used. This chapter first describes the overall research design then focuses on the design for research question 2.

Method Overview

The first research question (see Figure 4) is addressed using an analytical framework to assess five agile methods; DSDM, Scrum, XP, ASD and Crystal. The framework characterises and describes them in a format where they can be easily compared. Application of the framework also adds to the theoretical model of the target environment for agile methods described in the Chapter 2 Literature Review.

To address the second research question (see Figure 5) I use a multi-case study of nine software development projects. The case studies are designed to investigate the target environment for agile methods by assessing both the extent of method tailoring on projects, and the project environment. The method follows this logic: for each of the environmental factors in the theoretical target model, if there is a relationship between the extent of method tailoring on the project and the observed project environment factor then the corresponding target environment factor in the theoretical model is valid.

Research Methods in Information Systems Development

Dube and Pare (2003) described four main research methods in information systems; laboratory experiments, field experiments, field studies and case studies. Experimental methods involve manipulation of independent variables in order to measure their effect on dependent variables.

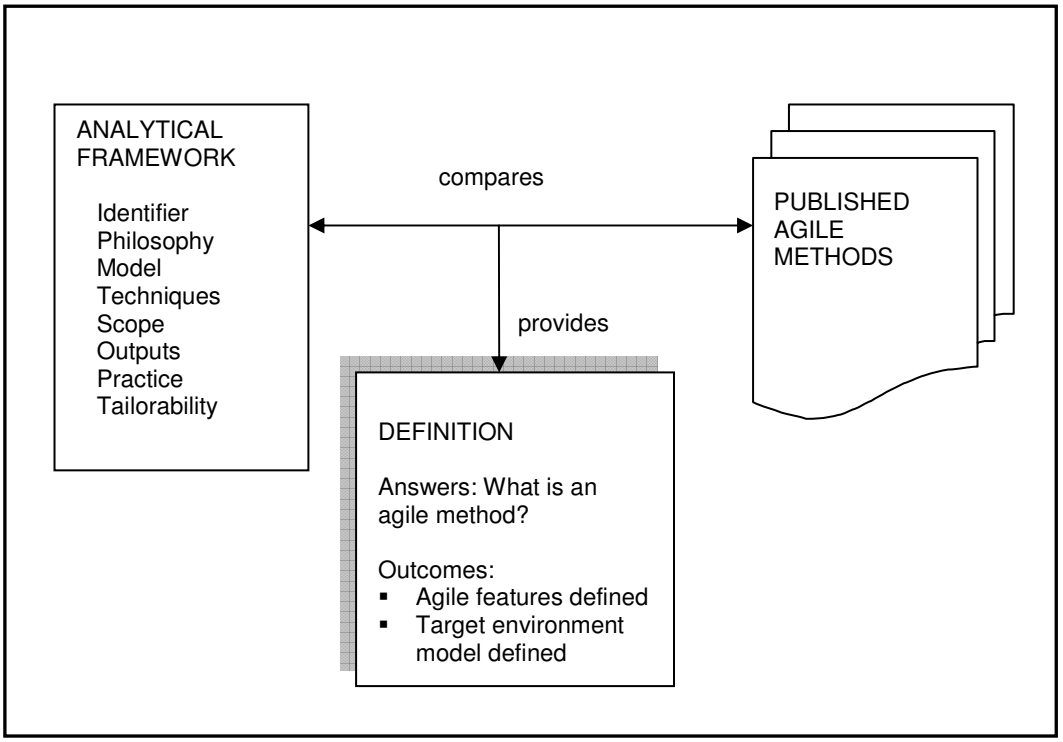

Figure 4: Research design for research question 1: What is an agile method?

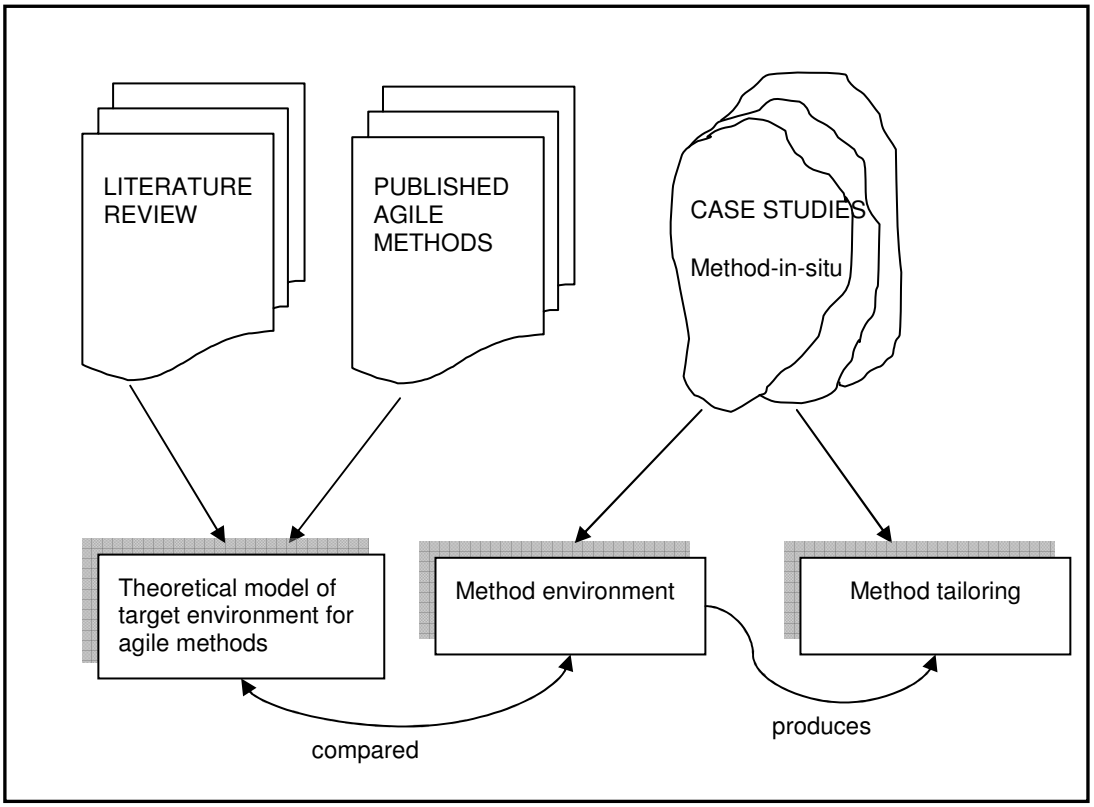

Figure 5: Research design for research question 2: For what type of environment are agile methods most suitable?

This occurs either in a laboratory or in a natural setting (e.g. industrial practice). The distinction between field studies and case studies is less clear because both methods are carried out in natural systems and do not involve control or manipulation of independent variables. The difference is that case studies involve a small number of cases which are studied in depth using a variety of data collection methods whereas field studies involve more sites and usually only one type of data collection method (Boudreau, Gefen, & Straub, 2001).

The Case Study Method

There are two main philosophical approaches to case study research in information systems: the positivist approach and the interpretivist approach (Darke, Shanks, & Broadbent, 1998). Interpretivist research is concerned with understanding phenomena by studying social situations and how they are interpreted by people according to their beliefs and values. The researcher carries out the analysis in a subjective way and aims to gain a deep understanding of a particular situation without trying to either repeat or generalise the findings. The interpretivist case study approach uses research methods such as ethnography (Beynon-Davies, MacKay, & Tudhope, 2000) and dialectical hermeneutics (Butler & Fitzgerald, 1997). Positivist research is concerned with testing theories in order to discover general principles or laws governing the natural and social world, and is based upon the scientific paradigm and collection of empirical data. The researcher remains neutral and objective and the data gathered is qualitative, quantitative or both. An analysis of seven major IS journals between 1990 and 1999 showed that positivist case studies are the main type of case study research (Dube & Pare, 2003).

Case studies are used to describe phenomena, to develop theory, and to test theory (Yin, 2003). Case studies typically combine data collection techniques such as interview, observation, questionnaires, and document and text analysis. Both qualitative methods of data collection and analysis, concerned with words and meanings, and quantitative methods, concerned with numbers and measurement, can be used (Wohlin, Host, & Henningsson, 2003; Yin, 2003).

The positivist case study method is appropriate for my research questions for a number of reasons. Case study research is considered particularly appropriate for the study of information systems development (Darke et al., 1998) and it is also appropriate in software engineering research (Wohlin et al., 2003). Both of these fields are associated with the agile methodologies. Case studies are the 'preferred strategy when "how" or "why" questions are posed' Yin (2003, p. 1). Real-life context, contemporary events and a lack of control of variables by the researcher are also situations where case study research is appropriate (Yin, 2003). This study of systems development projects and the environment of agile development methods meet these criteria. The final reason why a positivist case study was chosen was that a fully quantitative research methodology was not feasible. Random sampling was not possible because locating a representative sample from a larger population of all software development projects could not be achieved. This was due to the small population of software development projects using agile methods within New Zealand and the difficulty in locating such projects.

The positivist nature of this qualitative study was confirmed by comparing it with Fitzgerald and Howcroft's (1998) definitions of research as either 'hard' or 'soft' at each of four levels; the ontological, epistemological, methodological and axiological level. Hard research is realist at the ontological level, and both positivist and objectivist at the epistemological level. This research fits that profile. At the methodological level it is less well aligned with the positivist tradition because both qualitative (e.g. method users experience with pair programming) and quantitative measurements (e.g. exactly which techniques in XP are used) are used. At the axiological level this research does not fall clearly into either the soft or hard "research dichotomies" (Fitzgerald & Howcroft, 1998, p. 10) because I try to achieve both rigor and relevance. Rigor is addressed in the section on validity. Relevance is addressed because this research is important to practitioners using or planning to use an agile method. The importance of determining the target environment for a systems development methodology was discussed in the literature review in Chapter 2.

The research design in this thesis addresses each of the primary criteria for positivist case study research (Dube & Pare, 2003):

1. A positivist perspective is clearly stated in the study.
2. Formal propositions or hypotheses are stated.

3. Qualitative or quantitative measures of constructs or variables are used.
4. The case has the explicit purpose of theory testing.
5. There is a concern with validity and reliability issues as they are used in the natural sciences.

The method followed in this study is that defined by Yin (2003) and Dube and Pare (2003), and illustrated by Lytinnen and Rose (2003b) when describing the impact of Internet computing in systems development organisations.

In summary, this research study is carried out using a positivist case study method because this method is suitable for addressing the research questions and for investigating systems development projects. The case study method I use involves collecting both qualitative and quantitative data that is analysed using both qualitative and quantitative techniques.

Research Questions and Propositions

Research propositions were developed for the sub questions of research question 2 (see Table 10). The propositions link the research questions to the questionnaire and interview. Proposition 2 (there are specific environmental conditions that are suitable for agile methods) is addressed using the theoretical model of the target environment for agile methods.

Unit of Analysis

The unit of analysis in case study research defines the boundaries of the case investigated (Yin, 2003). In this study an individual system development project is the unit of analysis. Each project met the following criteria:

1. The project had a well-defined purpose.
2. The project was carried out using a set of activities that aimed to produce a software application.
3. The project had both a beginning and an end date (estimated if the project was not complete at the time of the study).
4. The project was underway in that at least 1/3 of the estimated project duration was complete.
5. At least one person was working on the project.

6. The project was carried out using either:
    - An agile method.
    - A systems development methodology that was not an agile method (i.e. Rational Unified Process, another named methodology, or a documented in-house methodology).
    - No systems development methodology (i.e. ad hoc development).

Table 10: Research question two and its propositions

Research question 2 **For what type of environment are agile methods suitable?**			
	Secondary questions	**Purpose**	**Propositions**
2.1	How are agile methods used in projects and how does the project environment affect the use of an agile method?	To determine how agile methods are tailored for use. To determine why agile methods are tailored for use. To determine the type of environment in which agile methods are used. To determine if there is a relationship between the project environment and how the method is tailored for use. To support or refute the stated target environment in which agile methods are most suitable.	1. The full method is not used in practice; the method is tailored-for-use. 2. There are specific environmental conditions that are suitable for agile methods. 3. An agile method is not tailored when it is used in its target environment. 4. An agile method is tailored when it is not used in its target environment.
2.2	What are the perceived benefits of agile methods?	To determine any perceived benefits of agile methods To determine if there is any relationship between the method environment and the perceived benefits.	5. Method users perceive benefits from the use of the method which match those of the published method. 6. There are more perceived benefits when environmental conditions suitable for agile methods are present.
2.3	What are the problems with agile methods?	To determine any perceived weaknesses in the published method To determine if there is any relationship between the method environment and the perceived problems.	7. There are problems with some techniques used in agile methods. 8. There are more perceived problems when environmental conditions are not suitable for agile methods.

Validity

Validity is the extent to which a measurement device measures what it is designed to measure. The trustworthiness of empirical research is addressed by designing and

carrying out procedures that limit the threats to validity. There are four tests to establish the quality of positivist case study research, construct validity, internal validity, external validity, and reliability (Yin, 2003).

Construct validity is concerned with ensuring the constructs used in the study actually measure what they are intended to measure. The case study tactics for addressing construct validity are to use triangulation, to establish a chain of evidence and to have key informants review a draft case study report.

Constructs in this study were the factors making up the target environment and the techniques used in each of five agile methods. These constructs were explicitly defined before any data was collected and the questions, and question items, used in the instruments were drawn from existing literature on information systems and organisational culture whenever possible (e.g. domain of application types, categories of technology).

The first tactic for addressing construct validity is triangulation. Triangulation comes in three forms (Mathison, 1988): (a) data triangulation, that is data gathered at different times, locations, and from different people, (b) investigator triangulation, when data is gathered by more than one investigator, and (c) methodological triangulation, when data is gathered using more than one method. This study uses data from two sources, company web site documentation, and interviews and questionnaires completed by project leaders. The interview verifies the section of the questionnaire about use of agile techniques by asking why the particular technique was not always used. Interviews occurred 1 to 2 weeks after the questionnaire was returned. Multiple sources of evidence were not used because "the use of multiple sources of evidence imposes a great burden" on the researcher (Yin, 2003, p. 99) and such evidence was not easy to obtain. I made a compromise between the number of cases and the number of participants involved. In order to have more cases fewer people were asked to be participants. The original research plan had two questionnaires, one for the project leader and another for the team members, but because of the difficulty of getting agreement to participate from all members of a project team as well as the project leader, this idea was abandoned and only the project leader's perceptions and knowledge was drawn upon. Methodological triangulation was used. The

questionnaire gathered qualitative and quantitative data and a follow-up interview gathered qualitative data. Investigator triangulation is not used because I was the only investigator available.

The second tactic for addressing construct validity is to establish a chain of evidence. A chain of evidence from initial research questions, to propositions, instrumentation, data analysis and conclusions is provided in this thesis.

The third tactic for addressing construct validity is to have the case study reviewed by key informants (respondents). Key informants were not asked to comment on the draft case study due to time constraints and because I did not want to impose any further on those who had already provided from one to three hours of their time.

Another type of validity is internal validity. This is concerned with identifying the true relationships between variables and is of concern in explanatory (or causal) case studies. This study is an explanatory case study because it investigates the relationship between the project environment and the use of agile methods. Internal validity is concerned with ensuring that an effect is caused by a particular variable and not some unknown confounding variable. Case study research using qualitative methods is less internally valid in general than purely quantitative methods (Collis & Hussey, 2003). There are four methods for addressing internal validity in explanatory case studies, the use of rival explanations, pattern matching, logic models, and explanation building in case analysis (Yin, 2003). Chapter 6 (Conclusion) is where rival explanations for the outcomes of the research are discussed. Pattern matching is used when the theoretical target environment (the predicted pattern) is compared with the actual environment of the project cases (the empirically based pattern) in Chapter 5 (Results). Logic models and explanation building are not used in this study because they are more suitable when there is a complex chain of events over time or the study is about social policy and social behaviour (Yin, 2003).

External validity is concerned with generalising the results of a study to other cases and the wider population. Case study research does not allow for statistical generalisation but it is appropriate to use analytical generalisation (Yin, 2003). Statistical generalisation is when a statistically determined sample is taken from a whole

population and the results of the study are then generalisable to that population. Analytical generalisation is when a theory is developed and then cases studies are carried out. The results can then be generalised to the stated theory. The theory is tested using both literal replication which means selecting cases because they are predicted to give similar results, and theoretical replication which means cases are selected because they are predicted to provide contrasting results for predictable reasons. In this study the literal replications are those cases using agile methods and the theoretical replications are those that are not using agile methods. This is likened to selecting multiple experiments in scientific research (Yin, 2003). If the cases selected to predict similar results are shown to be contradictory then the initial propositions must be revised and retested with another set of cases. If the cases provide the results as predicted then this provides support for the theory or propositions being tested.

Reliability is concerned with whether a measure provides the same results on different occasions. Improving reliability minimises the errors and bias in a study. In positivist case study research reliability is achieved by using a case study protocol and by keeping an organised record of all case evidence (a database) (Yin, 2003).

The protocol is a set of documented research procedures that, if repeated by another researcher on the same case, would enable them to arrive at the same conclusions. The protocol used in this study was to get all respondents to complete the same questionnaire. Interviews were carried out using a protocol that was semi-structured as described by Wengraf (2003). All interview questions were the same except for those related to the use of individual agile techniques.

The database is a complete set of all data gathered during the case stored in a format suitable for independent inspection by others. A database was kept containing all Internet search results, questionnaires, interview notes, interview recordings, web site results, and any notes or emails taken or received during initial contact or following up from receipt of questionnaires. Notes and emails were recorded when discussing and assessing the suitability of a case for this study.

Case Selection

Cases were selected using a convenience sampling strategy (Dube & Pare, 2003). Participants were identified by an Internet search for organisations carrying out software development in New Zealand who advertise or state (typically in practitioner journals) that they use agile methods, and through personal contacts made at developer conferences, at local IT seminars, and at a specialist developers group. Participants were selected because they met the criteria of working on a project (see Unit of Analysis above) and had first hand experience with the project and an overview of the process used during the whole length of the project. These people I have called project leaders but their role in the workplace included project managers, team leaders, analyst programmers, analysts and others.

In order to follow a replication logic five cases reporting using agile methods were selected. One of these was found to be a mixed method project (agile and RUP). Theoretical replication was achieved by selecting four cases where agile methods were not used. The questionnaire was distributed to all cases. Interviews were carried out on all agile cases and one ad hoc case.

Preparation for Data Gathering

Pre-testing of questionnaires, and the use of pilot case studies are recommended for case study research (Dube & Pare, 2003; Yin, 2003). Cognitive pre-testing was used to ensure the questionnaire was easy to understand and did not contain any inappropriate language. The questionnaire was given to various people including a business analyst, four academics (including one expert on agile methods and one statistician), a software developer, and a person with no knowledge of academia, development methodologies or software development. Their recommended changes were incorporated into the questionnaire.

A pilot case study was used to ensure the questionnaire and the interview were carried out effectively and efficiently and provided appropriate data. The pilot study provided an estimate of the time that the questionnaire would take for respondents and gave me practice in understanding the complexity of the data gathering process (e.g. managing both note taking and interview recording). I also learned that time was needed to check

the completed questionnaire for missing answers and to set up the appropriate questions for the interview. Based on this experience I decided to deliver and receive the questionnaire by post.

Data Collection

There were three data collection methods, a questionnaire, an interview and a web site analysis. The questionnaire was given to project leaders then followed up by an interview to explore the questionnaire findings. The questionnaire assessed the project environment and how the agile method was used within the project. The interview was used to find out why the agile method was tailored for use and what, if any, environmental factors contributed to the tailoring. The web site belonging to the organisation where the development occurred was analysed to determine organisation type and size.

Initial contact with project leaders was made by phone, email or personal contact. The purpose of the study was explained to the project leader and a brief overview sheet (see Appendix A) was emailed out. Once informal consent was given the questionnaire was posted out. The package included a hand written covering note, an information sheet (see Appendix B), a consent form (see Appendix C), the project questionnaire (see Appendix D), and a return envelope. Posting the package allowed the respondent time to consider the consent form, to decide if they wanted to take part in an interview and if they would consent to being audio recorded.

After about a week the respondent was phoned or emailed to ask how they were getting on with the questionnaire. Once the questionnaire was returned, which usually took about one week, I checked to see that all questions were answered and if there were any questions with ambiguous answers or comments such as: 'this does not apply to us'. Any incomplete questions were discussed during the interview to make sure that they were answered if possible. If the respondent was not interviewed (ad hoc respondents were not interviewed) then any unclear answers were reviewed by phone call or email.

Once the questionnaire was complete it was analysed to see which agile method was used and to identify any agile method techniques selected that were not used fully. If

the respondent had agreed to an interview then an interview sheet was prepared and an interview was arranged to suit the respondent. This was either at the development site, at my workplace or at a public place. An off-site location tended to be selected by those who were carrying out contract work for a company who was not their employer. The interview sheet was standard in most questions except for the section on the use of agile methods which were different for each respondent (see Appendix E). The interviews were audio recorded in each case except one, which was carried out by telephone because of distance. Careful notes were taken at each interview and reviewed immediately following the interview to make sure that each answer was recorded accurately.

The description of the organisation was taken from the organisation's web site. The organisation where the development project was physically carried out was the one checked. In some cases the web site could not be used because the name of the organisation had to remain confidential (for contractors) and in another case the organisation had been taken over by another organisation just after the end of the project. If no web site was available the respondent was asked to provide equivalent detail.

Data Analysis

Each individual project was described as to the overall background of the organisation where the project was carried out, the type of project, technologies used, and the experience and training the project team had in using their agile method. The extent of method tailoring was calculated. Each of the criteria in the theoretical model of the target environment for agile methods was investigated in turn using a cross-case analysis. All cases were compared using quantitative non-parametric statistics to investigate the correlation between the project environment and tailoring of the agile method.

Instrumentation

This section describes both the questionnaire and the interview questions, the reasons for each question and any supporting literature for the items in each question.

The questionnaire gathered both quantitative and qualitative data. The questions had three aims:

a) To check that the project was a suitable case for this study.
b) To enable the theoretical model for the target environment of agile methods to be adequately assessed.
c) To determine the extent of tailoring of the agile method.

The questionnaire was divided into 11 sections each one labelled to provide logical groupings of questions and to give some guidance to the respondent as to the purpose of the questions in each section. Sections 1 to 9 had questions to determine if the project matched the unit of analysis requirements, to assess the organisational culture, the technology used, the application domain, project factors and the experience level of the project leader. Section 10 asked about individual techniques belonging to each of five agile methods; XP, Scrum, DSDM, Crystal and ASD. Section 11 asked for the project leader's opinion on the advantages and disadvantages of agile methods.

The purpose and source of each of the questions in the questionnaire and the interview is now described. Questions designed to assess factors in the theoretical model of the target environment for agile methods are marked with an asterisk (*).

Questionnaire Design

Section 1 General factors

This section asks about the organisation in which the physical work for the project takes place (for outsourced projects this is the site where the work is primarily carried out). Follow up interview questions were sometimes used to clarify that this was understood by the respondent.

Question 1: "What type of development does your organisation undertake?

Categories in this question are the most common types of software development carried out in New Zealand as determined from the results of a survey of software development practices (Groves, Nickson, Reeve, Reeves, & Utting, 2000).

Question 2: "Approximately how many staff are involved in software development in this organisation". This question was considered important because projects with many developers in large organisations with formal working practices were more likely to use formalised methodologies (Fitzgerald, 1998).

Question 3: "Approximately how many projects has your organisation completed using an agile method"? This question determined the level of experience of the

organisation in its use of agile methods. If the users are new to agile methods they are more likely to be biased towards showing successful adoption (Lytinnen & Rose, 2003b).

Section 2 Organisational culture

Question 4 – 9*: Agile method authors have stated that agile methods should not be adopted if the culture is not suitable (Beck, 2000; Cockburn, 2002) otherwise the culture may need to be changed for the method to be successfully adopted. The culture should be open, informal and non-hierarchical (Beck, 1999) and one which, "thrives on chaos" (B. Boehm & Turner, 2003, p. 56). Organisational culture is a major research field in business (Hofstede, Neuijen, Daval Ohayv, & Sanders, 1990; Wallace, Hunt, & Richards, 1999). The questions used in this section are adapted from those used by Paparone (2003) and originally developed by Cameron and Quinn (1999).

Section 3 Project description

Question 10 – 11: The name of the project is requested but it is not used to identify the project in the analysis; it is used to focus the respondent on the particular project that makes up the case because it is important that only one project is reported on in the questionnaire. The duration of the project was requested to ensure that the project fitted the unit of analysis specification that the project was underway and at least 1/3 completed. This helped to ensure that the project was viable and the use of the method could be effectively assessed (Williams, Kerbs, & Layman, 2004a).

Question 12*: the nature of this project"? This question is taken from the Extreme Programming Evaluation Framework of Williams, Kerbs and Layman (2004a).

Question 13: "What criteria do you use to determine if the project is completed"? What constitutes the end of a project may differ in each case. In order to compare cases the actual end point of the project must be known and this question helps to clarify any difference between cases in this respect.

Question 14 – 15*: How many… developers..? These questions provide information about the size of the project.

Question 16*: "…how many end-users will use the system"? This question indirectly measures the likely complexity of the project.

Question 17 – 20*: These questions provide information about the domain and technology used in the project. The list is adapted from that of Kroenke (1992), cited in Glass and Vessey (1995), who specialised in characterising types of systems.

Section 4 Project complexity

Question 21*, 22*: These questions provide information about the complexity of the project. The complexity of development cannot be directly determined, it is based on factors such as the novelty of the technology, the novelty of the business problem, team skills and the combination of technologies used (Schwaber & Beedle, 2002).

Section 5 Project rate of change

Questions 23 – 27*: These questions provide information about the rate of change in requirements and other types of changes occurring in the project.

Section 6 Project constraints

Question 28: "Is the budget for this project adequate…"?. Although this is not a part of the theoretical model it was added because budget is an important factor in any project. It can also indirectly reflect the importance of the project to the organisation.

Question 29*: "How much time pressure…"? This question indicates the likely time pressure on the project.

Question 30: "What is the expected level of quality of the software developed"? This question is used because agile methods are not considered suitable for very high quality (life-critical) projects. This is also of interest because the quality level is normally negotiable in agile methods.

Section 7 Project criticality

Question 30*, 31*: System criticality is important because agile methods are not considered suitable for highly critical projects (B. Boehm & Turner, 2004; Cockburn, 2002). This question indicates the criticality of the project.

Section 8 Project team factors

Question 33*: "Estimate the experience level of the team…" This question indicates the team experience level.

Question 34*: "Are team members actively involved in making decisions…" This question assesses the extent of team empowerment on the project.
Questions 35 – 37*: These questions address the issue of communication on the project.

Section 9 Project leader experience
Question 38 – 42*: These questions address the issue of the experience level of the project leader and ask for his or her exact role on the project.

Section 10 Agile method usage
Question 43: This question is made up of all of the techniques used in the five agile methods. The list was drawn from the findings of the analytical framework (see Chapter 4) and then the list was organised into similar types to make it easier for the respondent to understand. Space was provided for the respondent to add techniques not listed.
Question 44: The agile method used on the project was named here.
Question 45*: Iteration length was asked for to check that iterations were used and what the typical length was.
Question 46 – 51: These questions are the agile techniques that require either a 'yes' or 'no' answer.

Section 11 Opinion
Questions 52, 53: This section was used to get the opinions of the respondent as to the benefits of, and problems with, agile methods. A number of known factors (Reifer, 2002a) were provided to make the question easier to respond to.

Interview Design
The interview was semi-structured (Wengraf, 2003), and followed a standard format for most of the questions (see Appendix E). Question 6 and 7 were different for each respondent as they were based on the responses given in section 10 of the questionnaire; the section on exactly which agile techniques were used and how they were changed for this project. Notes were taken at the interview and it was taped with the consent, both verbal and written, of the respondent. The notes were used as the data source. I did not

transcribe the interviews they were only used as a reference if any of the notes taken at the interview were unclear.

Question 1: Why did you decide to start using agile methods? This was an introductory question to put the interviewee at ease and focus on the topic.

Question 2: How did you learn how to use the agile method you use now? This question was included because the amount or type of training may be related to the way the method is used.

Question 3: How long have you been using agile methods in your section/department/work group? This question verifies question 3 in the questionnaire.

Question 4: You use technique *x* often/seldom/never can you tell me the reasons for this? This became a set of questions in the interview. To work out what questions to ask in this section the following procedure was used:

a. The agile method used was determined from the answer to the questionnaire question "What agile method did you use in this project?" (Question 44).
b. Each technique in question 43 belonging to the method nominated in a. above, was marked.
c. Each marked technique that was not checked *always* was selected as a question for further investigation at the interview. Example: XP was selected as the method used. Pair programming was checked *seldom* in the questionnaire. The interview question was then: You seldom use pair programming, can you tell me the reasons for this?

Question 5: Do you believe that your team minimizes documentation during development? This question was not suitable for the questionnaire so it was used in the interview to get greater clarification on the role of documentation.

Question 6: "You mention benefit *y* can you explain this further"? (*y* was a benefit named in questionnaire question 52, "… what are the benefits of using an agile method in this project?") This question was included to draw out any further comments on the benefits of agile methods.

Question 7: You mention disadvantage *z* can you explain this further? (*z* was a disadvantage named in questionnaire question 53 "…what are the disadvantages of using an agile method in this project?") This question was included to draw out any further comments on the disadvantages of agile methods.

Question 8: "If project is completed was the project a success…? This question was included to give the project leader the chance to comment on the success of the project to date.

Question 9: Would you work this way again? This question is designed to give the respondent the chance to explain any further possible changes to the method and to gauge their satisfaction with the agile method they use.

Question 10: Do you have anything else to add? This question closes the interview.

Summary

This chapter has presented an overview of the research design and a description of why the case study method was selected to address research question 2, for what type of environment are agile methods suitable? How effective case study research is carried out was described, and then the research propositions were presented linking the research questions to the instrument questions. The research design was described including the unit of analysis; a *project*. How validity is addressed in the research, the reasons for selecting the cases, and how the procedure was tested and data was gathered were described. A brief description of the data analysis procedures was given. Finally the reasons for including each of the questions in the questionnaire and interview schedule were described.

Chapter 4 THE ANALYTICAL FRAMEWORK

Introduction

This chapter describes the purpose of analytical comparative frameworks in the study of system development methodologies and defines the framework for agile methods. Five agile methods; DSDM, XP, Scrum, ASD, and Crystal methods are analysed using the framework. The source of data for each method is a first or early edition of a publication in book form which fully describes the method. Results of applying the framework are presented in Appendix K and summarised in Appendix J.

What is a Comparative Analytical Framework?

A comparative analytical framework (framework) is a set of criteria against which a methodology is assessed. The framework is analytical because it breaks the methodology into parts, and comparative because it organises the detail of the methodology into a format that makes it easy to compare different methodologies. A framework provides a systematic way of understanding, comparing and evaluating methodologies. It can be used to characterise the philosophy and properties of an individual methodology, to compare one methodology with another, to discover differences and similarities between methodologies, to isolate properties unique to a particular methodology and to determine the strengths and weaknesses of a methodology. A framework can be used to select a suitable methodology for a project from a set of possible methodologies and can show how methodologies with particular weaknesses can be coupled with other methodologies to improve their coverage or efficacy (Avison & Fitzgerald, 2003b; Jayaratna, 1994). A framework can also be used to assess and compare maturity models, metrics, and experience reports of a methodology. The purpose of the framework of this study is to clarify the body of knowledge about agile methods.

Frameworks in Methodology Studies

A large number of very different frameworks have been used to define, describe and compare system development methodologies. The second Comparative Review of Information Systems Design Methodologies (CRIS-2) conference consisted of seven

different frameworks used to analyse design methodologies (Olle et al., 1983). Avison and Fitzgerald (1995a) reviewed a further seven frameworks before describing their own general comparative framework which they used to synthesis information from a number of methodologies of the 1980s and early 1990s. A framework is a well-used method for clarifying the properties of, and comparing, methodologies.

In his introduction to the CRIS-2 conference Sol et al. (1983) described five approaches to the assessment of methodologies:

1. One may try to describe an idealized methodology and evaluate other methodologies against this frame of reference. Then the problem remains how to develop such an ideal.
2. Another approach is to distil a set of important features in an inductive way from a number of methodologies. The methodologies can be compared against this yardstick. Evaluation depends very heavily on the subjectivity in scoring the various methodologies against the framework and on the relative weight given to a feature.
3. A third approach is to formulate a-priori hypotheses on a (partial) ordering of features, and to try to derive a possible framework from the empirical evidence in a number of methodologies. The difficulty in this approach lies primarily in the formulation of hypotheses.
4. Quite another approach is to define a meta-language as a vehicle for communication and as a frame of reference in which various methodologies can be described. The attractiveness of this approach is that implicit, contextual features as well as the process aspects of a methodology can be made explicit. However, a meta-language may have a limited expressive power. It also may blind us for specific features of some methodologies.
5. Finally, a contingency approach tries to relate features of methodologies to contingencies in applying this methodology in specific problem situations. (Sol, 1983, p. 4)

Sol's (1983) list is used by other researchers to categorise their frameworks. They all fall into the category of the meta-language approach (approach 4 above). Jayaratna (1994) analysed methodologies using his NIMSAD (Normative Information Model-based Systems Analysis and Design) framework to investigate if problem-solving is

addressed by the methodology. His framework had four perspectives; the problem situation, the problem solver, the problem-solving process (consisting of eight criteria), and the evaluation. He described each of his selected comparison criteria and the reasons for using it in detail. To show the range of methodologies which could be effectively evaluated with this framework he selected three uniquely different traditional methodologies; Structured Analysis and Systems Specification, ETHICS, and Soft Systems Methodology, and carried out a critical evaluation of each. Jayaratna noted the deficiencies of the methods uncovered by the framework. However it is still possible that particular features or unique properties of a method are not illuminated by the framework if the criteria for such features are not part of the framework. This is a fundamental and insolvable problem with using frameworks that, I believe, can be mitigated to some extent by selecting criteria used in previously proven frameworks, each with a different perspective.

Jackowski's (2003) framework was defined using standard UML class diagram notation and suffers from the problems stated by Sol (1983) (approach 4 above). The purpose of the framework was to allow methodology comparison from the earliest structured methodologies to the most recent agile methods. The framework provided is difficult to use for methodology specification or comparison as it is not verified against any existing methodologies and the 'classes' in the framework are not explicitly defined. This detail is needed for such a framework to be useful. This framework lacks items such as a class for concepts like the philosophy of the methodology. This means that the agile principle of 'trust the developers to do the job' (AgileAlliance, 2001), the XP principle of 'courage' (Beck, 2000), or the ETHICS principle of 'systems designers as teachers, advisers and learners' (Jayaratna, 1994) would not find a place in this framework.

There are two effective frameworks in the agile methods literature. Abrahamsson, Salo, Ronkainen, & Warsta (2002) carried out a systematic review of the existing literature on agile software development methods and used a comparative framework to organise the results, and Williams, Kerbs, & Layman (2004a) developed an evaluation framework for the Extreme Programming method.

Abrahamsson, Salo, Ronkainen, & Warsta (2002) addressed the question of; 'What makes a development method an agile one' (ibid, p. 98) in their systematic study of ten agile methods. The methods were included in the comparison if they met the values published in the agile manifesto (see Table 2). From an analysis of the existing literature (descriptive rather than empirical) they concluded that a development method is agile when software development is incremental, cooperative, straightforward and adaptive. Incremental development consists of rapid cycles with small software releases. Cooperative development is when the customer and developers work constantly together with close communication. Straightforward development occurs when the method is well documented, easy to learn and to modify, and development is adaptive when customers are able to make last moment changes with ease. I believe there is a problem with this characterisation because it is not clear how the authors drew this conclusion from their review of the literature.

The second contribution of Abrahamsson's et al. (2002) study was to describe the main features of each agile method. They used the criteria of process, roles and responsibilities, practices, adoption and experiences, scope of use and current research. They give no reasons for selecting their particular framework criteria, and its coverage is limited, areas such as the application domain, models, tools and deliverables are not covered.

The third contribution of the study was an analysis of each method against the criteria of the methods key points, any special features, identified shortcomings, maturity (against stated criteria), adoption, and coverage of software development lifecycle activities. The problem with this analysis is that it is subjective in its judgements, a fault of all frameworks. To strengthen the conclusions this analysis could have been more specific in its sources by including more comprehensive referencing to the authors' stated principle or practices in the comparative tables.

This study has two main strengths. As a summary of the literature of each of the methods and a comparison of methods this is the only article available which is not produced by the authors of an agile method or a member of the agile alliance (AgileAlliance, 2001). In addition it covers methods that are not usually included in the

set of agile methods such as Pragmatic Programming, RUP and Open Source Software development.

Another framework was published in a series of technical reports by Laurie Williams (Williams et al., 2004a). The purpose of this framework is to gather empirical data from case studies of XP whereas the previous frameworks drew information from published literature. Named The Extreme Programming Evaluation Framework (XP-EF) (see Figure 6), it provides a set of metrics that can be used as benchmarks so that organisations adopting XP can assess both the results of their adoption of XP and how they are adhering to the method. The authors plan to use the data gathered from case studies to study and evaluate the XP method and to evolve the framework. The use of a single framework to compare multiple cases will provide for replication of findings and improve the external validity of the results. Results from two case studies have been published.

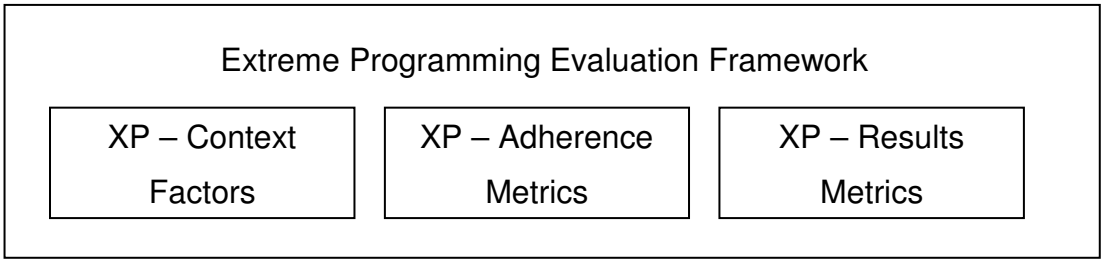

Figure 6: Framework for evaluation of XP (Williams et al., 2004a)

In order to make the case studies comparable, information about the context (environment) of a project is gathered along with qualitative data on factors such as customer satisfaction and developer morale. The framework is comprehensive in its coverage and should provide valuable comparative data on XP projects once data has been gathered. One disadvantage of this framework is it can be time-consuming to gather the data. This would best be achieved by using independent recorder/observers but currently the developers record the metric data.

Frameworks for methodology assessment do have problems. A framework may not uncover properties of a methodology that are not considered in the framework's 'terms of reference' and the assessment is subject to the bias of the assessor. The next section

describes the agile methods framework for this study where these problems are addressed.

A Framework for Agile Methods

The agile methods framework used in this study is based on the generalised framework for comparing methodologies of Avison and Fitzgerald (1995a, 2003b). I selected their framework for a number of reasons. Firstly, it is based on a comprehensive analysis of the academic literature on methodology comparison and the design of comparative frameworks for evaluation of methodologies published from 1982 to1998. Secondly, their framework is designed to provide guidance both when examining an individual methodology and when comparing a number of methodologies. Although it is based on a study of academic publications it is also designed for use by practitioners. Its use was illustrated by comparing the methodologies: Soft Systems Methodology (SSM), Effective Technical and Human Implementation of Computer-based systems (ETHICS), Structured Analysis, Design and Implementation of Information Systems (STRADIS), Yourdan Systems Method (YSM), Information Engineering (IE), Structured Systems Analysis and Design Method (SSADM), Jackson Systems Development (JSD), Object-oriented Analysis (OOA), Rational Unified Process (RUP), Dynamic Systems Development Method (DSDM), Information Systems work and Analysis of Change (ISAC), and Merise. The criteria used in my comparison include eight of the nine criteria developed by Avison & Fitzgerald (1995a). A third reason for selecting their framework was because it includes all of the criteria used by Jayaratna (1994) with the exclusion of the evaluation step. Jayaratna's evaluation is designed to assess a methodology before it is used, during its adoption and after its introduction to an organisation. My agile methods framework is designed only to assess the published characteristics of methods so Jayaratna's evaluation criterion is inappropriate here. Lastly, the framework criteria developed by Avison and Fitzgerald include all of the criteria used by Abrahamsson et al. (2002) in their framework, with the exception of the 'current research' which is discussed in the literature review of this thesis.

One possible way of structuring a comparative framework for agile methods would be to analyse each method against the themes described in the agile manifesto (AgileAlliance, 2001) following Sol's (1983) inductive approach (item 2 above), but the agile manifesto themes are too narrow in scope, and exclude too many other useful

criteria. Also the themes are based on the published thoughts of a group of people many of whom are publishers of agile methods, thus they are subject to bias. Also if the agile methods were assessed against the themes they would need to be 'scored' in some way. This is a subjective scoring and would not be useful for the purposes of this study.

This discussion of the background to the development of my framework was included to show that it is based upon an analysis of a large number of past methodology frameworks and therefore provides a good coverage of possible comparison criteria. A broad range of criteria means that the framework highlights both the characteristics of agile methods, and the characteristics which are not part of the agile methods.

In the early 1980s it was acknowledged that the evaluation of system development methodologies must be based on stated criteria and the purpose of the comparison stated in advance (Rzevski, 1983). Rzevski discussed how different purposes for the comparison framework mean that different criteria will be selected for the comparison. He discussed how frameworks for comparison are influenced by the author's frame of reference and stating the purpose of the comparison helps the audience understand this frame of reference. The purpose of a comparison may be to find the common characteristics of a set of methodologies, to find the most effective methodology to use when building systems in a particular programming language, to determine which are most effective for a particular class of problems, to determine which practices are common to a group of methodologies or it to determine which methodology is the most generally applicable (*ibid*, 1983). The purposes of my agile methods framework are:

1. To define each individual agile method by describing the philosophy, principles, practices and other stated properties of the method.
2. To provide an understanding of the philosophies, principles, concepts and practices that are common to all agile methods leading to a definition of 'agile method'.
3. To provide an evaluation of each agile method by defining what each method consists of, and what it does not, within the constraints of the framework.
4. To allow for a direct comparison of one agile method against another.
5. To provide an understanding of which agile method is most appropriate for a particular set of circumstances, when those circumstances are stated by the method author.

6. To provide insight into agile methods that could be used together when one method alone does not provide for all project contingencies.
7. To show any published metrics for individual agile method practices that could be applied in other (those which share practices) agile methods where they are stated by the method author.
8. To clarify the strengths and weaknesses of each agile method and of all agile methods within the criteria of the framework.

The framework is not used to determine if one method is better than another.

Therefore the framework I have developed is appropriate for addressing research question 1, what is an agile method.

The agile method analytical framework includes the criteria of Avison and Fitzgerald (2003b) and I have added the criteria of identifier, assumptions, values, perspective, metrics, and tailorability. The criteria of product and output are amalgamated because their similarity. This gives a full set of criteria: identifier, philosophy, model, techniques, tools, scope, outputs, practice, product and tailorability. Each of the criteria is now described (see also Table 11).

Identifier

This criterion uniquely identifies the method by name and author. Both the initial and the major publication are listed. The major publication is identified because it describes the method in greatest detail; initial publication year is included, as it may indicate the influence of an earlier method upon later methods, and the country of origin is included.

Philosophy

The philosophy of the method is broken down into its paradigm, assumptions, values and perspective, objectives, domain and target environment.

Paradigm is defined using the categories of Avison and Fitzgerald (1995a) which are based on those of Lewis (1994). The agile framework recognises two paradigms: the objectivist approach and the subjectivist approach. A method designed with the objectivist approach would work well under the assumption that the world and its artefacts can be observed objectively and consist of immutable objects and structures.

A method designed with the subjectivist view would be concerned with the whole picture, the environment of a system, the people who will work with the system (all stakeholders) and emergent properties, which are those properties which will emerge as the method is used. It is important to know where a methodology lies within these categories as a methodology should reflect the same philosophy as that of its users.

Table 11: The analytical framework for agile methods

	Criteria	**Sub criteria**
1	Identifier	Method name Author Date of first publication Major publication Country of origin
2	Philosophy	Paradigm Objectivist / Subjectivist Assumptions and values Perspective Objectives Domain Target: Size of project Type of problem Type of organisation Size of organisation Type of development Type of application Technology environment
3	Model	
4	Techniques	Practices Metrics
5	Tools	
6	Scope	
7	Outputs	
8	Practice	Background Roles and responsibilities Difficulties with the methodology Skill levels
9	Tailorability	

Assumptions and values are the underlying beliefs about system development and systems which influence the method. Understanding these assumptions and values can offer explanations of why the method consists of particular mechanisms or techniques.

Perspective is the view which the method author seems to take of development. This may be user-centred, developer centred, organisation-centred or take some other perspective. The perspective also influences the mechanisms and techniques of the method. The perspective is evaluated by me so it is a personal judgement of the methodology rather than a factor stated by the method author.

Objectives set a boundary on what the methodology can achieve. Avison and Fitzgerald (2003b) note that most commercially available methodologies aim only to provide a technological solution to a given problem whereas academic methodologies tend to have the objective of analysing the problem situation from a broader perspective and do not automatically impose a technological solution on a problem. Some methodologies aim to improve "manual, procedural, managerial, organisational, educational or political"(Avison & Fitzgerald, 2003b, p. 559) processes rather than merely offering a technological solution. Assessing the objectives of each of the agile methodologies to show similarities and differences in objectives is one of the aims of the agile framework.

Domain of a methodology is the range of problems it is designed to address. For example some methodologies address strategic goals by performing a top-down analysis of the whole organisation, others address problems of computerising manual systems, and others address the replacement of existing computerised solutions. Two general types of methodology are distinguished by Avison and Fitzgerald (2003c) those that address the organisational need for a system and those which address specific predefined problems. The domain is important because the type of domain influences the techniques used in the method.

The *target* of the methodology is the environment in which it is applicable. This includes the type of problem, the technology environment, the size and type organisation, the size of projects, the type of development, and the type of application.

Model

The model is often a common factor of different methodologies and shows similarities in philosophy. Models can be conceptual, mathematical, pictorial, or verbal (Avison & Fitzgerald, 2003b). For example object-oriented systems are typically based around a class model of the system showing its objects and interactions. Structured methodologies are often based on the Data Flow Diagram model and other methodologies are based on the Entity model and its variants. Inclusion of the model criteria in the framework is for practical purposes of comparison.

Techniques

The techniques of the methodology are those practical ways used to achieve the aims of the methodology. Techniques common to a number of methodologies may indicate that they are in fact the same methodology but have been given different names for commercial purposes. Metrics for measuring process, progress or technique usage are included here.

Tools

Tool support for a methodology is often important for its acceptance. It may be rejected if the tools are inadequate, expensive or difficult to learn and use. The presence of tools may indicate the level of maturity of the method, as tool support takes some time to become available. In addition tools developed to support one methodology may be useful in another methodology using the same or similar techniques.

Scope

The scope of a methodology could also be called the process or phases of the methodology. This involves defining the phases of systems development and how they are configured, named and carried out. It also includes any project management phase used. The detail of each phase is also important as some methods provide greater guidance on how to carry out some phases than others. If phases are not used any alternative mechanism for moving through the process of development would also be described here.

Output

The outputs from the methodology are the things produced. This may include software, written documentation, training, support and consultancy.

Practice

The practice of the methodology includes: the background (academic or practitioner), the participants in the methodology (which stakeholders take part), skill levels required, and constraints on practice such as recommended team size, difficulties, and problems. Avison and Fitzgerald (1995a) included a section on perceptions of success and failure of the method and the user base (numbers and types of user). Because such details are not readily available this has not been included in the analysis.

Tailorability

Tailorability is the extent to which the methodology can be adapted by its user either before or during use and remain effective.

Reasons for Selecting Early Agile Methods

Five of the earliest agile methods published between 1995 and 2002 were selected for analysis and comparison. There are three reasons for this. Firstly a study of more than five methods is too large a task to achieve in the time frame available for this study. Secondly studying five methods provides enough data for a meaningful comparison of the similarities and differences between each of the methods. Thirdly, they are likely to contain most of the techniques used in subsequently published agile methods.

Chapter 5 RESULTS

Introduction

This chapter presents the findings from application of the analytical framework and the nine case studies. The chapter is organised around the research questions and their propositions. Where necessary hypotheses are developed and tested. First the results of research question 1 are presented followed by an overview of all the cases used to address research question 2. Then each of the propositions of research question 2 is addressed leading to a revised model of the target environment for agile methods. Then the results of other factors assessed in the questionnaire are presented; the budgetary constraints, quality of the system and additional corporate culture factors. Discussion of results is integrated with their presentation.

Results for Research Question 1

The results for research question 1: *What is an agile method?* were provided by application of the analytical framework to five different agile methods (see Appendix K). The following sections present the findings for each of the sub questions of research question 1.

Research Question 1.1 The properties of individual agile methods

Research question 1.1: *What are the properties of individual agile methods?* A table of findings from the analytical framework provides a complete summary of each method including the philosophy, model, techniques, tools, scope, practice, output and tailor-ability of each method (see Appendix J for a summary).

Research Question 1.2 Common properties of agile methods

Research question 1.2: *What are the properties common to all agile methods?* The following common properties were found (see Appendix I). All of the methods were published in the period 1995 to 2002 by authors based in the USA or UK. The methods are primarily objectivist because they address how to provide a technological solution to a given business problem. All use incremental development with iterations of 1 week to four months with 1 month iterations recommended by all of the methods.

Active user involvement, feedback and learning, teamwork, and empowering teams to make decisions, are all important factors in the methods. There is also an emphasis on communication between all of the stakeholders in the project; managers, project leaders, developers, and customers. This is facilitated by frequent meetings.

Each of the methods addresses the need for effective control of incremental development in small teams of between 2 and 40 programmers, with the optimal team size of 3 to 10 people. The methodologies are all created by practitioners and are based on their experiences as developers and they all have a project manager and or developer perspective.

The methods are designed for solving business problems where changes in both requirements and technologies affect the project throughout its life cycle. The main product of development is working software. None of the methods specifies any particular modelling technique and minimising documentation is a goal in all of the methods. XP and Crystal place more emphasis on document minimisation.

Some characteristics are common to subsets of the methods studied. They are as follows:

a. DSDM, ASD and Crystal support tailoring of the techniques to fit the project, whereas XP is most effective if all of the techniques are used as specified. Scrum states that no tailoring is needed.
b. DSDM, XP, Scrum and Crystal recommend testing throughout the lifecycle. However there are no testing techniques recommend by all of the three methods.
c. DSDM, XP, Scrum and Crystal recommend that the customer is located with the developers or on-site and available at all times.
d. Time pressure is a problem addressed by ASD and DSDM, it is a project variable to be controlled in Scrum and XP, and is not mentioned in the Crystal methods.
e. XP, Scrum and ASD all support the idea of emergent properties of effective teams and state that their techniques will support this. Collocation of teams is another technique shared by these methodologies along with Crystal.

Emergence may be related to collocation but this is not clearly explored in the method descriptions. DSDM is not concerned with how teams are located.

f. Object technology and Internet technology, including the assumption that knowledge of UML will be used as needed during development, is common to XP, Scrum, ASD and Crystal. DSDM assumes either structured or object-oriented development with appropriate models created if necessary.

This leads to the conclusion that the agile methods have the common properties shown in Table 12.

Table 12: Common properties of agile methods

Common Properties of Agile Methods

- Published between 1995 – 2002 in the USA and UK
- Objectivist methods which provide technical solutions
- Address business problems
- Practitioner based
- Project manager and developer perspective
- Incremental development
- Iterative development with 1 month iterations optimal
- Projects undergoing constant change
- Active user involvement
- Feedback and learning
- Teamwork
- Empowered teams
- Communication between all stakeholders is critical
- Small teams of 3-10 programmers is optimal
- Frequent meetings, daily is optimal
- Working software is the main product of development
- Modelling techniques are not mandated
- Minimise documentation

Research Question 1.3 Differences between agile methods

Research question 1.3: *What are the differences between agile methods?* The difference between the methods is that they each serve a different purpose. They share some goals and some techniques as described above but it is the purpose of each methodology which distinguishes it from the others. The purpose of each of the methodologies is as follows:

- DSDM is a framework for RAD development.
- Scrum is a methodology for project management of iterative development.
- XP is a methodology for software development in high change environments using small teams and standard software engineering techniques to satisfy customer needs and maintain effective teams.
- ASD is a framework for managing software development projects which are under intense time pressure and have rapidly changing requirements. The method is based on complex adaptive systems theory and uses RAD techniques.
- Crystal methods are for designing a methodology to suit a specific project.

Each of the methods belongs to the family of agile methods but they are designed to achieve different, sometimes overlapping, purposes.

Research Question 1.4 Unique properties of agile methods

Research question 1.4: *What properties are unique to agile methods within the set of agile methods?* The analysis shows that there are three techniques not previously used by other system development methodologies or any other agile method. These are techniques described in XP; test-first development, pair programming, and the use of a system metaphor.

Research Question 1.5 Combinations of agile methods

Research question 1.5: *What combinations of agile methods are possible?* ASD and DSDM are frameworks therefore they do not specify how to carry out each technique described in the method or even what techniques must be used. This means that they can be used with techniques from other methods. ASD and DSDM could both be used with techniques from XP and Scrum. Scrum and XP can be effectively used together with Scrum providing the management practices and XP providing the software development techniques. Crystal method can also be used with XP. XP acts as the base

method which is tailored for each project using Crystal principles. Scrum, ASD and Crystal can use any software engineering techniques to fulfil their goals as long as those techniques achieve the goals of the methodology.

Research Question 1.6 Published problems

Research question 1.6: *What are the published problems with agile methods?* There is no consistent pattern or problems with all of the methods. Scrum reports no problems, ASD reports no problems except that it is not compatible with CMM initiatives. DSDM and XP report problems will occur when their base environmental conditions are not met. Crystal reports problems when projects become large and distributed.

Agile methods defined

Research question 1: *What is an agile method?* Based on the answers to research questions 1.1 to 1.6, I have defined an agile method in this way:
An agile method is a software development methodology designed for the management and support of iterative and incremental development of business systems in environments where change is constant. Agile methods use software development techniques that enhance teamwork in small empowered teams and support active customer involvement. An agile method is designed to produce working software early using communication, feedback, learning and frequent meetings rather than modelling and documentation. Agile methods adapt existing software development techniques to achieve these goals.

Results for Research Question 2

This section begins with the background of each of the nine cases, then I address each sub question and proposition of research question 2. In describing the projects, key informants are given pseudonyms and in some cases the business activity is not reported to maintain the anonymity of the organisation. Numbers of staff are placed into ranges to maintain anonymity. All quoted phrases come from the interviews and all data is drawn from the questionnaire, web-site search, and interviews.

Case 1 Alpha

Organisation type Privately owned company, founded early 1980s
Organisation size 100-150 full-time staff

Business activity	Manufacturing equipment
Market	New Zealand and international
Staff	10-15 staff involved in software development
Key informant	April; Software developer and method coach
Development method	XP

Type of development In-house development

Project information

This organisation has used XP for over five years for on-going development of two products. Product development was divided into sub projects. The software products form part of a piece of manufacturing equipment consisting of machinery, electronics, and software. The organisation customises the software to suit different client's needs. Alpha was current and ongoing at the time of this study and its purpose was to complete a contract to develop a one-off software product for external clients. The software was a control system to enable further automation of existing plant and development was considered complete after testing and release to the customer. The software had both a console and a Windows interface, object-oriented concepts and languages and a relational database were used in development. Alpha had six full-time and one part-time developer.

April had experienced formal object-oriented methods and was not impressed with them because they "did not allow you to develop software fast enough" and they "wasted time". These earlier experiences made her think that XP would solve some of the problems experienced in the past. April taught herself about XP from reading books and participating in an international on-line mailing list. She is personally committed to XP, and believes it is a very good way to develop software. April found XP to be "really effective" because "we have a good situation in this organisation", the team has "total control" and "we have a champion to drive it" who is experienced in software development and project work. As April explained, this person "can help in training and explain why certain practices are better than others", and "people have to be comfortable and understand why they are doing things and the experienced person is needed to do this". So far the project is a "huge success" and "we have hit targets using XP". Success is measured by the sales of their product into overseas markets and by the

organisations ability to make major changes to the software product on time. April's team will "definitely" work this way again.

Case 2 Beta

Organisation type	New Zealand Government State Owned Enterprise formed in the early 1990s
Organisation size	150-200 staff
Business activity	Providing information and presentation services
Market	New Zealand and international
Development staff	45-50 staff involved in software development
Key Informant	May; Project director
Development method	XP

Type of development

The organisation carries out development in-house, purchases software from outside sources, completes one-off contracts by customising pre-developed software, and provides services and support for existing software systems. The company has been using XP for about 5 years but has significantly tailored the method over that time and now focuses on the techniques of pair programming, user stories and refactoring.

Project information

Beta took 13 months to complete and the software is now in use at the client company. The project was an enhancement to an existing system that was adapted to suit the client's particular needs for a 3D, real-time data visualisation product. The software had both a console and a Windows interface and was based on ideas from gaming software. The product was developed using object-oriented concepts and languages, a relational database, and a control technology component. The project had 10 full-time and one part-time developer, and May the project director.

XP was first used on Beta and initially the whole of XP was adopted but it has been tailored since the project began. May decided to move to XP because she had heard that it would improve code quality and productivity. The company at that time was not using any traditional methodology. XP seemed to "fit in with the company's philosophy of trying to be adaptable and agile". May attended a conference where Kent

Beck, author of the XP method (Beck, 2000) was speaking and also did some reading and talking to others before deciding to adopt XP. The whole team on the project had formal training in XP.

May also stated that the organisation is ISO 9000 compliant and the organisation has had no problems with aligning ISO 9000 with XP.

May believes that "organisational culture is what will have the biggest effect on whether XP works or not" and that Beta was "a resounding success! – but not as a result of XP". The positive aspects of XP will be retained in future projects (see Table 51).

Case 3 Delta

Organisation type	Government funded, provides services to the New Zealand government
Organisation size	13,000 – 14,000 staff
Market	Provides services to their own staff
Development staff	50-55 are involved in software development
Key informant	June; Project manager and analyst
Development method	A combination of XP and Scrum

Type of development

All development is in-house and involves mainly unique development with some customisation of brought in-products.

Project information

Project Delta is estimated to take at least three years and was current and ongoing at the time of this study. Delta is a replacement and enhancement of a legacy system carried out to extend its life, to improve the usability by changing the interface, to modernise the base technology, and to realign the data with other existing systems. Delta has been divided into modules each comprising a sub-project of the main project and each sub project is assigned two to three developers. Developers are dedicated to one sub project. Module dependencies are a problem in this project and increase its complexity. At the time of the case study there were 13 full-time and six part-time developers involved in Delta. Delta is designed to provide management information and office

automation using systems software linked to an intranet system. The system interface uses middleware and web software to deliver both static and dynamic pages and is designed and built using object-oriented concepts, languages and a relational database.

June did not instigate the use of Scrum; the idea came from another employee involved in managing the same project. A local training company, who covered a combination of Scrum and XP techniques, trained the whole team in-house over five days. However the project was not started until a few months after the training and there were some problems with staff understanding and complying with the techniques. Scrum/XP had been in use for one year on Delta.

Case 4 Zeta

Organisation type	Private company
Organisation size	300-350 staff
Business activity	Servicing the utility industry with software
Market	UK
Development staff	2
Key Informant	July; Project manager
Development method	DSDM

Type of development

The organisation provides specific products for external customers by completing one-off contracts and they also carry out in-house development to support their own organisation.

Project information

Zeta took 2 months to complete and was a new system which is now in use with its intended user base. There were four full time developers working on the team which involved replacement of a paper-based system with an intranet browser-based one. The system included components involving marketing, management information, purchasing, sales and systems software. The application was a dynamic web application created using object-oriented concepts, an object-oriented language and a relational database.

July learned about DSDM about 10 years ago in the UK from a mentor experienced in the method and by studying the literature published by the DSDM consortium. July found that DSDM "fitted well with the marketing mindset of unformed ideas [requirements]", and that iterative development was useful. July said the project was "incredibly" successful and she would work this way again.

Case 5 Theta

Organisation type	Private company; a subsidiary of a larger private company
Organisation size	1400-1500 people
Business activity	Information and communications technology provider
Market	New Zealand and Australia
Development staff	150 staff involved in software development in the subsidiary
Key Informant	August; Business manager

Development method RUP combined with XP

Type of development

Provides specific products for external customers by completing on-off contracts, customising pre-developed software and providing service and support

Project information

Theta was a contract to provide software to a large government organisation by developing the software on the client site. At the time of the study Theta was in progress with an estimated life of 11 months. Theta was an enhancement to an existing system that involved a total project team of over 30 full-time people. The purpose of Theta was to provide a system to manage the daily operational activities of the client by development of a management information system available both by web and Windows interfaces and mobile devices. The software was designed and built using object-oriented concepts, an object database, object-oriented languages and a relational database.

August said that initially the organisation had no formal process before they moved to RUP in the late 1990s. They found over five years that using RUP was prohibitive due to the time and cost of generating the specified documentation so they refined RUP to

incorporate more agile techniques in order to reduce costs while retaining software quality. August learnt about agile methods from reading and attending conferences and decided to "swap out" some RUP techniques for XP techniques. The process continues to be "incrementally refined". The project is a success so far and August would work this way again.

Case 6 Iota

Organisation type	New Zealand Government department
Organisation size	600 staff
Market	Provides services to their own staff
Development staff	Unknown
Key Informant	Sarah; RUP expert, coach and mentor
Development method	RUP

Type of development

Development in the organisation is in-house with some customisation of brought-in products.

Project information

Project Iota was a replacement for an existing system that arose from the need to accommodate policy changes. The application was redesigned to enhance functionality. The business need was to satisfy claims from external parties. The system involved transaction processing and the final application consisted of a Windows interface supported with middleware and was developed using object-oriented concepts, object-oriented languages and a relational database.

Case 7 Rho

Organisation type	Government department
Organisation size	4500-5000 staff
Market	Provides services to their own staff
Development staff	Not known
Key Informant	Sally; Programmer/Analyst
Development method	ad hoc

Type of development

The development in this organisation involves both customising pre-developed software and providing service and support for existing software systems. Development is in-house and involves customisation of brought-in products and own development.

Project information

Rho was underway at the time of this study and was estimated to take nine months. It was an enhancement to an existing system requiring three developers. The application was an accounting and management information system implemented with both a web and Windows, LAN/Server architecture and a relational database.

Case 8 Tau

Organisation type	Private company providing software solutions
Organisation size	12 staff in New Zealand
Market	New Zealand and international
Development staff	3 involved in software development
Key Informant	Samina; Project leader, developer and tester
Development method	ad hoc

Type of development

This organisation produces specific products for external customers by carrying out mass production of shrink-wrapped products, they also provide service and support for their products.

Project information

Project Tau was an enhancement to an existing product that is tailored for particular clients' needs. The project took about three months to develop and release to its intended user base. One full time and one part time developer worked on the product. Other team members such as marketing, sales and IT support staff were also involved. The application was designed and built using object-oriented concepts, object-oriented languages, a relational database, web services and daemons and has a Windows interface.

Case 9 Chi

Organisation type	Government funded, owned by the New Zealand government, founded in early 1990s
Organisation size	300-400 staff
Business activity	Provides scientific services
Market	New Zealand and international
Staff	10-15 staff involved in software development
Key informant	Saneta; Analyst / Programmer / Support
Development method	ad hoc
Type of development	Various

Project information

Development at this organisation includes completion of one-off contracts for specific products for external clients, customising pre-developed software, and providing service and support for software systems. In-house development is carried out to support the running of the organisation and includes both own-development and customisation of brought-in products. Some software is also developed in conjunction with external groups. Chi was a four-month project involving the maintenance and enhancement of an existing system that provided accounting, scientific and management information. The system was developed as an intranet for processing transactions and had a Windows interface, incorporated middleware, and was designed using object-oriented concepts, object-oriented languages, and a relational database. At the time of this study the system was completed and is in use with its intended users. The project had four full-time and two part-time developers.

Research Question 2.1 Effect of the project environment

Research question 2.1 is: *How does the project environment affect the use of an agile method?* This research question is made up of four propositions:

1. The full method is not used in practice; the method is tailored-for-use.
2. There are specific environmental conditions that are suitable for agile methods.
3. An agile method is not tailored when it is used in its target environment.
4. An agile method is tailored when it is not used in its target environment.

Proposition 1 is addressed first, proposition 2, 3 and 4 are addressed by investigating the correlation between each of the target environment criteria and the extent of agile method usage (tailoring). Where a correlation cannot be calculated a dot plot is used to illustrate any relationship found.

Proposition 1 The method is tailored for use

Each of the projects was assessed to determine the extent of tailoring of the method. Each of the agile methods is made up of a unique combination of development techniques (see Appendix G). The following analyses were carried out to clarify both the differences between projects using agile methods (agile projects) and projects using non-agile methods (e.g. ad hoc and RUP), and the differences within the projects using agile methods. The data is shown in Appendix P.

The extent of method tailoring was assessed using two measures:

Equation 1: The percentage of techniques used:

$$\% \text{ Techniques used} = (T_c / T_{mc}) \times 100$$

where

T_c = count of techniques used on the project

T_{mc} = count of all techniques in the method

Count was 0 or 1 (used technique, did not use technique) for each technique selected by the project leader on the questionnaire. Only those techniques that belong to the selected method were included (i.e. if the respondent selected techniques that do not belong to their nominated method then this technique was not included in the count)

Equation 2: The percentage of agile method usage:

$$\% \text{ Agile method usage} = (\Sigma T_q / 3T_{qm}) \times 100$$

where

T_q = quantity of techniques used

T_{qm} = quantity of techniques in the method

Quantity was 0, 1, 2, or 3 (never used, seldom used, often used, always used the technique) as assessed by the project leader in the questionnaire. Only those techniques that belong to the selected method were included (i.e. if the respondent selected techniques that do not belong to their nominated method then this technique is not included in the summation)

Table 13 and Figure 7 show the extent of tailoring on each project using an agile method. Project Delta was assessed twice, once against XP and also against Scrum since a combination of techniques from the two methods was used on that project. These results show that each project tailored the method. Alpha, Theta and Zeta used all of the techniques that belonged to their chosen method but they did not use them 100% of the time. The other projects also tailored their method not only by reducing the number of techniques they chose to use but also in the extent to which they used

Table 13: Extent of method tailoring on projects using agile methods

Project	Alpha	Beta	Delta	Delta	Theta	Zeta
Development method used	XP	XP	XP Scrum	XP Scrum	XP RUP	DSDM
Compared with	XP	XP	XP	Scrum	XP	DSDM
Total techniques in method	19	19	19	13	19	13
Number of techniques used	19	18	16	11	19	13
Techniques used %	100%	95%	84%	85%	100%	100%
Total quantity of techniques in method	57	57	57	39	57	48
Total quantity of techniques used	55	32	38	26	36	45
Extent of usage %	96%	56%	67%	67%	63%	94%

them. Beta used the techniques of their chosen method only 56% of the time. These results show that the method is tailored in each project but there is a range of tailoring from 96% to 56% for these projects using agile methods. Proposition 1: the method is tailored for use, is true in each of these cases.

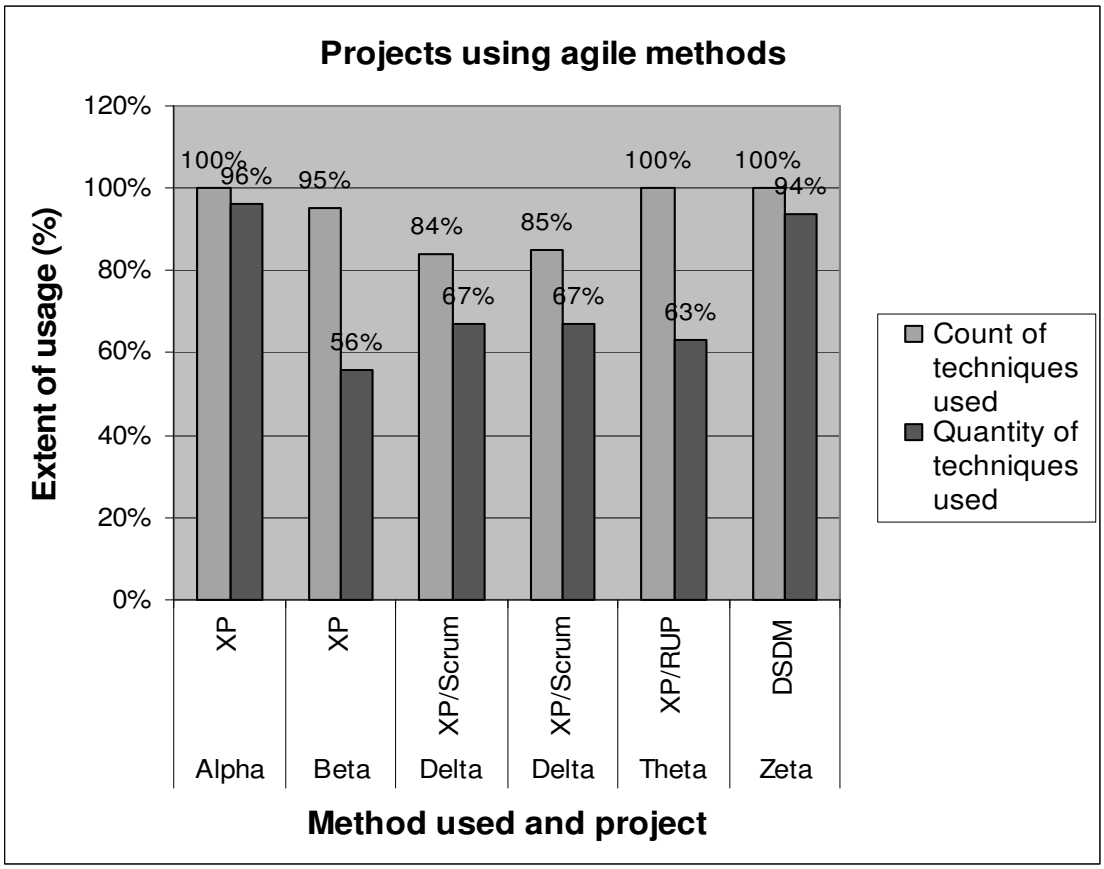

Figure 7: Method tailoring on projects using agile methods

In order to compare all of the projects, agile and non-agile, it was necessary to compare each non-agile project with each of the agile methods. This provided five sets of data, each set comparing a project with the extent of method usage for each of the five methods.

Appendix M shows each of the projects compared with all of the techniques listed in the questionnaire for the XP method, the DSDM method, the Scrum method, the Crystal method, and the ASD method. The method-in-situ matches the method-as-published when the techniques belonging to that method are used at the level of *always* (or 3) (see Appendix H for the techniques belonging to each method and Appendix D for their listing in the questionnaire). For example if the project was using XP then the number of XP techniques used should be 100% (all 19 techniques used) and the quantity of each XP technique should be 57 (all techniques used always: $19 \times 3 = 57$).

The results show (Appendix M, Table 60, Table 61) that although there are only two techniques common to all agile methods, iterative and incremental development (see Appendix H) the projects use many techniques from the other methods. For example project Zeta using DSDM also used all of the techniques from XP, DSDM, Scrum and ASD, and 90% of the techniques from Crystal; however when the extent of usage was calculated there were no projects using any method at the 100% level.

Another pattern that is shown in the graphs (see Appendix M, Figure 18, Figure 19, Figure 20, Figure 21, Figure 22) is the lower level of usage of the techniques by non-agile projects (this is summarised in Table 14).

Table 14: Differences in usage between agile and non-agile projects

Method	Extent of usage (%) Agile projects N=5		Extent of usage (%) Non-agile projects N=4	
	Upper limit	Lower limit	Upper limit	Lower limit
XP	96	56	54	18
Scrum	77	41	36	3
DSDM	94	52	54	19
Crystal	80	57	50	0
ASD	83	53	50	0

This is an expected pattern because non-agile projects were not expected to use techniques from the agile methods. This provides evidence that agile methods are distinct from ad hoc development and RUP development. However there is a high level of overlap in technique usage. This can be observed in the use of XP. Beta, who nominated XP as their method, used XP at the 56% level and Tau, who used no development method, used XP at the 54% level.

Projects report that they are using a particular method and they may be using all of the techniques that make up that method, but they may be using them to a much lesser extent than expected (at 100%) and the range between projects can vary. For example (see Appendix M, Table 63) comparing Alpha, Theta and Beta; Alpha and Theta used 100% of the XP techniques and Beta used 95% of the XP techniques. But observing the extent of usage a different picture appears; Alpha usage was 96%, Theta was 63% and Beta was 56%.

Another instance where the nominated method did not give an accurate picture of actual usage was in Delta (Scrum method). Delta nominated Scrum as their chosen method but they were also trained to use some XP techniques. Delta used XP at the 67% level and Scrum at the 67% level (see Appendix M, Figure 18, Figure 19). Zeta (DSDM) also selected all of the Scrum techniques (100% of techniques and 77% usage level) and used them at a higher level than Delta (see Appendix M, Figure 19).

Crystal and ASD were not nominated by any project. The results show (see Appendix M, Figure 21, Figure 22) that the Crystal and ASD techniques were still used, less in the non-agile projects and more in the agile projects. The results for these methods are less accurate because of the smaller number of techniques making up the method (see Appendix H).

An analysis of the techniques used on the projects is shown in Table 15. This shows that there are five techniques that are not unique to agile development. They are customer-on-site, 40 hour week, requirements are prioritised, and design and coded solution are kept as simple as possible. As these are simple actions and not techniques requiring training (such as test-first development or user stories) so this is not an unexpected result.

Table 15: The techniques used on projects

Techniques used on agile and non-agile projects			
	Techniques from the five agile methods. Q43 on questionnaire	Used on all projects agile and non-agile	Used on all agile projects
1	Concurrent development		
2	Iterative development		✓
3	Time boxing (iterations of set length)		✓
4	Incremental development		✓
5	Evolutionary prototyping		
6	Small releases of software product		✓
7	Component development		✓
8	Test first development		✓
9	Daily builds of complete system		✓
10	Automated regression testing		✓
11	Refactoring of code		✓
12	Testing throughout each iteration		✓
13	Software inspections		✓
14	Customer on-site	✓	✓
15	Method coach on site		✓
16	Tester(s) collocated with team		✓
17	Customer focus groups		
18	Rooms organised for pair programming		
19	Whole team works in same office/floor		✓
20	Dedicated meeting space		
21	Pair programming		✓
22	Coding to an agreed standard		✓
23	Collective ownership of code		✓
24	40 hour week	✓	✓
25	Sprint Goal		
26	Daily team meetings		
27	Iteration planning meeting		✓
28	Planning game		✓
29	Reflective workshops for adaptation		
30	User stories		
31	System metaphor developed		
32	Only what has direct business value		✓
33	Requirements are prioritised	✓	✓
34	Changes to requirements are negotiated		✓
35	Joint Application Development (JAD)		
36	Design is kept as simple as possible	✓	✓
37	Coded solution is kept as simple as possible	✓	✓
38	Risk assessment at each iteration		✓
39	Product Backlog		
40	Sprint Backlog		
41	Release Backlog		
42	Milestones to track progress		✓
43	Product Backlog Graph metric		
44	Sprint Backlog Graph metric		
45	Function point counts		
46	Project post mortem		
47	Feasibility study		
48	Business study		
49	Resource requirements analysis		
50	MOSCOW rules		
52	Unique methodology at start		
53	Tailored existing methodology at start		

These results show there is a distinct difference between agile and non-agile projects with respect to the techniques used. However an ad hoc or RUP project will perform some of the techniques belonging to the agile methods because of the techniques they have in common. Also when an organisation nominates a method they may be using it fully or partially making it necessary when comparing projects to make a careful analysis of the extent of usage rather than just counting the techniques used.

Proposition 2 Environmental conditions for agile methods

Proposition 2 is: *There are specific environmental conditions that are suitable for agile methods*. To address this proposition the results for each criteria of the target environment model (see Appendix F) were analysed. The data was drawn from the questionnaire, web site search and interviews (shown in appendices L, N and P). To carry out a comparison each agile project was assessed against its nominated method, the non-agile projects were assessed against XP. The data and explanation for this choice is given in Appendix M. Where a trend was visible a scatter plot or dot plot illustrates the data and either Spearman's correlation coefficient (for ordinal data) or Pearson's product moment correlation coefficient (for ratio data) was calculated to indicate any correlation between the usage of agile method techniques and the environmental factors in the target model. Where statistical tests are used the hypothesis tested is always of the form:

H_o there is no relationship between the project environment factor (size of team, style of communication etc.) and the extent to which the agile method is tailored for use.

H_1 there is a relationship between the project environment factor and the extent to which the agile method is tailored for use.

Results and analysis of the model criteria are now presented. The results of statistical analysis of all project data are shown in Appendix O (organisational culture data) and Appendix Q (project data). A summary table of results for each of the model criteria is provided at the end of this section.

1. The organisation values feedback and learning

Projects with an environment of feedback and learning are also the projects with higher usage of agile techniques (see Table 16). Feedback and learning are not addressed

directly in Questions 5a and 8a. 5a asks if the leadership in the organisation is mentoring, facilitating and nurturing and 8a asks if the organisation emphasises human development and high trust, openness and participation persist. Therefore the measure is indirect. I have assumed that feedback and learning are more likely in an organisation with these qualities. The results are illustrated in scatter plots in Figure 8 and Figure 9 and data and statistical test results are shown in Table 16 and Table 17.

Table 16: Data for organisation values feedback and learning

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q5a	4	5	4	5	4	3	3	2	2
Q8a	5	4	4	4	4	4	2	1	3
1 Almost never, 2 Seldom, 3 Sometimes, 4 Often, 5 Almost always									

Table 17: Statistical results for organisation values feedback and learning

Question	Technique	Rho	n	Test	Significance	Reject H_o
Q5a	Spearman's rank correlation coefficient	0.842	9	2-tailed	0.01 level	Yes
Q8a	Spearman's rank correlation coefficient	0.858	9	2-tailed	0.01 level	Yes

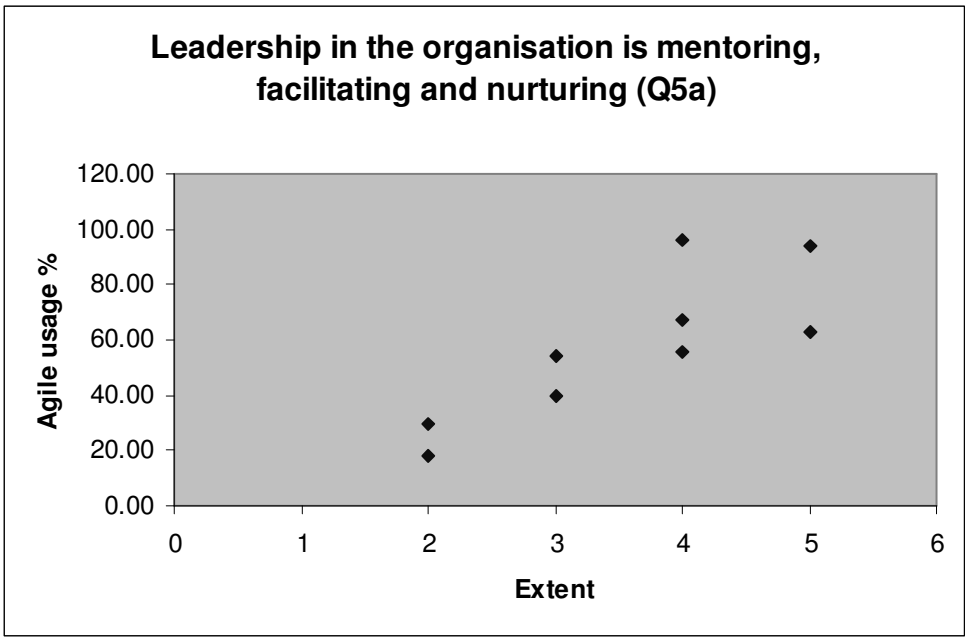

Figure 8: Results for organisation values feedback and learning, Q5a

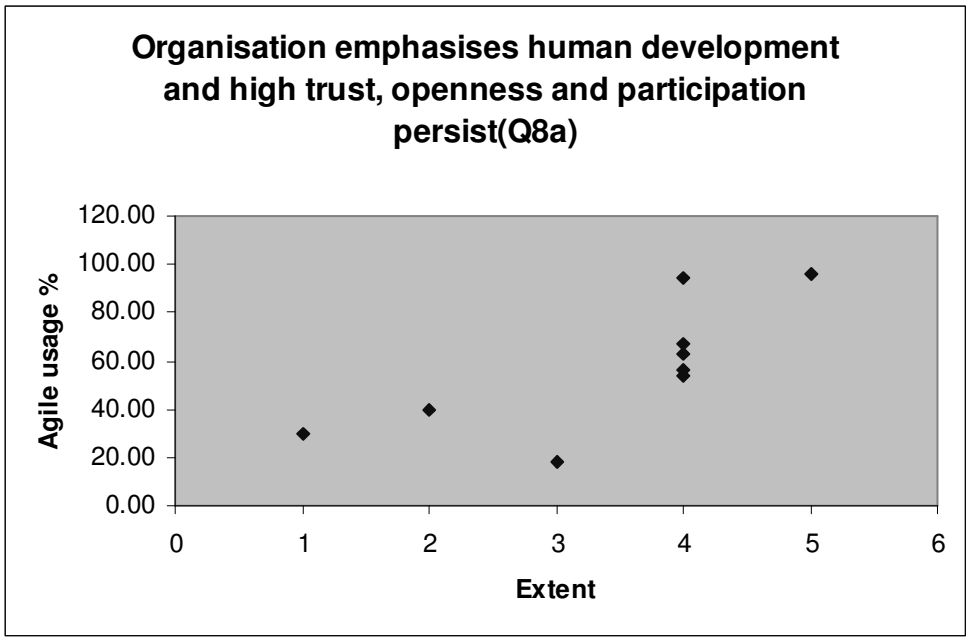

Figure 9: Scatter plot for organisation values feedback and learning Q8a

These results support the model criterion.

2. The organisation values teamwork

Q6a asks if the management of employees in the organisation is characterised by teamwork, consensus and participation. Q9a asks if the organisation defines success on the basis of the development of human resources, teamwork, employee commitment and concern for people. The data is shown in Table 18 followed by a scatter plot of the data for Q6a (see Figure 10) and the results of the statistical tests (see Table 19).

Table 18: Data for organisation values teamwork

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q6a	5	5	3	4	4	4	3	3	3
Q9a	3	5	4	5	5	3	3	1	3
1 Almost never, 2 Seldom, 3 Sometimes, 4 Often, 5 Almost always									

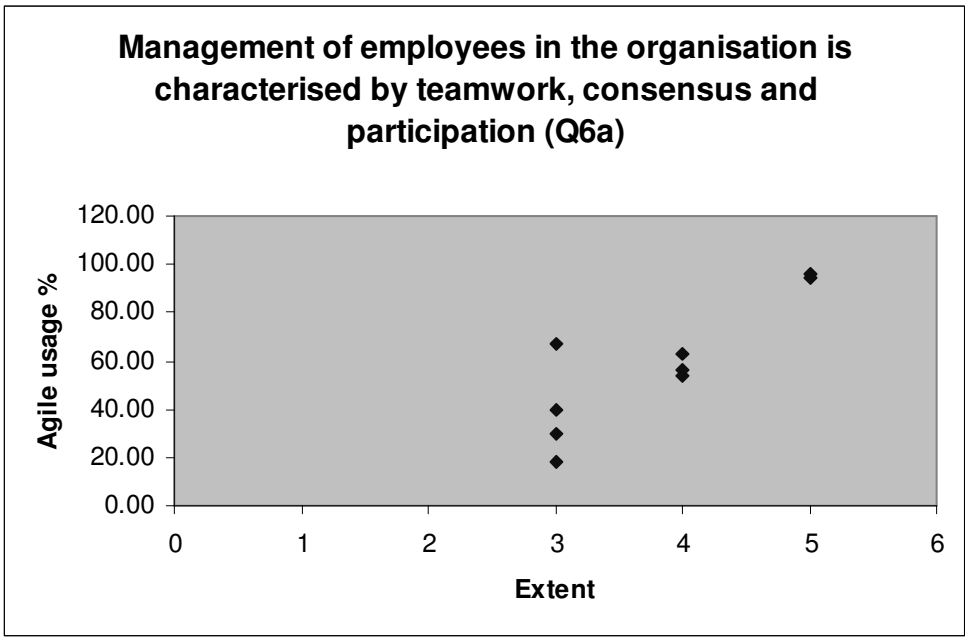

Figure 10: Scatter plot for organisation values teamwork Q6a.

Table 19: Statistical results for organisation values teamwork

Question	Technique	Rho	n	Test	Significance	Reject H_o
Q6a	Spearman's rank correlation coefficient	0.748	9	2-tailed	0.05 level	Yes
Q9a	Spearman's rank correlation coefficient	0.541	9	2-tailed	none	No

The two assessments are contradictory so weak evidence is provided to support this model criterion.

3. The organisation values face-to-face communication

The data for this criterion is shown in Table 20. The number of contacts with the customer is normalised to weekly. Q36 assessed within-team communication and Q37 assessed team-management communication. The results of the statistical tests are shown in Table 21 and dot plots show the pattern of responses for Q36 and Q37 (Figure 11 and Figure 12). For both intra-team and team-management the communication style on agile projects was *informal, mainly informal* or *balanced* indicating that the communication style is informal. This is not unique to agile projects, it also common to non-agile projects. These results provide weak support for the model criterion because no agile projects reported a formal communication style, but this is not unique to agile projects.

Table 20: Data for organisation values face-to-face communication

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q35	Lots!	15	0.5	100	5	5	0	0	5
Q36	4	3	4	4	5	4	2	5	5
Q37	4	4	3	4	5	4	4	4	4
1 Formal, 2 Mainly formal, 3 Balanced, 4 Mainly informal, 5 Informal									

Table 21: Results for organisation values face-to-face communication

Question	Technique	Rho	n	Test	Significance	Reject H_o
Q35	Pearson's rank correlation coefficient	.258	7	2-tailed	none	No
Q36	Spearman's rank correlation coefficient	-.443	9	2-tailed	none	No
Q37	Spearman's rank correlation coefficient	-.183	9	2-tailed	none	No

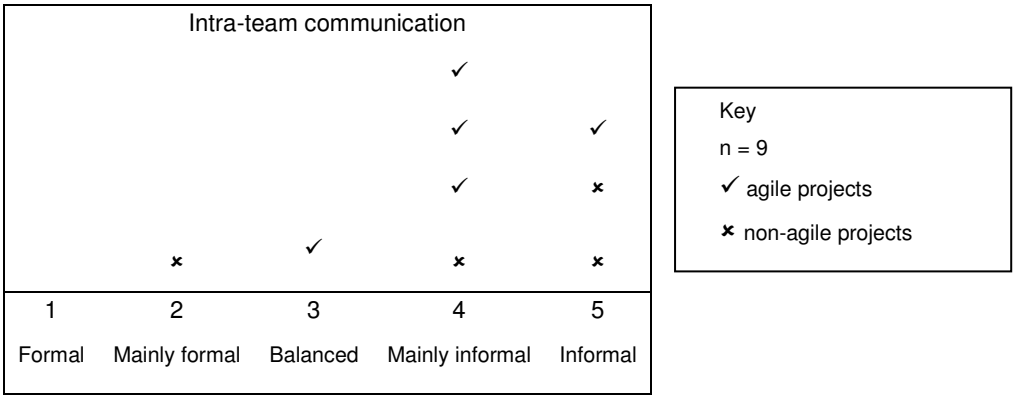

Figure 11: Dot plot for Q36 intra-team communication

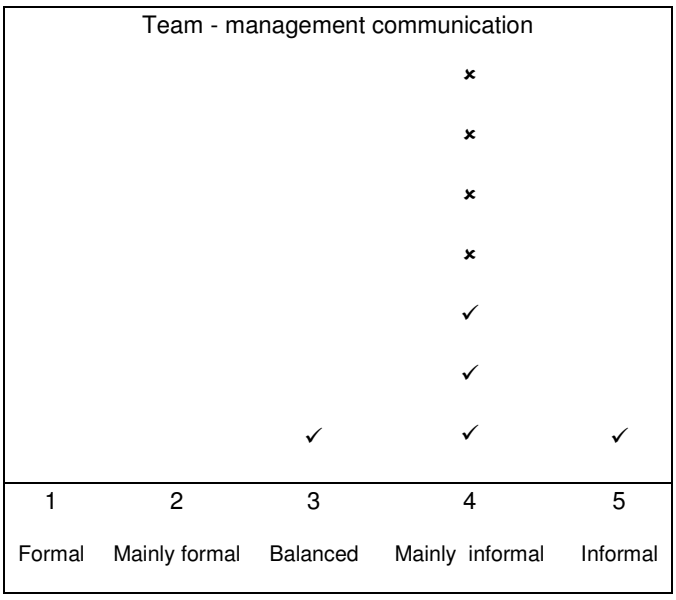

Figure 12: Dot plot for Q37 team-management communication

4. The organisation enables empowerment of people

Question 4b addresses empowerment indirectly. I have assumed that if the organisation is dynamic, entrepreneurial and risk-taking then this implies that the people within the organisation are empowered. Question 34 asks if the team are involved in decision making which is another indirect measure of empowerment. Data is shown in Table 22 and results are shown in Figure 13 and Figure 14, statistical results in Table 23.

Table 22: Results for organisation enables empowerment of people

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q4b	5	5	4	4	5	4	1	2	2
Q34	5	5	5	4	4	5	3	4	2
1 Almost never, 2 Seldom, 3 Sometimes, 4 Often, 5 Almost always									

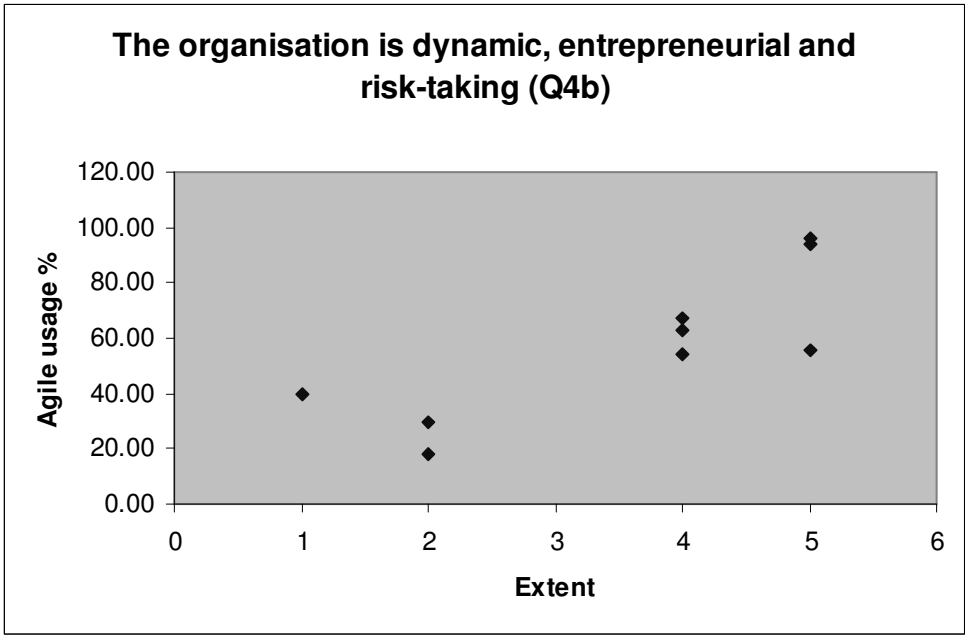

Figure 13: Scatter plot for organisation enables empowerment of people Q4b

Table 23: Statistical results for organisation enables empowerment of people

Question	Technique	Rho	n	Test	Significance	Reject H_o
Q4b	Spearman's rank correlation coefficient	0.806	9	2-tailed	0.01 level	Yes
Q34	Spearman's rank correlation coefficient	0.780	9	2-tailed	0.05 level	Yes

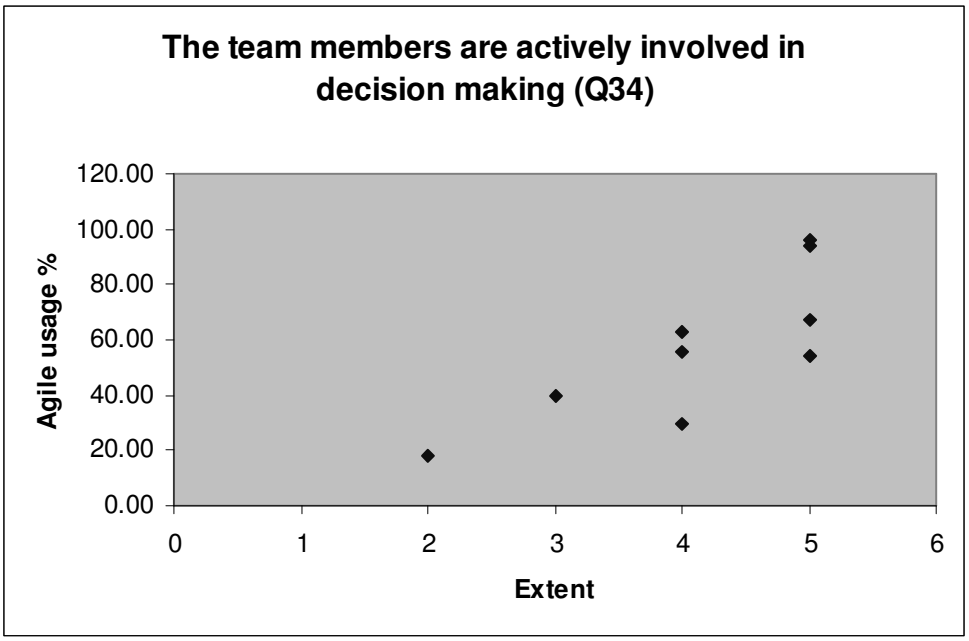

Figure 14: Scatter plot of organisation enables empowerment of people Q34.

These results support the model criterion.

5. The organisation is flexible and participative and encourages social interaction

This criterion is assessed using the same question (Q6a) as criterion 2; 'The organisation values teamwork'. These criteria are therefore amalgamated in the revised model.

6. Social interaction in the organisation is trustful, collaborative, and competent

This criterion is addressed indirectly by asking if the organisation emphasises human development and high trust, openness, and participation persist (Q8a). This question was also used for criterion 1 (Q5a, Q8a). These criteria are therefore amalgamated in the revised model.

7. Communication in the organisation is informal

This result is taken from the same source as model criterion 3 (Q35, 36, 37) the organisation values face-to-face communication. These criteria are therefore amalgamated in the revised model.

8. The management style is that of leadership and collaboration

Q4a is an indirect measure of management style. The question asks if 'the organisation is a personal place, like an extended family, where people share a lot of themselves'. Q5a asks if the leadership is mentoring, facilitating and nurturing and is also used to assess criterion 1. The results are shown in Table 24 followed by the statistical tests in Table 25 and a scatter plot in Figure 15 shows the correlation between leadership and collaboration and extent of agility on each project.

Table 24: Data for management style is that of leadership and collaboration

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q4a	5	5	4	4	5	3	2	4	3
Q5a	4	5	4	5	4	3	3	2	2
1 Almost never, 2 Seldom, 3 Sometimes, 4 Often, 5 Almost always									

Table 25: Statistical results for management style of leadership and collaboration

Question	Technique	Rho	n	Test	Significance	Reject H_o
Q4a	Spearman's rank correlation coefficient	0.719	9	2-tailed	0.05 level	Yes
Q5a	Spearman's rank correlation coefficient	0.842	9	2-tailed	0.01 level	Yes

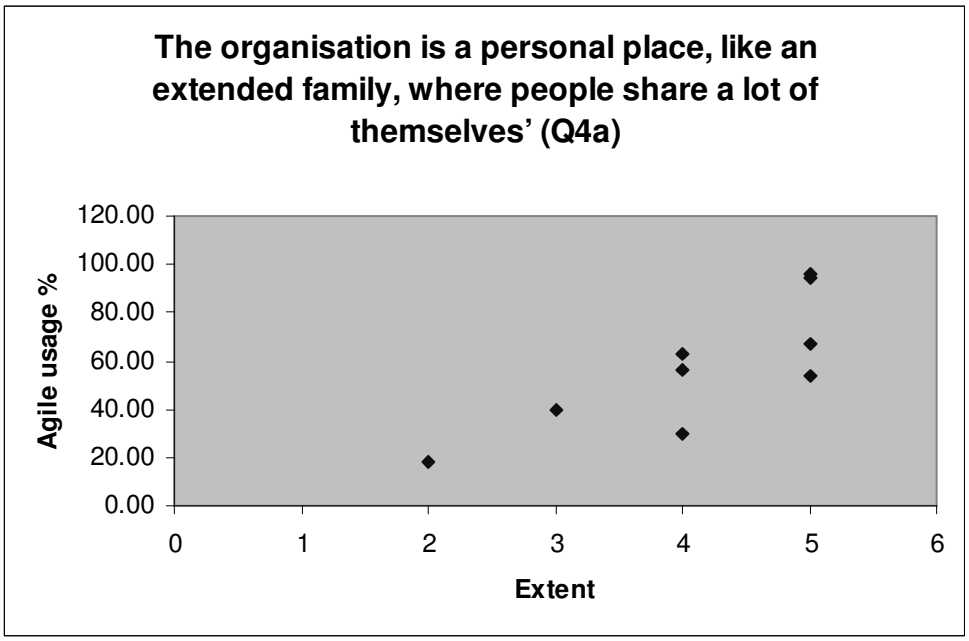

Figure 15: Scatter plot of management style is that of leadership and collaboration

These results support the model criterion.

9. The size of the organisation is large/ small

The type of organisation and the various size measurements recorded are shown in Table 26. Correlations between agility and the various size measures were investigated and the results of statistical tests on these size factors are shown in Table 27. Project Rho was excluded from some tests due to incomplete data. Where a range was provided by the respondent an average figure was used.

Table 26: Sizes recorded for project organisations and teams

Project	Alpha	Zeta	Delta	Theta	Beta	Iota	Rho	Chi	Tau
Type of organisation From interview	Private	Govt dept.	Govt dept.	Govt dept.	Private	Govt dept.	Govt dept.	Private	Private
Size of organisation (total staff) From web site or interview	100-150 Avg= 125	300	13000-14000 Avg= 13500	1400-1500 Avg= 1450	150-200 Avg= 175	600	4500-5000 Avg= 4500	300-400 Avg= 350	5-10 Avg=7.5
Number of Software development staff Q2	13	10-30 Avg= 20	50	150+	45	24	Don't know	10	3
Team size Q14,15 (added)	7	4	19	30	11	8	3	6	2

Table 27: Statistical results of tests on size measurements

	Factors measured	Technique	Test	n	rho	Sig.	Reject H_o
A	% Agile and Organisation size	Pearson's product moment correlation coefficient	2-tailed	9	-.052	None	No
B	% Agile and Number of software development staff	Pearson's product moment correlation coefficient	2-tailed	8	0.22	None	No
C	% Agile and team size	Pearson's product moment correlation coefficient	2-tailed	9	0.160	None	No
D	Number of SD staff and team size (agile projects only)	Pearson's product moment correlation coefficient	2-tailed	5	0.940	0.05 level	Yes H_o = there is no relationship between the number of SD staff and the size of a team

There is a correlation between the number of people involved in software development in an organisation and the size of the project team on an agile project (see Table 27, item D). But for organisations with lower numbers of software development staff this effect is spurious because the smaller pool of developers would limit team size. Therefore this correlation has been recorded but further evidence is needed on more projects before it can be acceptable in this study.

There is no statistical evidence to indicate that organisation size or number of staff involved in software development is correlated with the extent to which agile techniques are used. Both high usage projects were medium-sized organisations (less than 300 staff) but the project with the lowest level of agile usage was also medium sized (~200 staff).

There is no evidence to support this criterion.

10. The system uses any of: Internet application domains, shrink wrapped software, web-based systems, component delivery, component development, component assembly, client/server systems, networked systems, web-deployed applications

The results show (see Appendix L) that there is a mixture of these domains and technologies and types of development. The agile projects involved networked systems (Zeta) and web systems (Delta, Zeta, Chi) and so did the non-agile projects (Iota, Rho). Tau (non-agile) stated that their project developed shrink-wrapped software. Some of the agile projects (Alpha, Beta) did not involve either of these types. So although agile projects do use these technologies and systems there is no indication that they must use them while using an agile method. Therefore there is no evidence to show that these types of development are unique to agile methods or are more likely to occur in the high usage projects. This criterion is not supported.

11. Automated testing is used

Test-driven development (TDD) is a technique used only in the XP method and automated regression testing (ART) is unique to Crystal methods. Therefore these items are not part of the project environment so I have excluded them from the revised target environment model.

12. Object oriented technology

All of the projects used relational database technology, and all of the agile projects used object-oriented concepts and languages except Zeta (DSDM) (see Appendix L). DSDM explicitly states that the method is suitable for object-oriented or structured development. One non-agile project (Rho) did not use object-oriented concepts or languages. This shows that agile projects are not restricted to object-oriented projects

and the use of this technology is not unique to agile projects. The evidence does not support this model criterion.

13. Any of: application frameworks for external use, e-commerce and e-business, data warehouse, and products for the Internet software market

None of the projects in this study provide any of these business solutions. The evidence does not support this model criterion.

14. The domain is interface intensive systems

This is a requirement specific to DSDM. To determine if the system was interface intensive I made a subjective assessment based on the data provided for Q17 'Briefly describe the purpose of the project' and Q19 'What type of system is under development in this project'. The data is shown in Table 28. The DSDM project, Zeta and two other agile projects, Delta and Theta were interface intensive. Alpha, Beta, and one of the non-agile projects, Tau, did not involve interface intensive systems.

Table 28: Results for the domain is interface intensive systems

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q17, Q19	×	✓	✓	✓	×	×	✓	✓	✓

This evidence does not support the model criterion.

15. The domain is business problems, business systems and applications

All of the projects were creating business applications. This supports the model criterion but is not unique to agile projects.

16. The domain is non-critical projects/ critical new business initiative

Data was gathered on the importance of the project to both the organisation creating the system and to any external client. Table 29 and Table 30 show the data and results.

Table 29: Results for non-critical/critical new business initiative

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q31	3	1	1	3	3	1	1	3	1
Q32	3	N/A	1	2	3	1	2	3	N/A
1 Loss of comfort, 2 Loss of discretionary income 3 Loss of essential income 4 Loss of life									

Table 30: Statistical results for non/critical/critical new business initiative

Question	Technique	Rho	n	Test	Significance	Reject H_o
Q31	Spearman's rank correlation coefficient	0.173	9	2-tailed	none	No
Q32	Spearman's rank correlation coefficient	0.296	7	2-tailed	none	No

This is no evidence to support a decision on this criterion. This data set indicates that the criticality of the project is not correlated with the extent of agile method usage.

17. Projects that are complex

Project complexity was measured in two ways; by counting the number of tools for modelling, testing, and other aspects of development, and by asking for an indication of complexity based on given factors such as 'more than two new technologies not used before by the team'. Table 31 and Table 32 show the data and results.

Table 31: Results for projects that are complex

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q21	9 tools	2 tools	7 tools	4 tools	6 tools	4 tools	16 tools	3 tools	3 tools
Q22	2 items	1 item	4 items	3 items	4 items	2 items	2 items	1 item	2 items

Table 32: Statistical results for projects that are complex

Question	Technique	Rho	n	Test	Significance	Reject H_o
Q21	Pearson's product moment correlation coefficient	0.032	9	2-tailed	none	No
Q22	Pearson's product moment correlation coefficient	0.037	9	2-tailed	none	No

Therefore is no evidence to support this model criterion.

18. Projects with vague requirements.

This was assessed by asking how well defined the requirements were before programming began. Table 33 and Table 34 show the data and results.

Table 33: Data for projects with vague requirements

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q23	2	4	3	4	4	3	3	4	3
1 Not defined, 2 Minimal, 3 Some, 4 Adequate									

Table 34: Statistical results for projects with vague requirements

Question	Technique	Rho	n	Test	Significance	Reject H_o
Q23	Spearman's rank correlation coefficient	0.075	9	2-tailed	none	No

There is no evidence to support this model criterion.

19. Projects with constant changes in requirements

Two questions addressed this criterion; 'how many requests for new functional requirements have you received in the past two weeks', and 'how many requests for changes to existing functional requirements have you received in the past two weeks'. This caused problems for respondents reporting on completed projects as they had to estimate this from memory. Questions 24 to 27 all suffered from this design fault. Table 35 and Table 36 show the data and the results.

Table 35: Data for projects with constant changes in requirements

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q24	15+	0	0	>20	0.1	5	0	3	0
Q25	10+	0	0	0	0.2	5	0	1	0

Table 36: Statistical results for projects with constant changes in requirements

Question	Technique	Rho	n	Test	Significance	Reject H_o
Q24	Pearson's product moment correlation coefficient	0.370	9	2-tailed	none	No
Q25	Pearson's product moment correlation coefficient	0.465	9	2-tailed	none	No

There is no evidence to support this model criterion.

20. Projects undergoing constant change

Two questions assessed this criterion. Question 26 reports changes to technology requirements in a two week period and the results show that two agile projects (Alpha, 1 change and Beta 0.1 change) showed any changes. Question 27 reports changes to other project factors in a two week period (e.g. budget, staff, delivery dates). All of the agile projects showed some changes here. Table 37 and Table 38 show the data and results.

Table 37: Data for projects undergoing constant change

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q26 Q27	1 2	0 2	0 2	0 5+	0.1 0.3	0 0	0 0	0 0	0 1

Table 38: Statistical results for projects undergoing constant change

Question	Technique	Rho	n	Test	Significance	Reject H_o
Q26	Pearson's product moment correlation coefficient	0.552	9	2-tailed	none	No
Q27	Pearson's product moment correlation coefficient	0.428	9	2-tailed	none	No

A student t-Test was carried out to test the hypotheses:

H_o – there is no difference between agile and non-agile projects with respect to change.

H_1 – there is a difference between agile and non-agile projects with respect to change.

This test showed that the two groups are significantly different at the 90% level but not at the 95% level (see Table 39).

Table 39: t-test result for projects undergoing constant change

Question	Technique	t statistic	df	Test	Sig.	Reject H_o
Q27	Student t-test :Two sample assuming unequal variances Variance agile projects = 2.888 Variance non-agile projects = 0.25	2.5123	5	2-tailed	0.1 level	Yes

The result shows weak evidence that agile projects report more change than non-agile projects for *other project factors*.

21. Projects with intense time pressure.

Agile projects (4 of 5) reported that the project was under *extreme time pressure*, one selected *moderate*. The non-agile projects selected *high time pressure* and one selected *moderate*. Table 40 shows the data, Table 41 shows the results of statistical analysis. There is no statistical correlation between the time pressure and extent of agility.

Table 40: Data for projects with intense time pressure

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q29	3 Moderate	5 Extreme	5 Extreme	5 Extreme	5 Extreme	3 Moderate	4 High	4 High	4 High

1 No time pressure
2 Low
3 Moderate
4 High
5 Extreme

Table 41: Statistical results for projects with intense time pressure

Question	Technique	Rho	n	Test	Significance	Reject H_o
Q29	Spearman's rank correlation coefficient	0.241	9	2-tailed	none	No

22. Projects with teams of 2 to 10 developers

This is a requirement for each agile method; therefore this criterion is not a project environment factor so I have excluded it from the revised target environment model.

The results show (see Appendix L) that both high usage agile projects fit the desired size of 2 to 10 people but the medium usage projects have higher numbers (10 to 30 people). The three non-agile projects are all small (1-6 people). This is a technique which is not fully complied with in the projects and is more often not met in the medium usage agile projects. High usage projects complied with this requirement.

23. Documentation is minimised

Four of the agile projects were assessed on this aspect of development (see Appendix L). All said at interview that they did minimise documentation with a proviso that any documentation should be of some benefit or serve an explicit purpose for the project. This result provides evidence to support the model criterion but there is no evidence that this is unique to projects using agile methods.

24. New development or precedented systems

One agile project (Zeta) was developing a new (unprecedented or green field) system. All of the other projects, both agile and non-agile were replacement of an existing system with enhancements or outright replacement of an existing system. There is no evidence to support a decision for this model criterion.

25. Projects involving any of fix-price contract software development, in-house development, outsourced software

All of the projects carry out one or more of these types of development. Therefore there is evidence to support the model but this assumption is not unique to agile methods.

26. and 27 Projects using incremental and iterative development

These two techniques are specified in each of the five agile methods, therefore these items are not part of the project environment and I have excluded them from the revised target environment model (see Appendix L). Table 42 shows the results of iteration usage (Q43) and the reported length of iterations (Q45). There is no pattern in the results. Chi's respondent may have misinterpreted the meaning of iteration as they report conflicting results for these questions.

Table 42: Data for iterative development

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q43 Q45	3 Always 1 week	3 Always 4 week	3 Always flexible	3 Always 2 month	2 Often 1 month	3 Always 1-2 days	2 Often 1 month	2 Often 0 no iterations	0 Never
0 Never use iteration, 1 Seldom, 2 Often, 3 Always									

28. Projects where the project manager acts as a facilitator

This criterion was assessed using Question 5a (the same as for criterion 1 above). Q5a is an indirect measure. The question states 'The leadership in the organisation is mentoring, facilitating and nurturing'. These criteria are therefore amalgamated in the revised model. This criterion is amalgamated with criterion 5.

29. Project teams are collocated

This situation is specified as part of the XP, Scrum, ASD and Crystal methods. Therefore this criterion is not a project environment factor so I have excluded it from the revised target environment model.

30. Users are actively involved in the project

Customer on-site is a technique specified in the Scrum and XP methods and is less stringent and becomes *user is actively involved in project* in the methods DSDM, Crystal and ASD. This criterion is not a project environment factor so I have excluded it from the revised target environment model.

Q16 'number of end-users' and Q35 'number of face-to-face discussions carried out with the end-user in the last 2 weeks' were included in the questionnaire to triangulate or verify the extent of contact with the customer because the customer may not be on site but contacts could still be high. Table 43 and Table 44 show the data and results. These results show no relationship between the number of end users or the number of face-to-face discussions per week and the extent of agility.

Table 43: Data for users are actively involved in the project

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q16 Q35	300+ users Lots!	300 users 30 /2 weeks	7000 users 1 /2 weeks	3000 users 200+/2 weeks	50 users 1/day contact	Not known 1/day	Don't know 0/2 week	Don't know 0/2 week	100 users 1/week

Table 44: Statistical results for users are actively involved in the project

Question	Technique	Rho	n	Test	Significance	Reject H_o
Q16	Pearson's product moment correlation coefficient	0.032	6	2-tailed	none	No
Q35	Pearson's product moment correlation coefficient	0.258	7	2-tailed	none	No

31. Developers are experienced

This criterion was assessed across a number of dimensions of project leader experience; length of service in their main role, agile method use, the business domain and in software development. The number of projects that had been carried out by the organisation and the experience level of the team were also assessed. Table 45 and

Table 46 show the data and results of statistical analysis. Item A is the summed value of all experience factors (non-agile projects are excluded) and item B is the summed value of experience factors (experience in agile methods is excluded).

Q39 shows that the respondents to the questionnaire had senior roles that indicate they would be in a position to give a valid assessment of the state of a whole project. This was one of the factors specified in the case study unit of analysis. Chi had the one respondent who was not experienced. Item A: total experience, shows that for the agile projects the project leaders had a minimum of 28 years of experience. This reduces the likelihood that the responses were skewed by respondents being inexperienced and overly enthusiastic about their agile method.

Table 45: Statistical results for developers are experienced

Question	Technique	Rho	n	Test	Sig.	Reject H_o
Q33	Spearman's rank correlation coefficient	-.105	9	2-tailed	none	No
Q39	Pearson's product moment correlation coefficient	0.412	9	2-tailed	none	No
Q41	Pearson's product moment correlation coefficient	0.411	9	2-tailed	none	No
Q42	Pearson's product moment correlation coefficient	0.107	9	2-tailed	none	No
A	Pearson's product moment correlation coefficient	-.101	5	2-tailed	none	No
B	Pearson's product moment correlation coefficient	0.372	9	2-tailed	none	No

Table 46: Data for developers are experienced

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q33	3	5	2	4	4	5	2	4	5
Q38: Role	Software developer, method coach	Project manager	Project manager, analyst	Practice manager	Project director	Subproject lead, developer, tester	Method coach and mentor	Analyst Programmer Support	Programmer Analyst
Q39 main role	5 yr	15 yr	10 yr	10 yr	10 yr	3-5 yr	2 yr	Up to 6 mth	10 yr
Q40 Agile method	5 yr	10 yr	1 yr	4 yr	1 yr	n/a	n/a	Up to 6 mth	n/a
Q41 Business domain	5 yr	5 yr	20+ yr	4 yr	8 yr	never before	never before	Up to 6 mth	never before
Q42 Software development	15 yr	21 yr	15+ yr	10 yr	30 yr	9+ yr	25 yr	1 yr	20 yr
A B	30 25	51 41	46 45	28 24	49 48	14	27	2	30

	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	96	94	67XP 67Scrum	63	56	54	40	30	18
	High usage		Medium usage			Non-agile			
Q3: No. agile projects	Ongoing over 5 yrs on 2 products	100+ projects	0 projects	50+ projects	1 project	n/a	n/a	n/a	n/a
1 Inexperienced 2 Slightly experienced 3 Moderately experienced 4 Very experienced 5 Highly experienced									

In all types of project, high usage, low usage and non-agile projects, the project leaders (with the exception of the Chi respondent) had over 9 years of experience in software development and 14 years total experience. This increases the validity of the research in that eight of the nine respondents were experienced developers.

There is no evidence to support this model criterion.

Analysis of Additional Factors

Item 32 and 33 budgetary constraints and quality expected

These two factors do not form part of the target environment model for agile methods. They were included in the questionnaire as they may have influenced agile method uptake. All projects were *adequately* funded or the respondent did not know what the budgetary level was for the project. All projects reported *high* quality expectations except Theta which reported *moderate* quality expectations (see Table 47).

Table 47: Statistical results for budgetary constraints and quality level expected

Question	Technique	Rho	n	Test	Significance	Reject H_o
Q28 budget	Spearman's rank correlation coefficient	0.060	7	2-tailed	none	No
Q30 quality	Spearman's rank correlation coefficient	0.207	9	2-tailed	none	No

The results show no relationship between budgetary constraints or expected quality on how an agile method is tailored.

Corporate culture data

The section in the questionnaire on corporate culture had additional questions beyond those related to the target environment model. The raw data and results of statistical

analysis of all questions on corporate culture are listed in Appendix N and Appendix O. These questions were included because they come from a set (Paparone, 2003) of related questions which provide a comprehensive assessment of corporate culture. Table 48 is a summary of the results. The question wording is abbreviated; the full question wording is in Appendix D.

There are three corporate culture factors, which are not present in the theoretical model of the target environment for agile methods, which show statistical correlation with the extent of agility on the projects. These are question 4c, 5b and 7a. 4c: *The organisation is very results oriented. A major concern is with getting the job done. People are very competitive and achievement oriented.* Question 5b: *The leadership in this organisation is entrepreneurial, innovative, and risk taking*. Question 7a: *The glue that holds the organisation together is loyalty and mutual trust. Commitment to this organisation runs high*. These three factors are added to the revised target environment model and the wording is simplified as follows: 4c becomes: The organisation is results oriented. The wording of 5b is used as stated and 7a becomes: the organisation is based on loyalty and mutual trust and commitment.

Table 48: Summary table for organisational culture and percentage agility

Source (questionnaire number)	Question wording	Statistically Significant[#]
Q4a*	Organisation is personal place	✓
Q4b*	Organisation is dynamic and entrepreneurial	✓
Q4c	Organisation is results oriented	✓
Q4d	Organisation is controlled…	
Q5a*	Leadership is mentoring…	✓
Q5b	Leadership is entrepreneurial…	✓
Q5c	Leadership is no-nonsense…	
Q5d	Leadership is coordinated…	
Q6a*	Teamwork…	✓
Q6b	Individual risk-taking	
Q6c	Hard-driving competitiveness	
Q6d	Security of employment…	
Q7a	Glue is loyalty…	✓
Q7b	Glue is innovation…	
Q7c	Glue is achievement…	
Q7d	Glues is rules…	
Q8a*	Emphasis on human development…	✓
Q8b	Emphasis on new resources…	
Q8c	Emphasis on competitive actions…	
Q8d	Emphasis on permanence…	
Q9a*	Success is human resource development…	
Q9b	Success is newest products…	
Q9c	Success is winning…	
Q9d	Success is efficiency…	

Key
\# Using Spearman's rank correlation coefficient on all projects (N=9)
* Present in the target environment model for agile methods
Shaded rows indicate criteria that must be added to the revised target environment model

Research question 2: For what type of environment are agile methods suitable?

The results of the analysis of data gathered to support the target environment model for agile methods is summarised in Table 49. From these results a revised model was developed (see Table 50) which amalgamates certain criteria, excludes others and adds the organisational culture items.

The revised model of the target environment for agile methods shows that propositions 2, 3 and 4 are true for the cases in this study:

Proposition 2: There are specific environmental conditions that are suitable for agile methods.

Proposition 3: An agile method is not tailored when it is used in its target environment.

Proposition 4: An agile method is tailored when it is not used in its target environment.

The results show there are a number of characteristics of agile methods that are less prevalent or not present in non-agile projects. These are the organisational characteristics of feedback and learning, teamwork, empowerment of people, and a results-oriented culture that values entrepreneurship, innovation and risk taking. The project environment factor more prevalent in agile projects is constant change.

Table 49: Summary of results for the target environmental model for agile methods

Model criterion	Target environmental model for agile methods	Model supported	Unique to agile methods
1	The organisation values feedback and learning.	✓✓	A
2	The organisation values teamwork.	✓	A
3	The organisation values face-to-face communication.	✓	N
4	The organisation enables empowerment of people.	✓✓	A
5	The organisation is flexible and participative and encourages social interaction [Amalgamate with 2].	✓✓	A
6	Social interaction in the organisation is trustful, collaborative, and competent [Amalgamate with 1].	✓✓	A
7	Communication in the organisation is informal [Amalgamate with 3].	✓	N
8	The management style is that of leadership and collaboration.	✓✓	A
9	The size of the organisation is large/ small.	-	
10	Any of: Internet application domains, shrink wrapped software, web-based systems, component delivery, component development, component assembly, client/server systems, networked systems, web-deployed applications.	×	
11	Automated testing is used. [EXCLUDE]		
12	Object-oriented technology.	×	
13	Any of: application frameworks for external use, e-commerce and e-business, data warehouse, and products for the Internet software market.	×	
14	The domain is interface intensive systems	×	
15	The domain is business problems, business systems and applications	✓✓	N
16	The domain is non-critical projects/ Critical new business initiative.	×	
17	Projects that are complex.	×	
18	Projects with vague requirements	×	
19	Projects with constant changes in requirements/ Projects with stable requirements.	×	
20	Projects undergoing constant change.	✓	A
21	Projects with intense time pressure.	-	
22	Projects with teams of 2 to 10 developers. [EXCLUDE] ft - full time pt – part time		
23	Projects where documentation is minimised.	✓✓	N
24	Projects involving new development/ Precedented systems/ Projects that are not constrained by an existing computing environment.	-	
25	Projects involving any of fix-price contract software development, in-house development, outsourced software	✓✓	N
26	Projects using incremental development. [EXCLUDE]		
27	Projects using iterative development. [EXCLUDE]		
28	Projects where the project manager acts as a facilitator. [Amalgamate with 5]		
29	Projects teams are collocated. [EXCLUDE]		
30	Users are actively involved in the project	×	
31	Developers are experienced	×	

Key

✓✓ Supported
✓ Weakly supported
× Evidence to show this is not true
- Evidence is inconclusive
A More common in agile methods projects
N Not unique to agile methods projects
[EXCLUDE] this criterion is excluded in the revised model because it is a technique that forms part of one or more specific agile methods.

Table 50: Revised model of the target environment for agile methods

Criterion from original model	Revised target environmental model for agile methods	Model supported	Unique to agile methods
1	The organisation values feedback and learning. Social interaction in the organisation is trustful, collaborative, and competent.	✓✓	A
2	The organisation values teamwork is flexible and participative and encourages social interaction. The project manager acts as a facilitator	✓	A
4	The organisation enables empowerment of people.	✓✓	A
8	The management style is that of leadership and collaboration.	✓✓	A
New	The organisation is results oriented.	✓✓	A
New	The leadership in the organisation is entrepreneurial, innovative, and risk taking.	✓✓	A
New	The organisation is based on loyalty and mutual trust and commitment.	✓✓	A
20	Projects undergoing constant change.	✓	A
3	The organisation values face-to-face communication and communication in the organisation is informal.	✓✓	N
15	The domain is business problems, business systems and applications	✓✓	N
23	Documentation is minimised.	✓✓	N
25	Projects involve any of fix-price contract software development, in-house development, outsourced software	✓✓	N

Key

✓✓	Supported
✓	Weakly supported
A	More common in agile methods projects
N	Not unique to agile methods projects

Research Question 2.2 Benefits of agile methods

Research question 2.2 is 'What are the perceived benefits of agile methods'. This was restructured into two propositions;

Proposition 5: Method users perceive benefits from the use of the method which match those of the published method.

Proposition 6: There are more perceived benefits when environmental conditions suitable for agile methods are present.

These propositions were addressed in the questionnaire by providing a check list for the respondent and a space for a written response. The reasons why the method was seen as beneficial was addressed in the interview using these questions (see Appendix E for the interview schedule):

Interview question 7: You mention benefit y can you explain this further?

Interview question 9: If the project was completed was it a success?

Interview question 10: Would you work this way again?

Interview question 11: Do you have anything else to add?

The results are summarised in Table 51. The responses were coded according to the list of benefits provided in the questionnaire (see Appendix D) as follows:

A. Reduction of effort
B. Increased customer satisfaction
C. Improved quality of delivered system
D. Improved team morale
E. Faster development
F. More productive team
G. Improved communication with customers
H. Better control of development schedule

Table 51: Summary of the benefits of agile methods

Project	Method	Usage %	Benefits and comments
Alpha	XP	96	A, B, C, D, E, F, G, H System metaphor: described by respondent as a "Weird one", "hotly debated", "hard to come up with a good metaphor for our system". They substituted this technique with good communication. Additional benefits: better code quality, team is more innovative and creative because "you can act on things more quickly and change things more quickly", and the team communicates ideas more freely.
Zeta	DSDM	94	B, C, D, E, F, G
Delta	Scrum/XP	67	A, B, C, D, F, H
Theta	RUP/XP	63	D, E, F, G "Ownership of end result is driven across all team members. Requirements are validated against technical solution on an ongoing basis, and end result expectations are normally aligned before development effort is expended".
Beta	XP	56	D "Pair programming is one of the best aspects of XP" because you can get help faster, improve code quality and skill transfer – "you can't get it any other way" The floor will be rearranged soon with bench desks to facilitate this technique. The "bits to keep" from XP are: ▪ Pair programming ▪ User stories ▪ Test-first design ▪ Tester collocated with team

Analysis shows that D (improved team morale) was the single benefit common to all agile projects. The lowest usage project saw the least benefits in their chosen method and the highest usage projects saw the most benefits. This provides support for the two

propositions of research question 2.2 and for the proposed benefits listed for agile methods reported in the literature.

Research Question 2.3 Problems with agile methods

Research question 2.3 is 'What are the perceived problems with agile methods'. This was restructured into two propositions;

Proposition 7: There are problems with some techniques used in agile methods.

Proposition 8: There are more perceived problems when environmental conditions are not suitable for agile methods.

This was addressed in the questionnaire by providing a check list for the respondent and a space for a written response. The reasons why a technique was not used at the 3 or *always* level was also investigated for each agile project at the interview. The responses were coded according to the list of problems provided in the questionnaire (see Appendix D). Lack of training, the impact of team size and the impact of a traditional organisation structure was added to the list because they appeared as problems on more than one project. Results of the analysis are shown in table Table 52. The codes are as follows:

I. Reduced design documentation creates problems
J. Customer on site is difficult to achieve
K. Gaining the support of management for agile techniques is difficult
L. Some techniques are not acceptable to the team members
M. Controlling iterations is difficult
N. Requirements changes are difficult to control
O. Clients want the development to meet a fixed price contract
P. Clients can not prioritise requirements
Q. Clients are not prepared to drop minor requirements when new requirements emerge
R. Team lacked training or understanding of the technique
S. Team size had a detrimental effect
T. Existing traditional organisation structure was not changed (i.e. quality assurance team does all testing)

Table 52: Summary of the problems with agile methods

Project	Method	Usage %	Problems and comments
Alpha	XP	96	No problems were provided in the questionnaire "Need an experienced person in software development and project experience". This was a comment and not given as a problem. Any other problems: "not for us" System metaphor: "weird one", "hotly debated", "hard to come up with a good metaphor for our system". Substituted this technique with good communication.
Zeta	DSDM	94	L, T
Delta	Scrum/XP	67	I, J, L, M, R "Although this project is attempting to use *some* agile techniques we will not use all. For example, story cards were trialled and waste far too much time. It is difficult to make iterations of 2-3 weeks small enough to make a useable product". This project was very complex (made up of many smaller projects of 2-3 developers). This impacted on usage of: ▪ Automated testing ▪ Testing through each iteration
Theta	RUP/XP	63	R, T, S, Q, P "Experienced team leads are required to manage overall process and end results. Fast-changing agile environments require team personalities / make-ups that can work together as a cohesive group." System metaphor: "I was not sure what that one meant"
Beta	XP	56	I, J, L, M, O, R "Insufficient promulgation and agreement on methodology definition at start of project'. Many of the techniques were not implemented fully due to 'time pressure' i.e. refactoring, daily builds, test-first development, pair programming, simplest coded solution System metaphor "it wasn't written up it was a guiding principle between us and the customer" Main problem with adoption of XP techniques was caused by a doubling in the team size half way through the project with people untrained in XP.

This analysis shows few problems common to all projects. The system metaphor technique was problematic on three of the four XP projects. Item R (training and experience) is the common factor causing problems in uptake and success with the methods. The initial training is described in the case analysis presented above. The training on those projects with the highest usage was both initial and on-going with an experienced core of people or a single leader who was experienced both in the method and was capable of 'promulgation' of the method throughout the team. On-going training and experience was explicitly mentioned by Theta's respondent as necessary and Beta had a problem with untrained people joining the project and Delta's team underwent initial training but this was not continued at the same level as new people were added to the team.

This analysis provides support for the propositions 7 and 8 of research question 2.3. The higher usage projects had a smaller number of reported problems than the lower usage projects. The proposed problems matched the perceived problems providing support for the problems reported in the literature.

Summary of Results

This chapter provided the results for each of the research questions. First, a definition for agile methods was provided based on the results of applying the analytical framework. Secondly, a revised version of the theoretical model of the target environment for agile methods was provided based on the investigation of a number of criteria developed from both the literature review of chapter 2 and application of the analytical framework in chapter 4. The results showed that aspects of organisational culture are the most important factors influencing the high use of the techniques of an agile method. The perceived benefits and disadvantages of agile methods are summarised and this shows that the benefits drawn from the literature on agile methods are correct and the main problem with successful continuance of an agile method within an organisation is training and method experience. Chapter 6 discusses the conclusions and contributions of the results of this study.

Chapter 6 CONCLUSION

Introduction

This chapter describes the conclusions and main contributions of this study of agile methods. This is followed by a discussion of the limitations of this study and the indications for further research. Where the term 'agile methods' is used I mean the five agile methods covered in this study: DSDM, XP, Scrum, ASD and Crystal methods.

Conclusions from the Research Questions

This study was organised around two main research questions:

1. What is an agile method?
2. For what type of environment are agile methods most suitable?

By answering these questions, I have formed two main conclusions about agile methods; they are a distinct group of system development methodologies and there are distinct project environments where they are more likely to be successful.

I developed a definition of agile methods by applying an analytical framework to each of the five earliest published agile methods leading to an explicit definition of what makes up each method. This showed the common characteristics and activities which each of these agile methods share and the activities that they unanimously minimise. The shared properties are as follows. All of the methods were published in the period 1995-2002 by authors based in the USA and UK who took the perspective of the developer or project leader. The methods are all objectivist and provide technological solutions to business problems. They all provide for development in an iterative and incremental manner and manage this by active user involvement, feedback and learning, teamwork, and by empowering teams to make decisions. They all specify that a project must be carried out in a small team of 3 to 10 people and emphasise informal communication both within the team and between the team and other stakeholders. The activities that are minimised are the documentation and systematic production of modeling artifacts during development. These activities and artifacts are not central to the methods as they were with many of the structured design methodologies of the 1980s (Olle et al., 1983) and object-oriented

methodologies of the 1990s (Graham, 1991). Agile methods allow for the use of UML models and produce required documents as needed but this is only as requested by the client or as needed to describe the architecture to the team.

Application of the framework made clear that each agile method is a methodology in its own right, consisting of many of the parts of a methodology as defined by Avison and Fitzgerald (1995b) in Chapter 2. Each method has a stated philosophy, collection of procedures, techniques, and tools. The agile methods use phased development and provide techniques to use during the phases. Agile methods do enable the planning, management, evaluation, and control of iterative and incremental software development. The agile methods omit documentation aids and modelling techniques. In this respect they do not meet the definition.

Another outcome of applying the framework was to show that while sharing some characteristics the agile methods each serve a different purpose. DSDM and ASD are frameworks. This means that these methods can easily adopt techniques from other methodologies. Scrum focuses on project management and XP defines a set of explicit interacting techniques for adapting rapidly to high change environments using software engineering practices such as test-driven development and focusing on effective communication with the customer. Crystal is a methodology for selecting an effective set of techniques for each development project.

Application of the framework also clarified which combinations of agile methods are likely to be most useful. This is because the framework makes clear the goal of the method and the techniques that make up the method. DSDM and ASD offer little detail on many of the techniques they use. Scrum has explicit techniques for managing the project and XP is explicit in all of the techniques, with the exception of 'system metaphor', and is the most commonly adopted and most studied method. Proposed method combinations were described in the literature review (see chapter 2, Table 5) and all involved XP with other methods. It is clear why this is so after seeing the number and explicit nature of the techniques belonging to each method. When a method is not explicit as to how a particular activity should be carried out XP provides many suitable techniques. For example testing in DSDM, ASD, Crystal and Scrum is called for but not described. XP provides explicit mechanisms for achieving comprehensive testing.

The agile method authors do not explain many of the problems with their methods but they do state that when using XP and DSDM, problems will occur if the base environmental conditions are not met. With Crystal, problems are likely when projects become large and distributed. Actual problems reported by researchers are problems with customer-on-site, and scalability to large and distributed environments.

The second research question asked about the target environment for agile methods. To investigate the type of environment where agile methods are most suitable I carried out a case study of five projects were an agile method was in use. A further three projects were found where no development method was used and two projects used RUP (one of these used combined RUP and XP). Thus, there were four agile projects and four non-agile projects and one project used both RUP and XP together. The project leader on each project completed a questionnaire about the project environment, which techniques were used and to what extent they were used. Then I interviewed all respondents who reported using an agile method on their project. This was to investigate problems with individual method techniques and to clarify aspects of the project such as training and the type of organisation and product developed. I gathered organisation size and type data from the organisation's web site.

The questionnaire contained a list of the techniques belonging to each of the five agile methods. This was developed from the results of the analytical framework. A section on organisational culture was included in the questionnaire as the method authors often state this as an important factor in the successful adoption and continued success of their method. The project factors in the questionnaire came from the literature review and analytical framework, each of which provided a list of target environment factors for agile methods. Other more general project factors were also included such as budget, and quality. The types of business domain and technology developed were also included to provide a full picture of the project.

The projects were from government departments and government funded organisations and privately owned businesses of various sizes. In eight of the projects the respondents were experienced individuals in systems development and could give a perceptive and accurate overview and history of the project and also an in-depth analysis of the methodology used. Only one non-agile project had an inexperienced respondent.

Analysis of the questionnaire results showed that when an agile method is named as the method used on a project the actual usage may be very low. Usage may be so low as to be at almost at the same level as that of a project carrying out ad hoc development. This was found by first counting the number of techniques used belonging to the projects nominated method. Then the extent of usage of the techniques belonging to the nominated method was calculated. From this, I found that a project leader might report that they are using all of the techniques of a method implying that they are using the method fully and correctly. However, the actual extent of method usage may indicate a considerable amount of method tailoring. That is, a project reporting 100% usage of a method's techniques may be using the techniques only intermittently. An extreme example of this occurred on project Beta who reported 95% of the techniques of XP were used on the project but they only used those techniques at the 56% level of usage. Another project who reported ad hoc development used XP techniques at the 54% level of usage. There are three conclusions I draw from this. Firstly, self-reporting by an organisation of their development method may not always give a true indication of how the organisation is developing software systems. Secondly, simple counting of the techniques used may not give a true indication of how the method is used on a project. Thirdly, it is by measuring the extent of usage that an accurate profile of method use can be gained. The overall conclusion I draw from this is; when profiling the use of an agile method on a project it is necessary to gather data on the extent of techniques used as well as which techniques are used.

This has implications for surveys and studies of agile methods. If the nominated method is agile and the project is using an agile method only to the extent of an ad hoc development then are the developers using an agile method or ad hoc development? Any conclusions drawn from such a study, if it purports to be about agile methods may be spurious because the study is really of ad hoc development.

This study has found that the principle factors affecting the use of an agile method on these projects are a specific set of organisational culture factors. The more the organisation has these factors the more they use the techniques of their selected agile method. A number of organisational culture factors were assessed using non-parametric

statistical tests. The results showed that high usage of the nominated agile method is related to each of these factors. From this, I conclude that successful use of an agile method requires the following to be present. The organisation, its management and the development teams must value feedback and learning; social interaction must be trustful, collaborative, and competent. The organisation must be results oriented and based on loyalty, mutual trust, and commitment. Teamwork must be valued, and a flexible, participative, and encouraging approach to social interaction is necessary. The management style must be that of leadership and collaboration, and the project manager must act in a facilitating role with the staff empowered to make decisions. The leadership in the organisation must be entrepreneurial, innovative, and risk taking.

Care must be used in drawing these conclusions. It is possible that adoption of an agile method improves the social interaction in an organisation, probably a small organisation. So the effect of the environment on the method may not be exclusively in one direction. It may also be possible that the use of the method effects the project environment. Longitudinal studies would be needed to test this possible relationship.

There is one project factor, constant change (changes to the project environment rather than the system requirements) that is reported more on agile projects than non-agile projects. However, the statistical results for this conclusion are weak.

Other factors were observed on each of the projects, both agile and non-agile projects. Each project reported that they use informal, face-to-face communication, and that the domain was business problems, business systems, business applications, and all projects involved either fixed price contract software development or in house development or outsourced software. This study found these factors are not unique to projects using agile methods. Documentation is minimized was also a factor common to all agile projects but this was not assessed on non-agile projects. These factors may be necessary for successful use of an agile method but they are not unique to agile projects.

Another conclusion is that the number of people involved in software development within an organisation is related to the size of the teams formed. Since the size of teams is a factor in the agile methods there may be an indirect relationship between the

number of people involved in software development in the organisation and the usage of an agile method. There were not enough large software development departments in this study to draw a conclusion and this could be investigated further.

Consideration of the benefits and disadvantages of agile methods shows that improved team morale is common to all of the agile projects. High usage projects reported more benefits and fewer problems than lower usage projects. Effective on-going training caused problems on the lower usage projects and the higher usage projects all reported the presence of a committed method champion capable of effective method promulgation. The technique that four of the five agile projects reported difficulty with was 'systems metaphor'. This finding supports the literature, as discussed in Chapter 2.

This study shows that the agile methods are a distinct group of systems development methodologies, and that each method serves a different purpose. A second finding is that specific factors in the organisational culture of a project influence the successful adoption of the agile method techniques on each agile project. So when deciding to use an agile method some consideration must be given to these factors.

Contributions of this Research

This research has generated a number of contributions; the literature review, the analytical framework, the target environment model, and the instrumentation.

The literature review provides a history of systems development methodologies and describes the precursor methods and techniques that led to the development of agile methods. Agile methods consist primarily of techniques, practices, and principles that already exist in other systems development methodologies and software engineering. Combining these techniques and practices in different ways in the agile methods was done to cope with the development environment of the period from 1995 to 2005. The literature review organises and summarises the literature on agile methods, and makes it clear that XP is the most commonly used agile method and consequently most studied *in situ*. The methods Scrum and DSDM have a small audience. There is scant research into cases of use of ASD and Crystal. There are a large number of XP case studies, but

the small studies and single case descriptions make conclusions difficult to draw. Each case reports on a different aspect of XP.

Application of the analytical framework has provided a precise unbiased characterisation of each of the agile methods. I have now described five agile methods as to their philosophy, models, techniques, tools, scope, outputs, practice, and tailorability. Such a characterisation enables other researchers to carry out accurate categorisation and comparison of the agile methods with other systems and software development methodologies both existing and yet to be created. It also allows people selecting a method to understand what the method consists of without the interference of author bias. Finally, the characterisation of methods allows for individual techniques from a method to be used to fulfill a particular purpose in another context.

This application of the analytical framework has also provided a proof-of-concept of the framework developed by Avison and Fitzgerald (2003b). I adjusted their framework by amalgamating the concepts of output and product because they are similar, and by adding the concepts assumptions, values, perspective, metrics, and tailorability because they were of interest in this study and provided additional decomposition of the method. This framework provides a useful and comprehensive mechanism for assessing the content of a methodology.

Exploring the target environment for agile methods using a list of techniques against which to benchmark method use provided information on the techniques most used on all projects, techniques most used on agile projects and which techniques are problematic, such as the system metaphor in XP. This aspect of the study also supports the findings of Fitzgerald (1997) who also found that methodologies are tailored for use.

Analysing the factors in the target environment also highlighted the influence of organisational culture on agile method use. Anecdotally the culture is believed to affect agile method use (Beck, 2000; B. Boehm & Turner, 2004; Cockburn, 2003; J. Highsmith, 2002; Schwaber, 1995) and this study has provided support for these ideas. I found no evidence to support the ideas that the project factors of size, time pressure, complexity, quality, budget, domain, or technology affects the use of an agile method.

Further research is needed into the affect of these project factors before they can be discounted. Weak evidence was found to support the idea that constant project changes affect the use of an agile method.

Another contribution of applying the target environment model was the finding that when assessing method use it is better to determine the extent of usage rather than just the number of techniques used. When respondents self-nominate their method this must be checked with a usage measurement. Otherwise, any conclusions drawn about a method are likely to be spurious. This has implications for people carrying out process maturity studies and surveys of method use.

The instrumentation designed for this study is an artifact that is suitable for adaptation and reuse in studies of other agile methods and other system development methodologies.

The overall contribution of this study is to provide a mechanism for exploring individual methodologies or a group of methodologies both *in situ* and as-published. The characterisation of the most commonly used agile methods, in particular their techniques and phases, provides a resource for people using or planning to use an agile method, for people carrying out research into agile methods, and for those creating process maturity models or proposing method combinations.

The contribution of this study to the software development industry is to provide an unbiased assessment of the target environment for agile methods. That is the project, people, organisational, technology, and domain factors and what effect they have on the successful use of an agile method. Specific organisational factors were found to have the most important influence on successful adoption in the cases in this study.

The information provided by this study is important for software developers including those planning to adopt an agile method and those trying to understand the limits on their likely usage of an agile method caused by an adverse organisational culture. Although the results are not generalisable to all projects using an agile method the influence of the organisational culture factors should be seriously considered.

Limitations of this Study

When I made the decision, about which methods to include in the study a number of different methods could have been included. If I had selected different agile methods or a greater number of agile methods the study may have provided different answers.

Limitations also occur with the use of an analytical comparative framework. A framework only finds those factors for which there are explicit criteria. Missing criteria mean that there may be characteristics of a method that are not brought to light. I addressed this by selecting a framework that was based upon an analysis of a number of frameworks and that had been used before to assess a variety of different methodologies (Avison & Fitzgerald, 2003b).

This study suffers from the problems inherent in all case study work, that of drawing conclusions from a small number of cases and generalisability of the results to the wider population of agile methods projects. The number of cases observed limits this study and further investigation may change these conclusions. The conclusions are therefore only relevant for this given set of project data. In addition, this study used a qualitative design to assess a proposed target environment model with quantitative statistical analysis carried out where possible to assess criteria within the model. This meant using the weaker non-parametric statistics due to the small sample size.

When using a survey instrument to gather data the instrument should be fully tested and validated. Some of the questions in the questionnaire, although tested a number of times on individuals, were not comprehensively assessed as to their validity (did they test what they were supposed to test). It was necessary to use indirect questions to assess some factors. For example complexity, empowerment, criticality, degrees of time pressure, and quality assessment are all open to interpretation and would be better assessed with follow up interviews to gain an understanding of their impact on a project.

This study did not analyse the affect of combinations of target environment factors or possible interactions between factors. For example, complexity may consist of combinations of factors such as size, technologies used, and domains of application. Such combinations were not addressed in this study.

Another limitation of this study was the lack of triangulation. Only one person from each project was a source of data. To improve the assessment of environment factors a broader sample of respondents, especially team members, would have strengthened the study. This would have provided stronger evidence to accept or reject factors such as the affect of team experience levels, and the perception data on organisational culture, the extent of customer contact, team empowerment, and communication style.

Future research

Information is lacking about factors that cause the use of a methodology adopted in an organisation to change over time. Usage may decay, remain the same, or improve over time. Longitudinal studies would provide this type of information. As Jayaratna (1994) proposed, pre-implementation, implementation and post-implementation studies of methodology adoption are needed to fully understand a method *in situ*. This study is a starting point for such investigation of agile projects because it provides instrumentation suitable for the purpose.

To understand the relationship between methodologies and types of methodology effective meta-analysis of systems and software development methodologies is needed. This study has defined agile methods, which is a first step in such analysis, and ensures that knowledge in this area is not lost to each new generation of system developers.

Appendix A Attachment emailed to prospective respondents

Agile methods study

I am carrying out a study that investigates the effect of the project environment on how agile methods are used. This information will be used to provide guidelines on the types of projects that are best suited for agile methods. I am looking for people to participate in this study.

You may be using all of the agile techniques or just a few, (for example pair programming, prototyping, daily meetings, or incremental development) or you may be using a mixed method (such as RUP with XP, XP with Scrum). Which ever way you use agile methods, you can provide valuable information for this study. I am also interested in finding projects where agile methods are not used.

The research involves a questionnaire for the project manager, project leader, or lead developer which takes approximately ½ hour. This is followed by an interview to clarify points in the questionnaire. This interview is optional and takes ½ an hour or less. The research focuses on individual projects so you will only be commenting on one project. All data provided will be treated as confidential and the final report will not identify any individuals, projects or organisations.

Your information will contribute to an unbiased report that will be published in a Masters thesis for Massey University and in a separate technical report.

Interviews will be arranged by phone or email if you are located outside the Wellington region.

If you would like to participate in this study please contact me.

Many thanks

Diane Strode
Senior Lecturer
Faculty of Business and Information Technology
Whitireia Community Polytechnic
Wineera Drive
Private Bag 50910
Porirua City

d.strode@whitireia.ac.nz

(04) 237 3100 ext 3849

Appendix B Information sheet for respondents

An Evaluation of Agile Methods

INFORMATION SHEET

This research investigates the environment of software projects in which developers are using agile methods, in order to build up a picture of when agile methods should be used, and when they are less effective. Information is being gathered about projects and teams from a number of different organisations.

By taking part in this study you will provide valuable information for organisations who are considering using agile methods or who need to improve their methods of software development. The data will be used in a research thesis for a Master of Information Sciences undertaken at Massey University.

The results of this research will be provided to you in a report once the study is complete.

This study consists of a questionnaire and an optional follow-up interview.

This questionnaire is to be completed by a person in the organisation who works with a team of software developers as a ***supervisor, team leader or project manager*** (or has a similar role). It should take about 30 minutes to complete.

Please note

You are not obliged to complete either of these questionnaires or take part in any interviews.

If you decide to participate, you have the right to:

- decline to answer any particular question;
- withdraw from the study at any time;
- ask any questions about the study at any time during participation;
- provide information on the understanding that your name will not be used.
- be given access to a summary of the project findings once it is concluded.
- If you agree to a follow-up interview you have the right to ask for the audio recording to be turned off at any time during the interview.

You cannot be identified in the questionnaires and the raw data gathered will not be made available to your company or any other individual or organisation. You will not be identified in any research report or thesis which uses the information you provide. The information will be analysed and used to draw general conclusions and make recommendations.

The completed questionnaire should be returned to the researcher in the envelope provided, either in person or by mail

The following section describes how you were recruited, what will happen to the data, and who to contact for further information.

Participant Recruitment

Participants are selected because they are working on software development projects using agile development methods. Some participants are selected because they are using other methods. The names of project/team leaders were gathered from a number of sources, an Internet search, by personal contact or because the persons name, or the name of their organisation was published in a computing journal. Participants in this study are then recruited by email, phone call or personal contact to ask if they would be interested in participating.

A maximum of 20 project/team leaders from at least five different organisations are participating in the study in order to provide a variety of different environments where agile methods are used. The method of investigation is by questionnaire and interview and no discomfort or risk to participants is anticipated using this research methodology.

Research Procedures

The data will be used to establish how particular software development methods are used in projects from a variety of different organisations, business areas and types of technology. This will involve statistical analysis and analysis of trends in the data. The data will remain with the researcher and the raw data will not be given to any other person(s). No individuals or organisations will be identified in the final report. The data will be stored by the researcher in her home and a copy will be kept at her workplace. Both copies will be kept in locked cabinets. The raw data will be disposed of once the final report is published. The researcher will publish a summary of the results of the data, in the form of a technical report, and send this to all participants once the thesis is submitted.

Participant involvement

- Participants complete one questionnaire.
- Participants who are using agile methods may then choose to take part in one semi-structured interview of 20 to 30 minutes.

Project Contacts

If you have any queries about the study or the questions please contact me:

Diane Strode
Lecturer
Faculty of Business and Information Technology
Whitireia Community Polytechnic
Wineera Drive
Porirua City
Wellington
d.strode@whitireia.ac.nz
(04) 237 3100 ext 3849

You may also contact my supervisor:

Alexei Tretiakov
Lecturer
Information Systems Department
Massey University
Palmerston North

A.Tretiakov@massey.ac.nz
(04) 801 5799 ext 2683

Committee Approval Statement

This project has been evaluated by peer review and judged to be low risk. Consequently, it has not been reviewed by one of the University's Human Ethics Committees. The researcher(s) named above are responsible for the ethical conduct of this research.

If you have any concerns about the conduct of this research that you wish to raise with someone other than the researcher(s), please contact

Professor Sylvia Rumball

Assistant to the Vice-Chancellor (Ethics & Equity)

(06) 350 5249

email humanethicspn@massey.ac.nz

Appendix C Consent form for respondents

An Evaluation of Agile Methods

PARTICIPANT CONSENT FORM

This consent form will be held for a period of five (5) years

I have read the Information Sheet and have had the details of the study explained to me. My questions have been answered to my satisfaction, and I understand that I may ask further questions at any time.

I agree to participate in this study under the conditions set out in the Information Sheet.

If you have agreed to a follow-up interview then

I agree/do not agree to the interview being audio taped.

I wish/do not wish to have my audio files or tapes returned to me.

Signature: ..... **Date:** .....

Full Name - printed .....

Appendix D Questionnaire

Agile Methods Study: Project Questionnaire

This research investigates the environment of software projects in which developers are using agile methods, in order to build up a picture of when agile methods should be used, and when they are less effective. Information is being gathered about projects and teams from a number of different organisations.

By taking part in this study you will provide valuable information for organisations who are considering using agile methods or who need to improve their methods of software development. The data will be used in a research thesis for a Master of Information Sciences undertaken at Massey University.

This information will be analysed and provided to you in a report once the study is complete.

This questionnaire is for a person in the organisation who works with a team of IT developers as a **supervisor, team leader** or **project manager** (or has a similar job role).

The survey should take about 30 minutes to complete.

You are not obliged to complete this questionnaire.

You cannot be identified in this questionnaire and the information gathered will not be made available to your company or any other individual or organisation.

Further information is provided in the **information sheet** provided with this questionnaire.

Thank you for taking part in this study

The researcher will collect this questionnaire from you in person or you can mail it in the return envelope provided

Section 1 General factors

This section of the questionnaire gathers information about your organisation. ***Your organisation*** *means your department, section, organisational unit or major work group.*

1. What type of development does your organisation undertake?

Tick all that apply

☐ We provide specific products for external customers by:
- ☐ completing one-off contracts
- ☐ mass production of shrink-wrapped products
- ☐ customising pre-developed software
- ☐ providing service/support of software systems

☐ We provide in-house development to support the running of the organisation by:
- ☐ carrying out our own development
- ☐ customisation of brought-in products

☐ We develop software for inclusion in our organisation's products by:
- ☐ in-house development
- ☐ purchasing software from an outside source
- ☐ other, please specify

_____

2. Approximately how many staff are involved in software development in this organisation?

_____ OR if you don't know please tick this box ☐

3. Approximately how many projects has your organisation completed using an agile method?

_____ OR if you don't know please tick this box ☐

OR we are not using agile methods ☐

Section 2 Organisational culture

Circle the number that most closely describes your organisation.

*1 means your organisation is **almost never** like this*
*2 means your organisation is **seldom** like this*
*3 means your organisation is **sometimes** like this*
*4 means your organisation is **often** like this*
*5 means your organisation is **almost always** like this*

4. This organisation is a very…

	Almost never	Seldom	Some-times	Often	Almost always
a. **personal place**, it is like an extended family, people share a lot of themselves.	1	2	3	4	5
b. **dynamic and entrepreneurial** place, people are willing to stick their necks out and take risks.	1	2	3	4	5
c. **results oriented**. A major concern is with getting the job done. People are very competitive and achievement oriented.	1	2	3	4	5
d. **controlled and structured** place. Formal procedures generally govern what people do.	1	2	3	4	5

5. The leadership in this organisation is…

	Almost never	Seldom	Some-times	Often	Almost always
a. **mentoring, facilitating, and nurturing**.	1	2	3	4	5
b. **entrepreneurial, innovative**, and **risk taking**.	1	2	3	4	5
c. **no-nonsense, aggressive,** and **results-oriented.**	1	2	3	4	5
d. **coordinated, organised**, and **smooth-running.**	1	2	3	4	5

6. The management of employees in this organisation is characterised by…

	Almost never	Seldom	Some-times	Often	Almost always
a. **teamwork, consensus**, and **participation**.	1	2	3	4	5
b. **individual risk-taking, innovation, freedom**, and **uniqueness**.	1	2	3	4	5
c. **hard-driving competitiveness, high demands**, and **achievement**.	1	2	3	4	5
d. **security of employment, conformity, predictability**, and **stability in relationships**.	1	2	3	4	5

7. The glue that holds the organisation together is …

	Almost never	Seldom	Some-times	Often	Almost always
a. **loyalty** and **mutual trust**. Commitment to this organisation runs high.	1	2	3	4	5
b. commitment to **innovation and development**. There is an emphasis on being on the cutting edge.	1	2	3	4	5
c. emphasis on **achievement** and **goal accomplishment**. Aggressiveness and winning are common themes.	1	2	3	4	5
d. formal **rules** and **policies**. Maintaining a smooth-running organisation is important.	1	2	3	4	5

8. This organisation emphasises…

	Almost never	Seldom	Some-times	Often	Almost always
a. **human development**. High **trust, openness**, and **participation** persist.	1	2	3	4	5
b. **acquiring new resources** and **creating new challenges**. Trying new things and prospecting for opportunities are valued.	1	2	3	4	5
c. **competitive actions** and **achievement**. Hitting stretch targets and winning in the marketplace are dominant.	1	2	3	4	5
d. **permanence** and **stability**. Efficiency, control and smooth operations are important.	1	2	3	4	5

9. This organisation defines success on the basis of…

	Almost never	Seldom	Some-times	Often	Almost always
a. the **development of human resources**, **teamwork, employee commitmen**t, and **concern for people**.	1	2	3	4	5
b. having the most unique or the **newest products**. It is a **product leader** and **innovator**.	1	2	3	4	5
c. **winning in the marketplace** and **outpacing the competition**. **Competitive market leadership** is key.	1	2	3	4	5
d. **efficiency**. **Dependable delivery, smooth scheduling**, and **low cost production** are critical.	1	2	3	4	5

Section 3 Project description

*The rest of this survey gathers information about **a single project**. Please select **one** typical project which was recently completed or is currently underway within your organisation which is using an agile method. If you are not using an agile method choose a project that you are currently working on or have recently completed.*

10. What is the name of the project? _____

11. If your project is completed: what was the duration of the project? _____

Otherwise: what is the estimated duration the project? _____

12. What is the nature of this project?

*Select **one or more***

- ☐ new system
- ☐ enhancement to existing system
- ☐ replacement of existing system
- ☐ maintenance
- ☐ other, please specify _____

13. What criteria do you use to determine when this project is completed?

*Select **one** only*

- ☐ system is handed over to testers
- ☐ client sign off
- ☐ systems moves into maintenance phase
- ☐ system is in use with its intended user base
- ☐ other, please specify _____

14. How many **full-time** developers (programmers, analysts, testers etc.) are involved in this project?

_____

15. How many **part-time** developers (programmers, analysts, testers etc.) are involved in this project?

_____

16. Approximately how many end-users will use the system developed in this project?

_____ OR if you don't know please tick this box ☐

17. Briefly describe the purpose of the project

Examples To enable restaurant bookings to be captured on-line
To provide on-line capabilities for staff to diarise future events

_____
_____
_____
_____

18. What business area does the project application address?

*Select **one or more***

- ☐ accounting
- ☐ banking
- ☐ entertainment (other than games)
- ☐ gaming
- ☐ inventory
- ☐ marketing
- ☐ management information
- ☐ office automation
- ☐ purchasing
- ☐ sales
- ☐ systems software
- ☐ scientific
- ☐ engineering
- ☐ other, please specify _____

19. What type of system is under development in this project?

*Select **one or more***

- ☐ decision support system
- ☐ management information system
- ☐ office automation system
- ☐ transaction processing
- ☐ systems software
- ☐ intranet system
- ☐ Internet system
- ☐ operating system
- ☐ extranet system
- ☐ other, please specify _____

20. Which of the following are currently used in the final application created in this project

*Select **one or more***

- ☐ gaming software engines
- ☐ interface (console)
- ☐ interface (web)
- ☐ interface (windows)
- ☐ mainframe development
- ☐ middleware
- ☐ mobile devices

☐ object database
☐ object-oriented concepts
☐ object-oriented languages
☐ relational database
☐ web development (dynamic)
☐ web development (static)
☐ other, please specify _____

Section 4 Project complexity

21. Please name the tools used in this project

If you use more than one tool in each category just name the main two tools used

Case/modelling tools _____

Unit testing tools _____

System testing tools _____

Programming languages _____

Other tools _____

22. Indicate the complexity of this project

*Select **all that apply***

☐ The project involves more than two new technologies not used before by the team
☐ The project involves a business domain with which most of the team is unfamiliar
☐ The project involves integration of more than two applications
☐ The project involves a non-standard solution with which the team is not familiar

If there are other factors that make this project complex, please describe them…

_____

Section 5 Project rate of change

23. How well defined were the requirements (business and user requirements) when programming began at the start of the project?

- ☐ **Not defined**; there were no stated requirements
- ☐ **Minimal** requirements were provided; there was no detail
- ☐ **Some requirements** were provided; but not enough to carry out the whole project
- ☐ **Adequate requirements** were provided; they were adequate to carry out most of the project
- ☐ Requirements were **fully defined** before the software development began

When you answer these questions include both formal documented requirements changes and informal changes

24. How many requests for **new functional requirements** have you received in the past two weeks? ____

    Examples: *a feature that involves a new calculation, a feature that allows the client to self-administer a web site*, *a feature to transfer taxation data to an external server*

25. How many requests for **changes to existing functional requirements** have you received in the past two weeks? ____

    Examples: *change the corporate logo on all web pages*, *change the calculation of rates on a payroll application for specific locations*

26. How many requests for **changes to technology requirements** have you received in the past two weeks? ____

    Examples: *new version of the development language will be used, a different database will be used*

27. How many requests for **changes to other project factors** have you received in the past two weeks? ____

    Examples: delivery date changes, budgetary changes, staff changes

Section 6 Project constraints

28. Is the budget for this project adequate to achieve its main business requirements?

- ☐ Barely adequate
- ☐ Adequate
- ☐ Generous
- ☐ There are no budgetary constraints on this project
- ☐ I don't know

29. How much time pressure do you feel this project is currently under?

- ☐ No time pressure
- ☐ Low time pressure
- ☐ Moderate time pressure
- ☐ High time pressure
- ☐ Extreme time pressure

30. What is the expected level of quality of the software developed?

- ☐ No quality level was specified
- ☐ Low level of quality was expected
- ☐ Moderate level of quality
- ☐ High quality
- ☐ The highest quality must be provided (i.e. loss of life will be caused if this software fails)

Section 7 Project criticality

31. How critical is the success of this project to **your organisation**? Defects showing up in operation will cause:

*Select **one** only*

- ☐ Loss of comfort *e.g. the developers will be disappointed or embarrassed by defects, or the end users will carry out more manual data entry*
- ☐ Loss of discretionary income *e.g. some invoice calculations will be incorrect*
- ☐ Loss of essential income *e.g. the Internet trading system is so slow that there are few new customers*
- ☐ Loss of life

32. Do you have a **client organisation** for this project?

- ☐ No, if you answered **no** please jump to question 33
- ☐ Yes, if you answered **yes** then how critical is the success of this project to the **client** organisation? Defects showing up in operation will cause:

*Select **one** only*

- ☐ Loss of comfort *e.g. the end users will carry out more manual data entry*
- ☐ Loss of discretionary income *e.g. the client will not be able to provide their customers with a complete banking service*
- ☐ Loss of essential income *e.g. the client will lose a substantial business advantage*
- ☐ Loss of life

Section 8 Project team factors

33. Estimate the experience level of the team on the project

Experience is a mix of programming experience and experience in the business area of the project

- ☐ The team is **inexperienced** (i.e. more than half the team has one years experience or less)
- ☐ The team is **slightly experienced** (i.e. more than half of the team has at least 2 years experience)
- ☐ The team is **moderately experienced** (i.e. more than half of the team has at least 3 years experience)
- ☐ The team is **very experienced** (i.e. more than half of the team has at least 4 years experience
- ☐ The team is **highly experienced** (i.e. more than half of the team has 5 years experience or more)

34. Are team members actively involved in making decisions, formally or informally, about how to deal with problems arising within the project?

- ☐ Almost never
- ☐ Seldom
- ☐ Sometimes
- ☐ Often
- ☐ Almost always

35. How many face-to-face discussions, either informal or formal, has the whole team, or any single member of the team had with the customer or user representative in the previous 2 weeks? ____ *(state how many contacts)*

36. The communication style most commonly used **between the team members** in the current project is…

- ☐ **Completely formal,** almost all communication is by **formal document**, formal meetings or by formal email
- ☐ **Mainly formal**, communication tends to formal with some informal interaction
- ☐ **A balanced mix** of formal and informal communication
- ☐ **Mainly informal**, communication tends to be informal with some formal interaction
- ☐ **Informal,** almost all communication is **face-to-face**

37. The communication style most commonly used **between team members and management** in the current project is…

- ☐ **Completely formal,** almost all communication is by **formal document**, formal meetings or by formal email
- ☐ **Mainly formal**, communication tends to formal with some informal interaction
- ☐ **A balanced mix** of formal and informal communication
- ☐ **Mainly informal**, communication tends to be informal with some formal interaction
- ☐ **Informal,** almost all communication is **face-to-face**

Section 9 Project leader experience

38. Write down the main role or roles you currently have in this project.
e.g. Team leader, Project manager, Programmer, Analyst, Tester, Customer (advisor on business requirements)

_____

_____

_____

_____

_____

39. How long have you carried out your **main role**?

- ☐ Never before
- ☐ Up to 6 months
- ☐ Up to 1 year
- ☐ Up to ____ years *(state how many years)*

40. How long have you been using this particular agile method or technique?
For example: Extreme programming, Scrum, Agile methods, prototyping

- ☐ We don't use an agile method
- ☐ Never before
- ☐ Up to 6 months
- ☐ Up to 1 year
- ☐ Up to ____ years *(state how many years)*

41. How much experience do you have in the **business domain** of this project?
For example: accounting systems, insurance systems, diary systems, advertising systems

- ☐ Never before
- ☐ Up to 6 months
- ☐ Up to 1 year
- ☐ Up to ____ years *(state how many years)*

42. How many years have you worked in **software development**?

- ☐ Never before
- ☐ Up to 6 months
- ☐ Up to 1 year
- ☐ Up to ____ years *(state how many years)*

Section 10 Agile method usage

43. Which of the following techniques do you use in this project?

Please circle the number to show that you use a technique and how frequently you use it

Technique	Always	Often	Seldom	Never
Concurrent development (analysis, design, code, and test carried out simultaneously)	3	2	1	0
Iterative development	3	2	1	0
Time boxing (iterations of set length)	3	2	1	0
Incremental development (small releases)	3	2	1	0
Evolutionary prototyping	3	2	1	0
Small releases of software product	3	2	1	0
Component development (sets of features are developed concurrently)	3	2	1	0
Test first development	3	2	1	0
Daily builds of complete system	3	2	1	0
Automated regression testing	3	2	1	0
Refactoring of code	3	2	1	0
Testing throughout each iteration (including unit, integration and acceptance testing)	3	2	1	0
Software inspections	3	2	1	0
Customer on-site	3	2	1	0
Method coach on site	3	2	1	0
Tester(s) collocated with team	3	2	1	0
Customer focus groups	3	2	1	0
Rooms organised for pair programming	3	2	1	0
Whole team works in same office or floor	3	2	1	0
Dedicated meeting space	3	2	1	0
Pair programming	3	2	1	0
Coding to an agreed standard	3	2	1	0
Collective ownership of code	3	2	1	0
40 hour week	3	2	1	0
Sprint Goal	3	2	1	0
Daily team meetings	3	2	1	0
Iteration planning meeting	3	2	1	0
Planning game (meeting with stakeholders at start of each iteration to plan scope)	3	2	1	0
Reflective workshops for methodology adaptation at each iteration	3	2	1	0
User stories	3	2	1	0

Technique	Always	Often	Seldom	Never
System metaphor developed	3	2	1	0
Only what has direct business value is developed	3	2	1	0
Requirements are prioritised with the customer or user	3	2	1	0
Changes to requirements are negotiated with the users to maintain time frames	3	2	1	0
Joint Application Development (JAD) sessions (requirements gathering sessions with selected stakeholders)	3	2	1	0
Design is kept as simple as possible	3	2	1	0
Coded solution is kept as simple as possible	3	2	1	0
Risk assessment at each iteration	3	2	1	0
Product Backlog	3	2	1	0
Sprint Backlog	3	2	1	0
Release Backlog	3	2	1	0
Milestones to track progress	3	2	1	0
Product Backlog Graph metric	3	2	1	0
Sprint Backlog Graph metric	3	2	1	0
Function point counts	3	2	1	0
Please add any other agile techniques you use -				
1.	3	2	1	
2.	3	2	1	
3.	3	2	1	
4.	3	2	1	
5.	3	2	1	

44. Which agile method did you use in this project?

You may select more than one

☐ We do not use an agile method
☐ Generic agile method
☐ Extreme Programming (XP)
☐ Scrum
☐ Other, please specify _____

45. What is the typical length of iterations in this project?

☐ No iterations
☐ 1 week
☐ 2 week
☐ 1 month
☐ Flexible length
☐ Other, please specify _____

46. We created a unique methodology at the start of the project yes / no

47. We tailored an existing methodology at the start of the project yes / no

48. We will carry out a project post mortem yes / no

49. A feasibility study has been carried out yes / no

50. A business study has been carried out yes / no

51. A resource requirements analysis has been carried out yes / no

Section 11 Opinion

This section is only for respondents who are using agile methods

52. In your opinion what are the benefits of using an agile method **in this project**?

Please select as many as you wish

- ☐ Reduction of effort
- ☐ Increased customer satisfaction
- ☐ Improved quality of delivered system
- ☐ Improved team morale
- ☐ Faster development
- ☐ More productive team
- ☐ Improved communication with customers
- ☐ Better control of development schedule
- ☐ Other, please describe:

53. In your opinion what are the disadvantages of using an agile method **in this project**?

Please select as many as you wish

- ☐ Reduced design documentation creates problems
- ☐ Customer on site is difficult to achieve
- ☐ Gaining the support of management for agile techniques is difficult
- ☐ Some techniques are not acceptable to the team members
- ☐ Controlling iterations is difficult
- ☐ Requirements changes are difficult to control
- ☐ Clients want the development to meet a fixed price contract
- ☐ Clients can not prioritise requirements
- ☐ Clients are not prepared to drop minor requirements when new requirements emerge
- ☐ Other, please describe:

If you are currently using agile methods I would like to interview you about your use of the techniques. This interview takes about 20 minutes and is carried out by email, by phone or in person. The interview will improve the strength of the study findings.

I agree to a follow-up interview Yes / No

Thank you for taking part in this study

The researcher will collect this questionnaire from you in person or
If you can mail it in the return envelope provided

Appendix E Interview sheet

Agile methods study: Interview Sheet

1. Why did you decide to start using agile methods?
   Prompt
   Organisational factors
   Technical factors
   People factors
   Project factors

2. How did you learn how to use the agile method you use now?
   Prompt
   External training
   Internal training
   Study at an academic institute
   Self-taught – source?
   Taught by an on site instructor
   Taught by others in the team
   Taught by the project leader

3. How long have you been using agile methods in your section/department/work group?

4. You use technique x often/seldom/never can you tell me the reasons for this?
   Prompt
   Organisational factors
   Technical factors
   People factors
   Project factors

5. Do you believe that your team minimizes documentation during development?

6. Do you use any metrics to assess progress?

7. You mention benefit y can you explain this further?

8. You mention disadvantage z can you explain this further?

9. If project completed
   Was the project a success?

   If the project is incomplete
   Is the project a success so far?

10. Would you work this way again?

11. Do you have anything else to add?

Appendix F Theoretical model of the target environment

Table 53 shows the target environment conditions for agile methods developed from the literature review (see Chapter 2). Table 54 shows the target environment conditions for agile methods that were developed from the application of the analytical framework to five agile methods. The original data is provided in Appendix K Application of the analytical framework. This data was then summarised and is shown in Appendix J

Summary of results from analytical framework. From studying this summary data Table 54 was developed. Table 55 is an amalgamation of the data in Table 53 and Table 54. Items which cannot be directly assessed are removed.

Table 53: Target environment factors from the literature review

Organisation factors	Social interaction is trustful
	Social interaction is collaborative
	Social interaction is competent
	Face-to-face communication
	Communication is informal
	Organisation enables empowerment of people
	Management style of leadership and collaboration
	Organisation is flexible and participative and encourages social interaction
Domain factors	Internet application domains
Technological factors	Automated testing
	Object-oriented technology
Project factors	High time pressure
	Unstable requirements
	Stable requirements
	Precedented systems
	Well-established architectures
	New development
	Qualitative project control
	Non-critical project
	Monthly increments
	Small teams
	Collocated teams
	Project manager acts as a facilitator
	Product is of high value to the customer
People factors	CRACK customers
	Experienced developers
	On-site usage experts
	Teams capable of self-organising

Table 54: Target environment factors from the analytical framework

Organisation factors	Feedback and learning
	Teamwork
	Empowered teams
	Size of organisation – large or small
Technological factors	Application frameworks for external use
	Shrink-wrapped software
	Application frameworks designed for reuse
	Software applications for web deployed wireless technologies
	Web-based systems
	Object-oriented systems
	*C*omponent development or component assembly
	Client/server, networked
	Software deployed using the Web
	Fully automated regression tests (unit and/or functional tests)
Domain factors	Interface intensive systems
	Computationally straightforward (not complex) problems
	Large applications that can be meaningfully split into smaller functional components
	E-commerce and e-business, data warehouse, and products for the Internet software market
	Business problems, business systems and applications
Project factors	The project is time constrained.
	Projects that are complex
	Projects with vague requirements
	Projects with constant changes in requirements
	A critical new business initiative
	Intense time pressure
	Small to medium-sized projects (ranging up to 10,000 function points)
	Teams of less than 10 developers
	The requirements are flexible and specified at a high level.
	Projects that are not constrained by an existing computing environment
	Outsourced software
	Fix-price contract software development
	In-house development
	Incremental development
	Iterative development with 1 month iterations optimal
	Projects undergoing constant change
	Frequent meetings, daily is optimal
	Working software is the main product of development
	Minimise documentation
	Large projects must be able to be split into functional components
People factors	Active user involvement
	Communication between all stakeholders is critical
	All classes of end users can be identified.
	Onsite usage experts available at all times to provide feedback to the developers
	Experienced developers

Table 55: Theoretical model of the target environment for agile methods

Factors	Source	Item number	Model criteria
Organisation	Q5a, Q8a	1	The organisation values feedback and learning.
	Q6a, Q9a	2	The organisation values teamwork.
	Q35,36,37	3	The organisation values face-to-face communication.
	Q4b	4	The organisation enables empowerment of people.
	Q6a	5	The organisation is flexible and participative and encourages social interaction.
	Q8a	6	Social interaction in the organisation is trustful, collaborative, and competent.
	Q35,36,37	7	Communication in the organisation is informal.
	Q4a, 5a	8	The management style is that of leadership and collaboration.
	Q2,I,W	9	The size of the organisation is large/ **small.**
Technology	Q19	10	Any of: Internet application domains, shrink wrapped software, web-based systems, component delivery, component development, component assembly, client/server systems, networked systems, web-deployed applications
	Q43	11	Automated testing is used.
	Q20	12	Object-oriented technology.
Domain	Q17, 19	13	Any of: application frameworks for external use, e-commerce and e-business, data warehouse, and products for the Internet software market
	Q17, 19	14	The domain is interface intensive systems.
	Q17, 18, W	15	The domain is business problems, business systems and applications.
	Q31, 32	16	The domain is **non-critical projects**/ Critical new business initiative.
Project	Q21,22	17	Projects that are complex.
	Q23	18	Projects with vague requirements.
	Q24,25	19	**Projects with constant changes in requirements**/ Projects with stable requirements.
	Q26,27	20	Projects undergoing constant change.
	Q29	21	Projects with intense time pressure.
	Q14,15	22	Projects with teams of 2 to 10 developers.
	I5	23	Projects where documentation is minimized.
	Q12	24	Projects involving **new development**/ Precedented systems/ Projects that are not constrained by an existing computing environment.
	Q1	25	Projects involving any of fix-price contract software development, in-house development, outsourced software,
	Q43	26	Projects using incremental development.
	Q43,48	27	Projects using iterative development.
	Q5a	28	Projects where the project manager acts as a facilitator.
	Q43	29	Projects teams are collocated.
People	Q16,35,43	30	Users are actively involved in the project.
	Q33	31	Developers are experienced.

Key

1. Items in **bold** are contradicted in the literature on agile methods.
2. Q source of information is questionnaire
3. I source of information is interview
4. W source of information is website

Appendix G Agile methods and techniques by method

This information is taken from the summary tables that compare the methods in Appendix J. The lists show all of the techniques that are associated with each agile method. These techniques appear in the Project Questionnaire after amalgamation of the lists. Items marked * are not used in the questionnaire.

DSDM techniques/practices/metrics

Evolutionary prototyping
Timeboxing
Iteration of phases
Incremental development
40 hour week
Feasibility study
Business study
Regular meetings (preferably daily)
Develop only what has direct business value
MoSCow rules used to prioritise requirements
Negotiation of requirements to maintain timebox limits
JAD sessions
Testing – unit, integration, system, acceptance, regression

XP techniques/practices/metrics

Pair programming
Planning game
Simple design
Refactoring
Small releases
Iterative
Incremental development
1-4 week iterations
Coding standards
*Collective ownership
Continuous integration
Test first development
40 hour week
Metaphor
On-site customer
Metrics
Room arrangements
User stories

Scrum techniques/practices/metrics

Product Backlog
Sprint
Sprint Goal
Sprint Backlog
Sprint Planning meeting
Daily Scrum
Sprint Review meeting
Release Backlog:
Customer on-site
Work space configuration
Daily builds and tests
Testing (all types)
Metrics:

- Product Backlog Graph
- Sprint Backlog Graph

ASD techniques/practices/metrics

Techniques are designed to support collaboration and learning and manage iterative development
Component development – feature sets are assigned to components which are developed concurrently
Function point counts
Timeboxing
Prototyping
Software inspections
JAD
Customer focus groups
Post mortems (= reflective workshop)
Risk assessment
Resource requirements analysis
Team collocated
Frequent meetings
Dedicated meeting space
*Any other RAD techniques

Crystal techniques/practices/metrics

Concurrent development
Iterative development
Incremental development
Methodology creation at start up
Methodology adaptation at startup if using a crystal method
Reflective workshops for methodology adaptation
Team in one room
Milestones to track progress
Automated regression testing
Direct user involvement
2 user viewings per release
Concurrent phases
Methodology tuning workshops during increments = reflective workshops

Appendix H Agile method techniques as shown in questionnaire

Table 56 lists the techniques as they appear in the questionnaire and the method they belong to.

Table 56: Techniques used in questionnaire and the method they belong to

	Technique	DSDM	XP	Scrum	ASD	Crystal
1	Concurrent development					✓
2	Iterative development	✓	✓	✓	✓	✓
3	Time boxing (iterations of set length)	✓		✓	✓	✓
4	Incremental development	✓	✓	✓	✓	✓
5	Evolutionary prototyping	✓			✓	
6	Small releases of software product		✓			
7	Component development				✓	
8	Test first development		✓			
9	Daily builds of complete system			✓		
10	Automated regression testing					✓
11	Refactoring of code		✓			
12	Testing throughout each iteration	✓	✓	✓		
13	Software inspections				✓	
14	Customer on-site		✓	✓		
15	Method coach on site					
16	Tester(s) collocated with team					
17	Customer focus groups				✓	
18	Rooms organised for pair programming		✓			
19	Whole team works in same office or floor		✓	✓	✓	✓
20	Dedicated meeting space				✓	
21	Pair programming		✓			
22	Coding to an agreed standard		✓			
23	Collective ownership of code		✓			
24	40 hour week	✓	✓			
25	Sprint Goal					
26	Daily team meetings	✓		✓		
27	Iteration planning meeting		✓			
28	Planning game		✓			
29	Reflective workshops					✓
30	User stories		✓			
31	System metaphor developed		✓			
32	Only what has direct business value is developed	✓				
33	Requirements are prioritised with the customer or user	✓				
34	Changes to requirements are negotiated with the users to maintain time frames	✓				
35	Joint Application Development (JAD) sessions	✓			✓	
36	Design is kept as simple as possible		✓			
37	Coded solution is kept as simple as possible		✓			

	Technique	DSDM	XP	Scrum	ASD	Crystal
38	Risk assessment at each iteration				✓	
39	Product Backlog			✓		
40	Sprint Backlog			✓		
41	Release Backlog			✓		
42	Milestones to track progress					✓
	Product Backlog Graph metric			✓		
	Sprint Backlog Graph metric			✓		
	Function point counts				✓	
	Project post mortem	✓				
	Feasibility study	✓				
	Business study	✓				
	Resource requirements analysis	✓				
	MOSCOW rules	✓				
	CRYSTAL techniques					
	Unique methodology at start					✓
	Tailored existing methodology at start					✓
	TOTAL (number of techniques)	16	19	13	12	10

Appendix I Common characteristics of agile methods

This information is developed from the analytical framework. The characteristics were selected by assessing the findings from each of the framework criteria. Table 57 shows the common characteristics of the agile methods.

Table 57: Common characteristics of agile methods

	Characteristic	DSDM	XP	Scrum	ASD	Crystal
Philosophy	*Objectivist	+	+	+	+	+
	Improve productivity	-	-	+	-	+
Assumptions	*Iterative development	+	+	+	+	+
	*Incremental development	+	+	+	+.	+
	Concurrent development	-	-	-	+	+
	Early delivery	+	-	+	-	-
	Frequent delivery	+	+	+	-	-
	Most valuable features	+	+	-	-	-
	*Active user involvement	+	+	+	+	+
	Customer on-site	+	+	[9]	-	+
	*Team work valued	+	+	+.	+	+
	*Team empowerment	+	+	+	+	+
	Multi-functional teams	+	+	+	-	-
	Trust and respect	-	-	+	+	+
	*Learning/feedback	+	+	+	+	+
	Testing throughout lifecycle	+	+	+	-	+
	Framework	+	-	-	+	-
	Emergent properties	-	+	+	+	-
Perspective		[10]	[11]	[12]	[13]	[14]
Objectives	Shorten development times	+	+	-	-	+
	Agreed level of quality	+	+	-	+	-
	Uses RAD techniques	+	-	-	+	-
	*Communication	+	+	+	+	+
	Collaboration	+	-	-	+	-
	Address project risk	+	+	-	+	-
	Fitness for business purpose	+	-	--	-	-
Domain	*Solving business problems	+	+	+	+	+
Target	*Small projects (team size)	2-6[15]	2-10	5-9	1-10	1-40[16]
	Object technology	-	+	+	+	+
	Internet technology	-	+	+	+	+
	High change environments	[17]	+	+	+	+[18]
	Time constrained projects	+	-	-	+	-
Model	*None proscribed	+	+	+	+	+
Techniques	Evolutionary prototyping	+	-	-	+	-
	Prototypes	-	-	-	+	-
	Timeboxing	+	-	-	+	-

[9] Recommended not mandated
[10] Balanced between users, developers and project managers, quality assurance personnel
[11] Developer
[12] Project manager , developer
[13] Project manager
[14] Project manager, developer
[15] Any size, but large projects must be divisible into functional components
[16] Preferred size 1-10
[17] Flexible requirements
[18] Assumed since this method uses techniques from XP, Scrum, ASD, DSDM

	Characteristic	DSDM	XP	Scrum	ASD	Crystal
	Iteration length	1wk-3mth	1-4wk	1mth	1-2mth[19]	1-4 mth
	40 hour week	+	+	-	-	-
	Team co-located	-	+	+	+	+
	Rooms arranged	-	+	+	-	+
	Daily meetings	+	+	+	[20]	[21]
	JAD sessions	+	-	-	+	-
	Reflective workshop	-	-	+	+[22]	+
	Prioritise requirements	+	+	+	-	-
	Negotiate	[23]	[24]	[25]	[26]	+
	Testing – unit, integration, systems, acceptance and regression	+	+	+	-	+
	Automated unit tests	-	+	+	-	+
	Automated regression tests	-	-	+	-	+
	Daily build	-	+	+	+	-
	Metric	+	+	+	+	-[27]
	Simplicity of code/design	+	+	-	-	-
	Iteration planning	+	+	+	-	+
	RAD techniques	+	-	-	-	-
Tools						
Outputs	*Minimize documentation	+	+	+	+	+
	*Working software	+	+	+	+	+
Practice	*Practitioner based	+	+	+	+	+
	Suitable for CMM or similar	+	-	-	-	-
	Skill level of developers	-	[28]	-	[29]	[30]
	Skill level of user	-	+	-	-	-
Tailorability	Tailor to suit project	+[31]	+	[32]	+	+

Key

+	This technique, practice, characteristic or philosophy is explicitly mentioned in the published method
-	Technique, practice, characteristic or philosophy is not mentioned in the method or is considered of little importance
*	A common factor in all five agile methods
	A common factor in four agile methods

Unique characteristics identified:

- Pair programming (XP)
- Test first development (XP)
- System metaphor (XP)

[19] Projects < 9mth iteration 1-2 mth, project >9mth iteration 6-10 weeks

[20] Frequent

[21] Techniques for XP, scrum, ASD all can be used which include frequent meetings

[22] Post mortem, quality review

[23] Time, scope, cost

[24] Time, scope, cost, quality

[25] Time, features, cost, quality

[26] Schedule, scope, resource, defect

[27] Recording what is perceived to work is substituted for measurement

[28] Average

[29] High

[30] Experienced developers

[31] Tailor the project but adhere to principles

[32] No tailoring needed, scales to large projects of any size

Appendix J Summary of results from analytical framework

Table 58 summarises the results of application of the analytical framework to five agile methods.

Table 58: Summary of analytical framework results

Summary Comparison of Agile Methods					
Criteria	**DSDM**	**XP**	**Scrum**	**ASD**	**Crystal**
First published	1995	1999	1997	1995	2002
Major publication	1997	2000	2000	2002	2002
Country	UK	USA	USA	USA	USA/Europe
Philosophy: Paradigm	Objectivist	Objectivist with emergent properties	Objectivist with emergent properties	Objectivist, based on complex adaptive systems theory, emergent properties	Objectivist

Summary Comparison of Agile Methods					
Criteria	**DSDM**	**XP**	**Scrum**	**ASD**	**Crystal**
Philosophy: Assumptions and values	Development is a team effort The method is about people not techniques or tools Active user involvement is imperative Teams must be empowered to make decisions Deliverables must be fit for their business purpose All changes during development are reversible Initial requirements are base-lined at a high level of abstraction Testing throughout the life cycle Collaboration and cooperation between stakeholder and developers is essential	Values are: communication, simple solutions, constant feedback, courage Project variables are: cost, time, quality, scope <u>Assumptions</u> Cost-of-change no longer valid Change is constant during a project <u>Principles</u> Rapid feedback, assume simplicity, incremental change, embracing change, quality work, teaching learning, small initial investment, play to win, concrete experiments, open, honest communication, work with peoples instincts, accepted responsibility, local adaptation, travel light, honest measurement	Empirical process control <u>Assumptions</u> Software creation is like new product development The process is unstable Teams are self-organising Development processes are overlapping 'Multilearning' is needed Subtle process control is needed Organisational transfer of learning occurs Team qualities: focus, openness, respect, courage Project variables are: cost, time, quality, functionality Control is by adjusting functionality	Combines RAD ideas and complex adaptive systems theory Suitable for high change, time pressure environments <u>Assumptions</u> Projects act like complex adaptive systems and show emergent behaviour Based on 3 models: 1. Adaptive conceptual model 2. Adaptive Development Model 3. Adaptive Management Model Cultural characteristics: • Emergent order • Simple principles • Rich connections • Distributed governance • Poise • Balance Values: Quality leadership Encourage innovation Mutual trust and respect Collaboration	Communication critical Methodology design parameters: size, ceremony, weight, problem size, project size, system criticality, precision, accuracy, relevance, tolerance, visibility, scale, stability <u>Assumptions</u> Face-to-face communication is best Methodology weight is costly Larger teams need heavier methodologies Greater criticality needs greater ceremony Communication and feedback reduces intermediate deliverables Discipline, skills understanding can substitute for process, formality and documentation Focus efficiency at process bottlenecks
Philosophy: Perspective	Balanced between users, developers, managers	Programmer centred	Project manager and programmer centred	Project manager centred	Project manager and developers

Summary Comparison of Agile Methods					
Criteria	**DSDM**	**XP**	**Scrum**	**ASD**	**Crystal**
Philosophy: Objectives	A framework to guide use of RAD tools and techniques Only technological solutions are considered <u>Aims</u> To provide early and frequent delivery of product To deliver only the features most valuable to the business To shorten development times To deliver greatest business value first To delivery only what is needed, on time and at an agreed level of maintainability (quality) To remove the 'quick and dirty' image of RAD To improve communication within the development team (which includes users) To provide technically robust systems	Only technological solutions are considered <u>Aims</u> To reduce development time, respond effectively to business and technology changes, to maintain and improve competitiveness, improve team productivity, address project risk, make software development fun, have a better relationship with customers, have a stable programmer team with high morale and a good working environment, produce software at some negotiated level of quality	Only technological solutions are considered <u>Aims</u> To provide techniques for project management, to support self-empowered teams, to give a global view of development, to produce complex, sophisticated software that meet s business needs, to provide productivity gains, to produce working functionality within one month and in monthly increments thereafter, to act as a wrapper for other software engineering practices, to deliver an increment of software by controlling functionality	Only technological solutions are considered <u>Aims</u> To provide a framework for managing software development projects which are under high time pressure and have rapidly changing requirements To deliver software to the client within designated scope, schedule, resource, and defect levels To provide software of 'good enough' quality To minimize documentation To produce maintainable, change-tolerant software	Only technological solutions are considered <u>Aims</u> To create unique methodologies for each project, to tailor methodologies effectively during development, to produce working software incrementally, to produce software for business problems, to set up for the next project by providing appropriate levels of documentation, to have a satisfied productive development team

Summary Comparison of Agile Methods					
Criteria	**DSDM**	**XP**	**Scrum**	**ASD**	**Crystal**
Philosophy: Domain	Business problems that are computationally straightforward	Problem-solving methodology for well-defined problems	Problem-solving methodology for well-defined problems	Problem-solving methodology for well-defined problems	Problem-solving methodology for well-defined problems
Philosophy: Target	Interface intensive business systems Small projects or large projects that can be divided Projects where users can be clearly identified Computationally straightforward systems Time constrained projects Projects where requirements are flexible and are specified at a high level	New projects Projects with vague and/or changing requirements Small projects with 2-10 programmers Object-oriented concepts and programming languages Outsourced software development Fix-price contracts In-house development <u>Applications</u> Application frameworks Software applications Web-based systems Shrink-wrapped software Not well suited to development for reuse	Large scale programming (that can be broken down into smaller development teams Complex projects Projects with vague requirements Projects with constant requirements changes Projects that can be carried out by 5-9 programmers <u>Applications</u> Application frameworks designed for reuse Web deployed wireless technologies Web-based systems Object oriented systems General method suitable for any environment	Critical new business initiatives Projects under intense time pressure Projects with constant requirements changes Small to medium sized projects 10 developers or less Projects of up to 10,000 function points <u>Applications</u> e-commerce, e-business, data warehouse, products for Internet software market Client/server, networked, Internet application server projects Component development or component assembly using object technology	Object technology Smaller projects of 1 to 40 developers Software for the web Projects with: 2-8 developers in one room Onsite usage experts One month increments Automated regression tests Experienced developers <u>Applications</u> Business systems
Model	None, documented code is a substitute, structured or object-oriented modelling acceptable, models produced to aid design and maintenance	None, knowledge of object modelling techniques assumed	None, knowledge of object modelling techniques assumed	None	None, knowledge of UML is assumed

Summary Comparison of Agile Methods					
Criteria	**DSDM**	**XP**	**Scrum**	**ASD**	**Crystal**
Techniques: practices and metrics	Techniques Evolutionary prototyping Timeboxing Iteration of phases Incremental development 40 hour week Feasibility study Business study Regular meetings (preferably daily) Develop only what has direct business value MoSCow rules used to prioritise requirements Negotiation of requirements to maintain timebox limits JAD sessions Testing – unit, integration, system, acceptance, regression	Techniques Pair programming Planning game Simple design and refactoring Small releases Iterative and incremental development 1-4 week iterations Coding standards Collective ownership and pair programming Continuous integration and testing 40 hour week System metaphor On-site customer Metrics Room arrangements Listening Designing User stories	Techniques are designed to enable management to carry out control by observation and incremental adjustment leaving the team to carry out development unhindered Techniques Product Backlog Sprint Sprint Goal Sprint Backlog Sprint Planning meeting Daily Scrum Sprint Review meeting Release Backlog Customer on-site Work space configuration Daily builds and tests Testing (all types) Metrics: Product Backlog Graph Sprint Backlog Graph	Techniques are designed to support collaboration and learning and manage iterative development Techniques Component development – feature sets are assigned to components which are developed concurrently Function point counts Timeboxing Concurrent development Prototyping Learning Software inspections JAD Customer focus groups Post mortems Risk assessment Resource requirements analysis Team collocated Frequent meetings Dedicated meeting space Any other RAD techniques	Techniques are designed to support concurrency, communication, effective team work, iterative and incremental development Any standard software engineering and project management techniques are acceptable Techniques from XP, Scrum, DSDM and ASD, RUP Design for use interface Techniques Concurrent development Methodology creation or adaptation start up Reflective workshops for methodology adaptation Minimizing documentation Methodology publication guidelines Training using role plays Crystal clear: Team size up to 6 Team in one room 2-3 month increments Milestones to track progress Automated regression testing Direct user involvement 2 user viewings per release Concurrent phases Methodology tuning workshops

Summary Comparison of Agile Methods					
Criteria	**DSDM**	**XP**	**Scrum**	**ASD**	**Crystal**
Tools	An integrated set of tools: Tools for common user interface ,requirements analysis, system prototyping, design, construction, testing, reverse engineering , requirements management, configuration management, project/process management, documentation, shared repository, Virtual operating environment	Automated unit testing and integration	None	Tools for collaboration are specified: e-mail, calendaring, scheduling, e-conferencing, e-meeting systems, source-document creation tools, configuration management tools, threaded discussion tools No development tools are specified	Compiler Versioning and configuration management Regression testing Printing whiteboards
Scope	Development is iterative and incremental <u>Phases</u> Feasibility study Business study Functional model iteration System design and build iteration Implementation All phases contain analysis, design, coding and testing	Development is iterative and incremental <u>Phases</u> Exploration Planning Iteration to first release Produtionizing Maintenance Death All phases contain analysis, design, coding, testing	Development is iterative and incremental <u>Phases</u> Startup Iterations of: Sprint planning meeting Sprint Sprint review meeting	Development is iterative and incremental <u>Phases</u> Project initiation Adaptive lifecycle planning Concurrent component engineering Quality review Final quality assurance and release Three middle phases contain analysis, design, coding and testing	Development is concurrent, iterative and incremental Phases not proscribed <u>Phases</u> Requirements Design Programming Testing

Summary Comparison of Agile Methods					
Criteria	**DSDM**	**XP**	**Scrum**	**ASD**	**Crystal**
Practice	Background: Practitioner-based Controlled by consortium Suitable for use with CMM, TickIT Roles Senior developer Developer Technical coordinator Ambassador user Advisor user Visionary Executive sponsor Project manager Difficulties Not for scientific or engineering applications Occur when target environmental conditions not met Skill levels: Not stated	Background: Practitioner-based Roles Programmer Tracker Coach Consultant Big Boss Customer Difficulties big teams, distrustful customers, ungraceful change (non-object oriented systems), hierarchical business culture, achieving collaboration, achieving simplicity Skill levels: Average programmers On-site coach required	Background: Practitioner-based method Roles Scrum master Product Owner Scrum team Difficulties None Skill levels Not stated	Background: Practitioner-based Roles Executive sponsor Project manager Core team Collaboration facilitator Client team meetings Difficulties Not compatible with CMM Skill levels Individuals with high software engineering skills are critical Individuals who are good collaborators are necessary Skills in technology, business, problem-solving, decision making and interpersonal skills necessary	Background: Practitioner based with some aspects from academic studies Roles Depends on methodology weight Crystal clear: Sponsor Senior designer programmer Designer programmer User Difficulties Large projects, distributed teams Skill levels Individuals with experience and higher skill levels are better for light weight method like crystal clear Heavier methodologies need less skills such as in methods like crystal orange

Summary Comparison of Agile Methods					
Criteria	**DSDM**	**XP**	**Scrum**	**ASD**	**Crystal**
Product	Working software of an agreed quality delivered in increments Documentation to explain the project Models and documents to enable maintenance Training schemes and user manuals	Working software, code and unit test suite delivered in monthly increments	Working software delivered in monthly increments	Working software that meets business needs delivered in increments Explicit documentation is produced: Project vision Project data sheet Progress data Product specification outline Product mission profile	Working software that meets business needs delivered in increments Crystal clear outputs: Release sequence, schedule of user viewings, use cases, design sketches and notes, screen drafts, common object model, migration code, test cases, user manual, templates for work products, code and user interface and regression testing standards
Tailorability	Overall principles must be met Tailoring expected to suit the individual project	Local adaptation acceptable	No advice on tailorability Scalable to large projects Can be used with XP or other engineering practices without tailoring	Adaptation at startup Large projects require: greater rigor, more precise components, collaboration and documentation Advanced Adaptive Life Cycle - practices for larger projects	Local adaptation expected Crystal methods are designed to be tailored to the individual project

Appendix K Application of the analytical framework

This document is available on the compact disc (CD) attached to the inside back cover of this thesis. The file is named: Appendix K Application of the analytical framework.pdf.

Appendix L Data for factors in theoretical target environment

This is the data gathered to investigate the target environmental model of agile methods. Data was taken from the interview, questionnaire and organisation websites. The projects are divided into three groups in the data display shown in Table 59. 'High usage' projects are those projects that nominated an agile method as their development method and use greater than 75% of the techniques belonging to that method. 'Medium usage' projects are those projects that nominated an agile method as their development method and use less than 75% of the techniques belonging to that method. Projects where agile methods were not nominated as the method of development formed the third group. This includes both projects where RUP was used and those using ad hoc development. A percentage of agile use is given for the non-agile projects based on how many techniques they selected from the XP method.

This table shows the results from the questionnaire.

Key

- ✓ the factor is present in the project
- × the factor is not present in the project

Table 59: Data for the target environment model for agile methods

	RESULTS FROM QUESTIONNAIRE	Source									
	Project (case)		Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	Method used		XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	Use of agile methods (%)		96	94	67XP/67Scrum	63	56	54	40	30	18
	Usage category		High usage		Medium usage			Non-agile			
1	The organisation values feedback and learning.	Q5a Q8a	4 Often 5 Almost always	5 Almost always 4 Often	4 Often 4 Often	5 Almost always 4 Often	4 Often 4 Often	3 Sometimes 4 Often	3 Sometimes 2 Seldom	2 Seldom 1 Almost never	2 Seldom 3 Sometimes
2	The organisation values teamwork.	Q6a Q9a	5 Almost always 3 Sometimes	5 Almost always 5 Almost always	3 Sometimes 4 Often	4 Often 5 Almost always	4 Often 5 Almost always	4 Often 3 Sometimes	3 Sometimes 3 Sometimes	3 Sometimes 1 Almost never	3 Sometimes 3 Sometimes
3	The organisation values face-to-face communication.	Q35 Q36 Q37	Lots! 4 Mainly informal 4 Mainly informal	30 2 weeks 3 Balanced 4 Mainly informal	0.5/week 4 Mainly informal 3 Balanced	200+/2 week 4 Mainly informal 4 Mainly informal	1/day 5 Informal 5 Informal	1/day 4 Mainly informal 4 Mainly informal	0/2 week 2 Mainly formal 4 Mainly informal	0/2 week 5 Informal 4 Mainly informal	1/week 5 Informal 4 Mainly informal
4	The organisation enables empowerment of people.	Q4b Q34	5 Almost always 5 Almost always	5 Almost always 5 Almost always	4 Often 5 Almost always	4 Often 4 Often	5 Almost always 4 Often	4 Often 5 Almost always	1 Almost never 3 Sometimes	2 Seldom 4 Often	2 Seldom 2 Seldom
5	The organisation is flexible and participative and encourages social interaction.	Q6a	5 Almost always	5 Almost always	3 Sometimes	4 Often	4 Often	4 Often	3 Sometimes	3 Sometimes	3 Sometimes
6	Social interaction in the organisation is trustful, collaborative, and competent.	Q8a	5 Almost always	4 Often	4 Often	4 Often	4 Often	4 Often	2 Seldom	1 Almost never	3 Sometimes

	RESULTS FROM QUESTIONNAIRE	Source									
	Project (case)		Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	Method used		XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	Use of agile methods (%)		96	94	67XP/67Scrum	63	56	54	40	30	18
	Usage category		High usage		Medium usage			Non-agile			
7	Communication in the organisation is informal.	Q35 Q36 Q37	Lots! 4 Mainly informal 4 Mainly informal	30 2 weeks 3 Balanced 4 Mainly informal	0.5/week 4 Mainly informal 3 Balanced	200+/2 week 4 Mainly informal 4 Mainly informal	1/day 5 Informal 5 Informal	1/day 4 Mainly informal 4 Mainly informal	0/2 week 2 Mainly formal 4 Mainly formal	0/2 week 5 Informal 4 Mainly informal	1/week 5 Informal 4 Mainly informal
8	The management style is that of leadership and collaboration.	Q4a Q5a	5 Almost always 4 Often	5 Almost always 5 Almost always	4 Often 4 Often	4 Often 5 Almost always	5 Almost always 4 Often	3 Sometimes 3 Sometimes	2 Seldom 3 Sometimes	4 Often 2 Seldom	3 Sometimes 2 Seldom
9	The size of the organisation is large/ small. SD – software development staff	Q2 I W	13 SD 100-150 staff	10-30SD 300 staff	50 SD 13000-14000 staff	150+ SD 1400-1500 staff	45 SD 150-200 staff	3 SD 5-10 staff	24 SD 600 staff	10 SD 300-400 staff	Don't know 4500-5000 staff
10	Any of: Internet application domains, shrink wrapped software, web-based systems, component delivery, component development, component assembly, client/server systems, networked systems, web-deployed	Q1 Q19 Q20	Control system	DSS/MIS Intranet	Web	MIS	3D real-time data visualisation	Installer Shrink-wrapped software	Transaction processing	Web	LAN/Server architecture Windows desktop application Web interface Windows interface

	RESULTS FROM QUESTIONNAIRE	Source									
	Project (case)		Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	Method used		XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	Use of agile methods (%)		96	94	67XP/67Scrum	63	56	54	40	30	18
	Usage category		High usage		Medium usage			Non-agile			
	applications										
11	Automated testing is used TDD – Test-driven development ART = Automated regression testing	Q43	3 TDD 3 ART	2 TDD 3 ART	3 TDD 2 ART	1 TDD 1 ART	2 TDD 1 ART	0 TDD 0 ART	2 TDD 0 ART	0 TDD 0 ART	3 TDD 0 ART
12	Object-oriented technology.	Q20 Q21	✓ Object database Object-oriented concepts and languages Console interface Windows interface	✓ Web interface Web development (dynamic) Object-oriented concepts Relational database	✓ Web interface Web development (dynamic and static) Middleware Object-oriented concepts and languages Relational database	✓ Web interface Windows interface Mobile devices Object database Object-oriented concepts and languages Relational database	✓ Gaming software engines Console interface Windows interface Object-oriented concepts and languages Relational database	✓ Interface (windows) Object-oriented concepts, languages Services / Daemons	✓ Windows interface Middleware Object-oriented concepts, languages Relational database	✓ Windows interface Middleware Object-oriented concepts C#.Net Relational database	× Relational database
13	Any of: application frameworks for external use, e-commerce and e-business, data warehouse, and products for the Internet software market	Q17 Q19	None	None	None	None	None	None	None	None	None

	RESULTS FROM QUESTIONNAIRE	Source									
	Project (case)		Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	Method used		XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	Use of agile methods (%)		96	94	67XP/67Scrum	63	56	54	40	30	18
	Usage category		High usage		Medium usage			Non-agile			
14	The domain is interface intensive systems Assessed subjectively from type of system	Q17 Q19	×	✓	✓	✓	×	×	✓	✓	✓
15	The domain is business problems, business systems and applications	Q17 Q18 W	✓ Plant automation	✓ Change control system Marketing Management information Purchasing Sales Systems software Software change control	✓ HR system Management information Office automation Personnel and training Systems software Intranet system	✓ MIS	✓ Graphical presentation system	✓ Communication storage system - installer	✓ Claims system	✓ Contracts system	✓ Case management system
16	The domain is **non-critical projects**/ Critical new business initiative.	Q31 Q32	3 Essential income 3 Essential income	1 Loss of comfort N/A	1 Loss of comfort 1 Loss of comfort	3 Essential income 2 Discretionary income	3 Essential income 3 Essential income	1 Loss of comfort 1 Loss of comfort	1 Loss of comfort 2 Loss of discretionary income	3 Essential income 3 Essential income	1 Loss of comfort N/A
17	Projects that are complex.	Q21 Q22	9 tools 2 items	2 tools 1 item	7 tools 4 items	4 tools 3 items	6 tools 4 items	4 tools 2 items	16 tools 2 items	3 tools 1 item	3 tools 2 items
18	Projects with vague requirements	Q23	2 Minimal	4 Adequate	3 Some	4 Adequate	4 Adequate	3 Some	3 Some	4 Adequate	3 Some
19	Projects with constant changes in requirements/	Q24 Q25	15+ 10+	0 0	0 0	>20 0	0.1 0.2	5 5	0 0	3 1	0 0

	RESULTS FROM QUESTIONNAIRE	Source									
	Project (case)		Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	Method used		XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	Use of agile methods (%)		96	94	67XP/67Scrum	63	56	54	40	30	18
	Usage category		High usage		Medium usage			Non-agile			
	Projects with stable requirements.										
20	Projects undergoing constant change.	Q26 Q27	1 2	0 2	0 2	0 5+	0.1 0.3	0 0	0 0	0 0	0 1
21	Projects with intense time pressure.	Q29	3 Moderate	5 Extreme	5 Extreme	5 Extreme	5 Extreme	3 Moderate	4 High	4 High	4 High
22	Projects with teams of 2 to 10 developers. ft - full time pt – part time	Q14 Q15	6 ft 1pt	4 ft 0pt	13ft 6pt	30+ 0	10ft 1pt	1 ft 1 pt	6ft 2pt	4ft 2pt	2-3ft 0pt
23	Projects where documentation is minimized.	I5	'Definitely' Some essential documentation produced	"Absolutely" Documentation produced has to move project forward	Question missed	'Absolutely, but they do enough'	"Yes", code comments and user documentation is used	Not assessed	Not assessed	Not assessed	Not assessed
24	Projects involving new development/ Precedented systems/ Projects that are not constrained by an existing computing environment.	Q12	New system Enhancement to existing system Replacement of existing system Maintenance	New system	Replacement of existing system with enhancement	Enhancement to existing system	Enhancement to existing system	New system Enhancement to existing system On-going product development	Enhancement and replacement of existing system	Enhancement to existing system Maintenance	Enhancement to existing system
25	Projects involving any of fix-price contract software	Q1	✓ Provides specific	✓ Provides specific	✓ In-house	✓ Provides	✓ Provides	✓ Provides specific	✓ Carry out own	✓ Provides specific	✓ Customising pre-

	RESULTS FROM QUESTIONNAIRE	Source									
	Project (case)		Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	Method used		XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	Use of agile methods (%)		96	94	67XP/67Scrum	63	56	54	40	30	18
	Usage category		High usage		Medium usage			Non-agile			
	development, in-house development, outsourced software,		products for external customers Completing one-off contracts Customising pre-developed software (own)	products for external customers Completing one-off contracts Provides in-house development of own product	development of own product Customisation of brought-in products Develop software for inclusion in the organisations products by in-house development and purchasing software from an outside source	specific products for external customers Completing one-off contracts Customising pre-developed software Providing service & support of software systems	specific products for external customers Completes one-off contracts Customises pre-developed software Provides service and support of software systems Develops software for inclusion in organisations products by in-house development and purchasing software from an outside source	products for external customers by mass production of shrink-wrapped products Provides service and support of software systems (probably fixed price contracts)	development In-house Customise brought-in products	products for external customers by completing on-off contracts, customising pre-developed software, providing service and support In-house development Own development and customisation of brought-in products Develops in conjunction with external groups	developed software Providing service / support of software systems In-house development Customising brought-in products Purchasing software
26	Projects using incremental development.	Q43	3 Always	3 Always	3 Always	1 Seldom	2 Often	3 Always	2 Often	0 Never	0 Never
27	Projects using iterative	Q43 Q45	3 Always 1 week	3 Always 4 week	3 Always flexible	3 Always 2 month	2 often 1 month	3 Always 1-2 days	2 Often 1 month	2 Often 0 no	0 Never

	RESULTS FROM QUESTIONNAIRE	Source									
	Project (case)		Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	Method used		XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	Use of agile methods (%)		96	94	67XP/67Scrum	63	56	54	40	30	18
	Usage category		High usage		Medium usage			Non-agile			
	development.									iterations	
28	Projects where the project manager acts as a facilitator.	Q5a	4 Often	5 Almost always	4 often	5 Almost always	4 Often	3 Sometimes	3 Sometimes	2 Seldom	2 Seldom
29	Projects teams are collocated.	Q43	3 Always	2 Often	3 Always	3 Always	3 Always	3 Always	2 Often	3 Always	0 Never
30	Users are actively involved in the project	Q16 Q35 Q43	300+ users Lots! 3 COS Always	300 users 30 /2 weeks 3 COS Always	7000 users 1 /2 weeks 1 COS Seldom	3000 users 200+/2 weeks 3 COS Always	50 users 1/day contact 1 COS Seldom	Not known 1/day 2 COS Often	Don't know 0/2 week 1 COS Seldom	Don't know 0/2 week 1 Seldom	100 users 1/week 1 COS Seldom
31	Developers are experienced	Q33	3 Moderate	5 Highly experienced	2 Slightly	4 Very	4 Very	5 highly	2 Slightly	4 Very	5 highly
		Q38: Role	Software developer, method coach	Project manager	Project manager, analyst	Practice manager	Project director	Subproject lead, developer, tester	Method coach and mentor	Analyst Programmer Support	Programmer Analyst
		Q39 main role	5 yr	15 yr	10 yr	10 yr	10 yr	3-5 yr	2 yr	Up to 6 month	10 yr
		Q40 Agile method	5 yr	10 yr	1 yr	4 yr	1 yr	n/a	n/a	Up to 6 month	n/a
		Q41 Business domain	5 yr	5 yr	20+ yr	4 yr	8 yr	never before	never before	Up to 6 month	never before
		Q42 Software development	15 yr	21 yr	15+ yr	10 yr	30 yr	9+ yr	25 yr	1 yr	20 yr

	RESULTS FROM QUESTIONNAIRE	Source									
	Project (case)		Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho
	Method used		XP	DSDM	XP/Scrum	RUP/XP	XP	Ad hoc	RUP	Ad hoc	Ad hoc
	Use of agile methods (%)		96	94	67XP/67Scrum	63	56	54	40	30	18
	Usage category		High usage		Medium usage			Non-agile			
		Q3: No. agile projects	Ongoing over 5 yrs on 2 products	100+ projects	0 projects	50+ projects	1 project	n/a	n/a	n/a	n/a
32	Budgetary constraints*	Q28	4 None	2 Adequate	5 Adequate	2 Adequate	3 Generous	4 None	5 Don't know	5 Don't know	2 Adequate
33	Quality expected*	Q30	4 High	4 High	4 High	3 Moderate	4 High	4 High	4 High	3 Moderate	4 High
*not part of model but used in questionnaire											

Appendix M Agile method tailoring

This appendix shows the data from each of the projects and the techniques they use. Each project was assessed to find:

- The percentage of techniques used as nominated in the questionnaire (a count). This was calculated using Equation 1: The percentage of techniques used: page 98.
- The extent of usage of the techniques as assessed against each of the five agile methods (a quantity). This was calculated using Equation 2: The percentage of agile method usage: page 98.

Table 60 and Figure 16 show the usage of all techniques listed in the questionnaire.

Table 60: Count of all techniques used on all projects

Project	Alpha	Zeta	Delta	Theta	Beta	Iota	Tau	Chi	Rho
Method used	XP	DSDM	XP Scrum	RUP XP	XP	RUP	Ad hoc	Ad hoc	Ad hoc
All techniques used (%)	67	98	75	92	81	67	50	50	15
XP techniques used (%)	100	100	84	100	95	79	68	58	32
DSDM techniques used (%)	56	100	81	100	94	69	69	44	25
Scrum techniques used (%)	54	100	85	69	62	62	46	38	8
Crystal techniques used (%)	80	90	70	100	80	80	50	40	0
ASD techniques used (%)	67	100	83	100	92	83	58	67	0

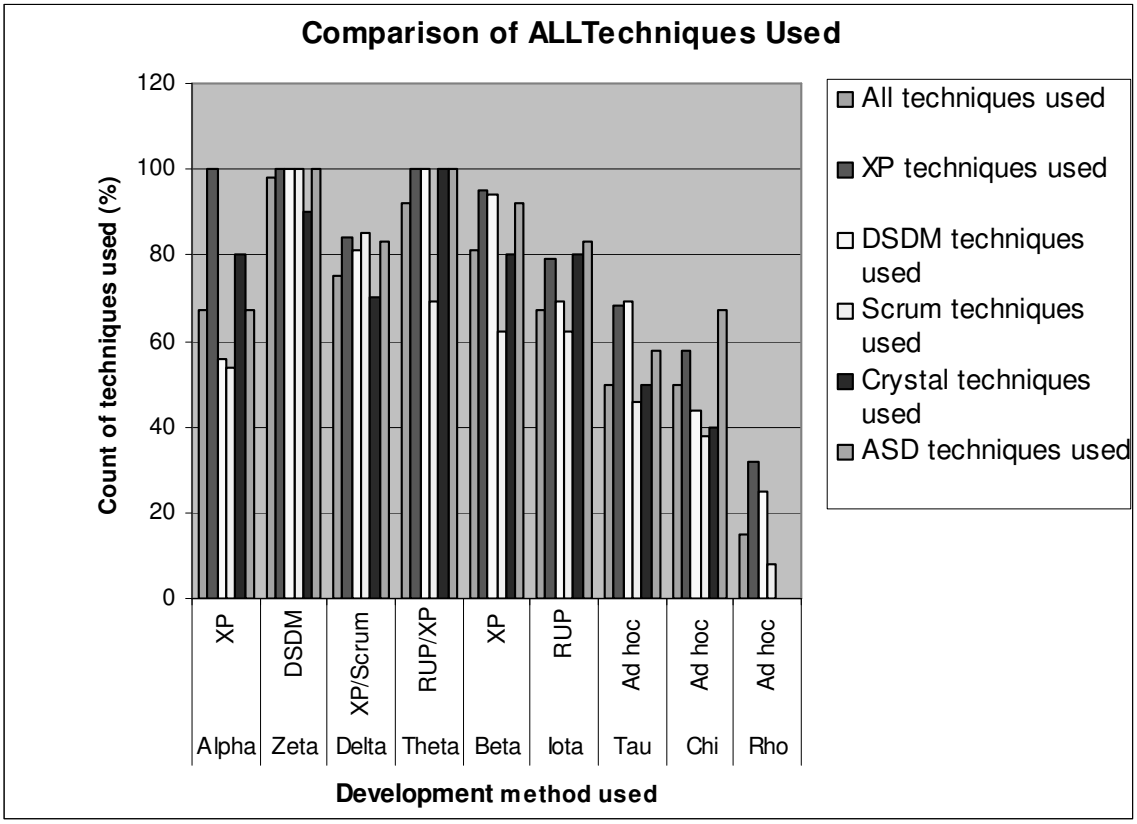

Figure 16: Count of all techniques used on all projects

Table 61 and Figure 17 show usage of all techniques. Data is the quantity of techniques used reported as percentages.

Table 61: Extent of usage of all techniques used on all projects

Project	Alpha	Zeta	Delta	Theta	Beta	Iota	Tau	Chi	Rho
Method used	XP	DSDM	XP Scrum	RUP XP	XP	RUP	Ad hoc	Ad hoc	Ad hoc
Quantity All (%)	60	81	56	66	51	34	38	22	10
Quantity XP (%)	96	86	67	63	56	40	54	30	18
Quantity DSDM (%)	52	94	65	79	65	38	54	29	19
Quantity Scrum (%)	51	77	67	49	41	36	38	21	3
Quantity Crystal (%)	70	80	57	73	60	50	43	23	0
Quantity ASD (%)	56	83	58	78	53	42	50	31	0

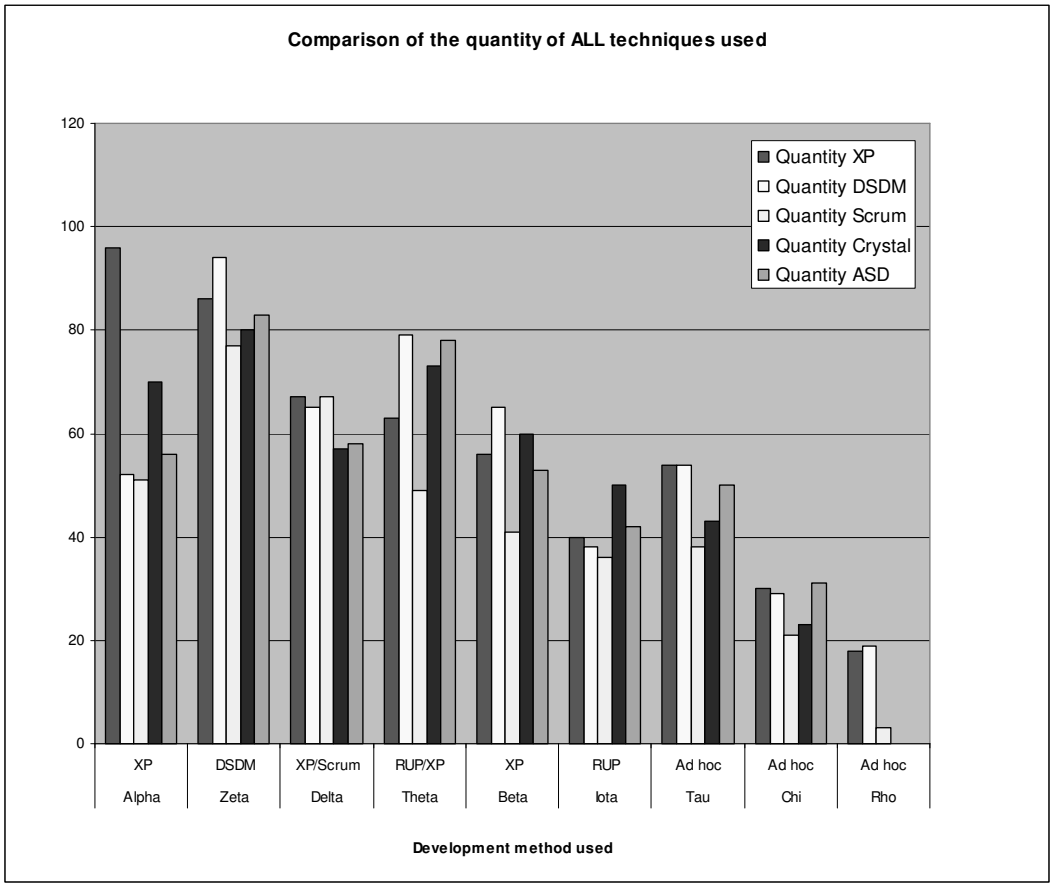

Figure 17: Quantity of all techniques used on all projects

In order to compare all of the projects, both agile and non-agile, it was necessary to calculate a method usage value for each of the non-agile projects. This was done by comparing each non-agile project with their usage of each of the agile methods techniques. This provided five sets of data, each set comparing a non-agile project with the extent of method usage of each of the five methods; the XP method, the DSDM method, the Scrum method, the Crystal method, and the ASD method. Table 62 shows the extent to which each of the non-agile projects used each of the methods. The sum of the ranks shows that XP is the highest ranked method (i.e. more XP techniques were used on the non-agile projects than techniques from the other agile methods). So for the non-agile projects XP was nominated as the method for the project. The XP usage value was used for subsequent comparison of the projects. This meant that a comparison of the method usage and environment factors in the target environment model (technology, domain, people, project and organisational) could be carried out for all projects both agile and non-agile. The analysis is presented in Chapter 5 Results.

Table 62: Agile method usage (%) for non-agile projects

	Iota	Tau	Chi	Rho	Σ ranks
Method used on project	RUP	Ad hoc	Ad hoc	Ad hoc	
XP	40 (3)	54 (1)	30 (2)	18 (2)	8
DSDM	38 (4)	54 (1)	29 (3)	19 (1)	9
Scrum	36 (5)	38 (5)	21 (5)	3 (3)	18
Crystal	50 (1)	43 (4)	23 (4)	0 (4)	13
ASD	42 (2)	50 (3)	31 (1)	0 (4)	10
Note: Bracketed number is the rank from lowest usage (5) to highest (1)					

Table 63 and Figure 18 show usage of XP techniques.

Table 63: XP techniques used

Project	Alpha	Beta	Delta	Zeta	Theta	Iota	Rho	Tau	Chi
Method used	XP	XP	XP Scrum	DSDM	RUP XP	RUP	ad hoc	ad hoc	ad hoc
Count of XP techniques used (%)	100	95	84	100	100	79	32	68	58
Quantity of XP techniques used (%)	96	56	67	86	63	40	18	54	30

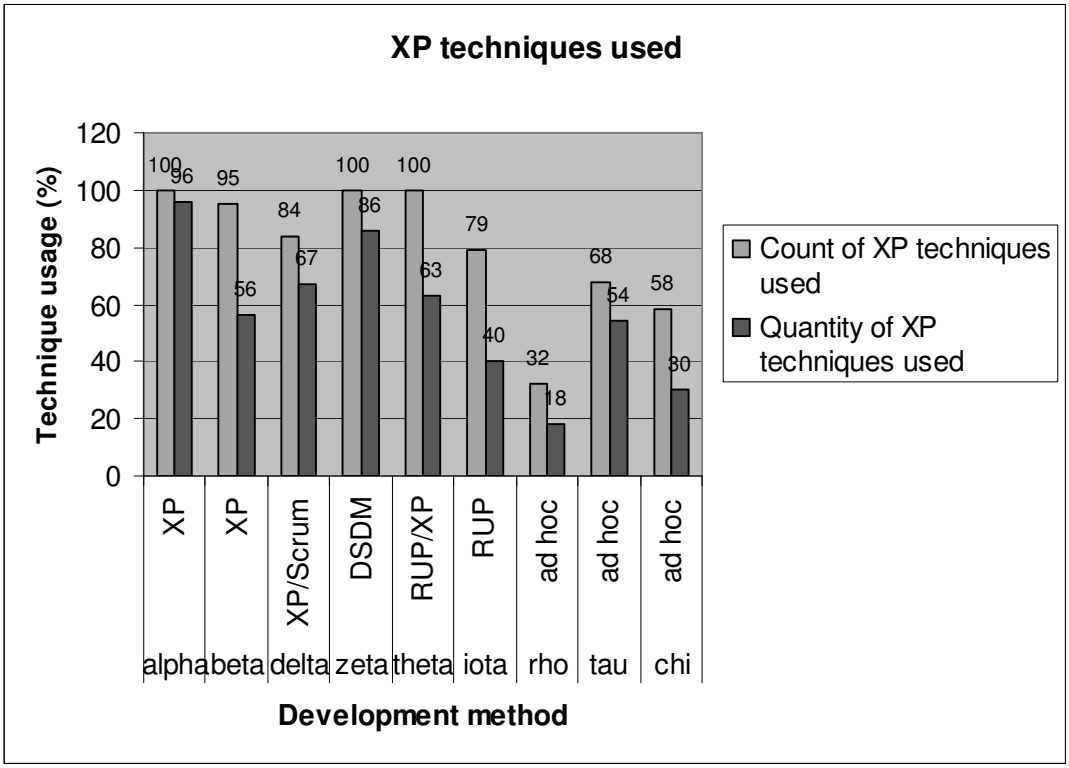

Figure 18: XP techniques used

Table 64 and Figure 19 show the usage of Scrum techniques.

Table 64: Scrum techniques used

Project	Alpha	Beta	Delta	Zeta	Theta	Iota	Rho	Tau	Chi
Method used	XP	XP	XP Scrum	DSDM	RUP XP	RUP	ad hoc	ad hoc	ad hoc
Count of Scum techniques used (%)	54	62	85	100	69	62	8	46	38
Quantity of Scrum techniques used (%)	51	41	67	77	49	36	3	38	21

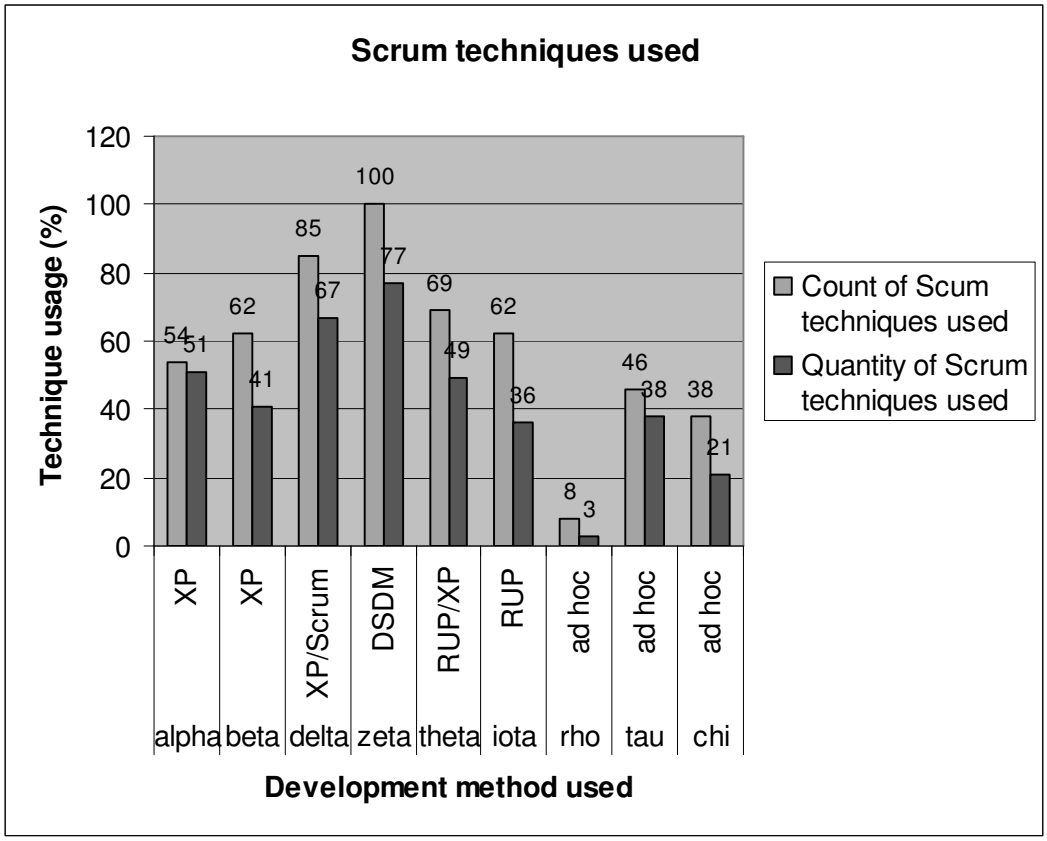

Figure 19: Scrum techniques used

Table 65 and Figure 20 show usage of DSDM techniques.

Table 65: DSDM techniques used

Project	Alpha	Beta	Delta	Zeta	Theta	Iota	Rho	Tau	Chi
Method used	XP	XP	XP Scrum	DSDM	RUP XP	RUP	ad hoc	ad hoc	ad hoc
Count of DSDM techniques used (%)	56	94	81	100	100	69	25	69	44
Quantity of DSDM techniques used (%)	52	65	65	94	79	38	19	54	29

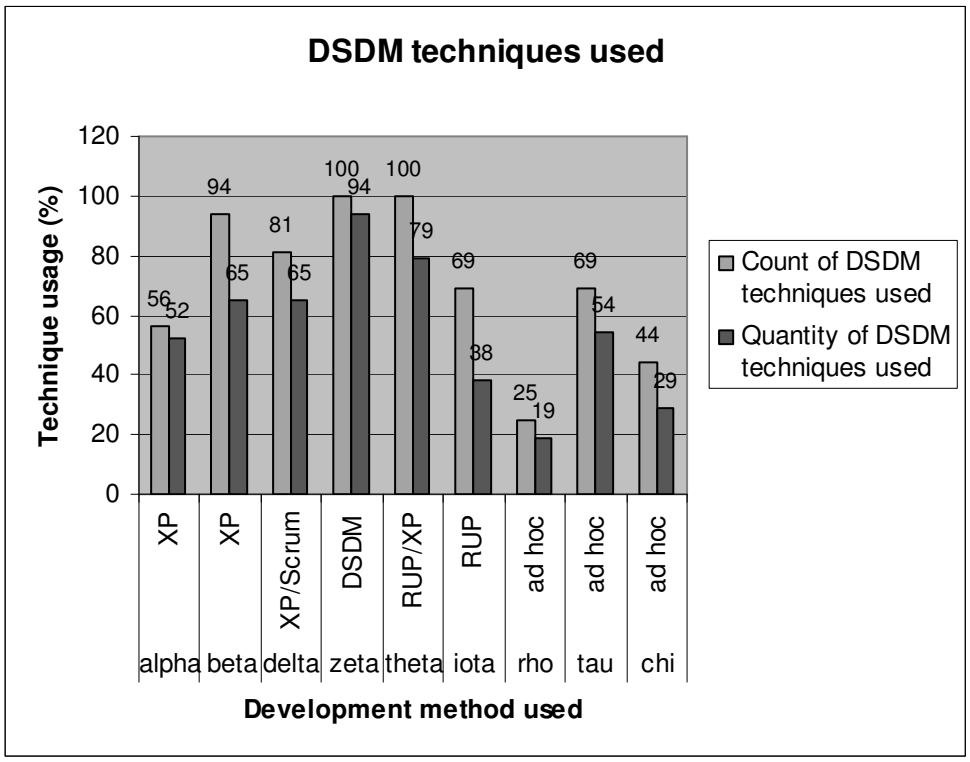

Figure 20: DSDM techniques used

Table 66 and Figure 21 show usage of Crystal methods techniques.

Table 66: Crystal techniques used

Project	Alpha	Beta	Delta	Zeta	Theta	Iota	Rho	Tau	Chi
Method used	XP	XP	XP /Scrum	DSDM	RUP /XP	RUP	ad hoc	ad hoc	ad hoc
Count of Crystal techniques used (%)	80	80	70	90	100	80	0	50	40
Quantity of Crystal techniques used (%)	70	60	57	80	73	50	0	43	23

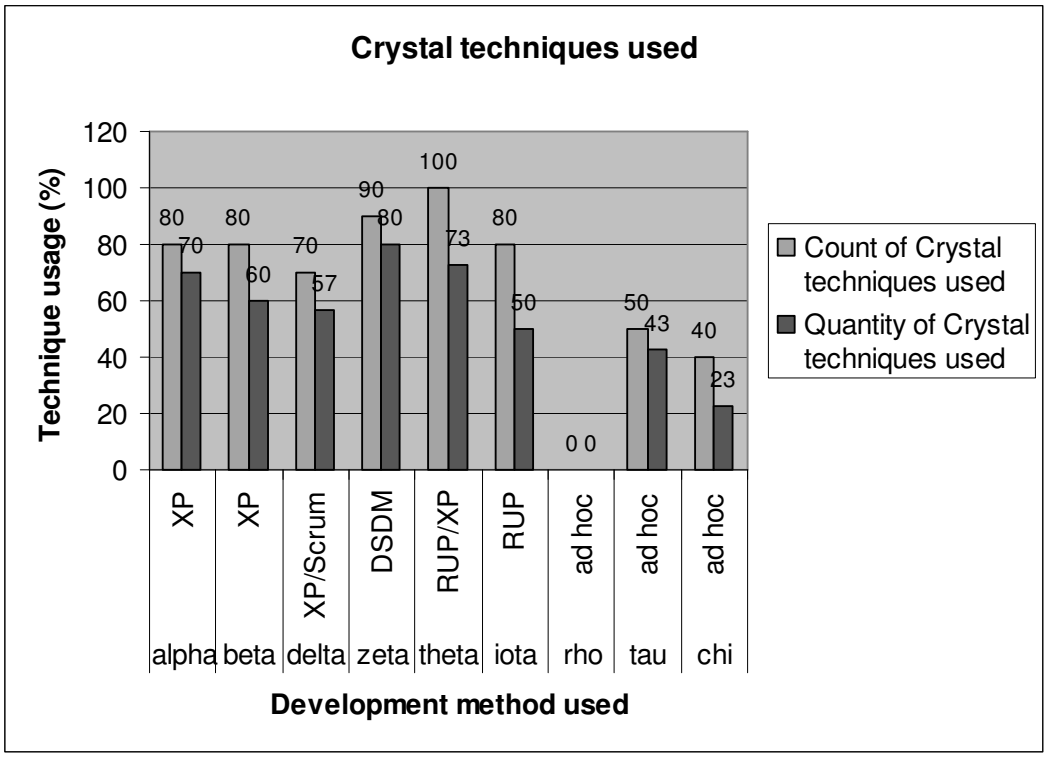

Figure 21: Crystal techniques used

Table 67 and Figure 22 show usage of ASD techniques.

Table 67: ASD techniques used

Project	Alpha	Beta	Delta	Zeta	Theta	Iota	Rho	Tau	Chi
Method used	XP	XP	XP Scrum	DSDM	RUP XP	RUP	ad hoc	ad hoc	ad hoc
Count of ASD techniques used (%)	67	92	83	100	100	83	0	58	67
Quantity of ASD techniques used (%)	56	53	58	83	78	42	0	50	31

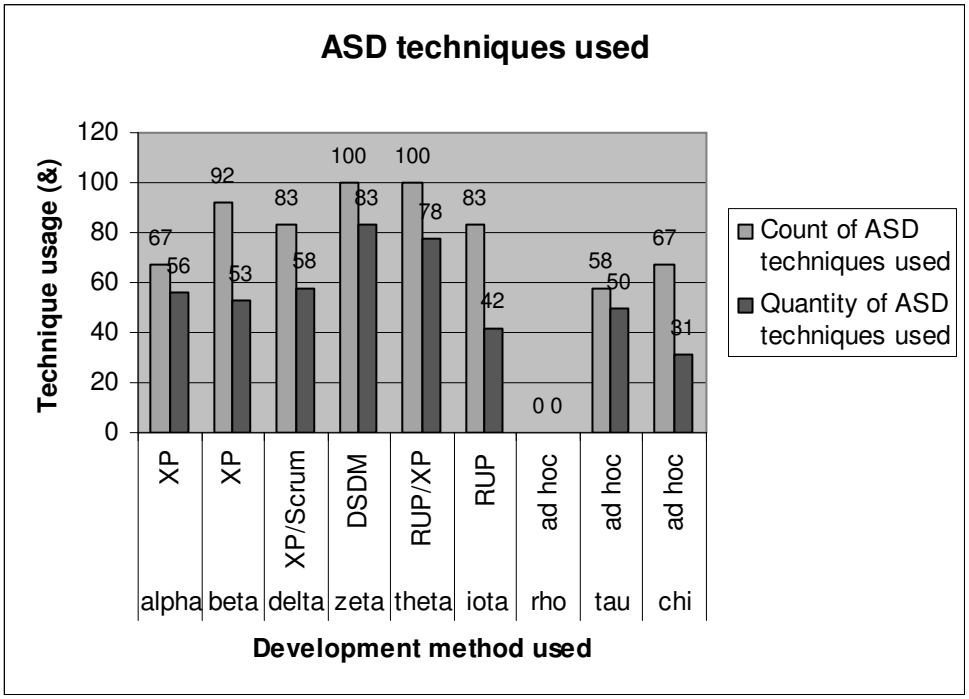

Figure 22: ASD techniques used

Appendix N Organisational culture data

Table 68 shows the data on organisational culture gathered from the questionnaire. A percentage of agile use is given for the non-agile projects based on how many techniques they selected from the XP method.

Table 68: Organisational culture data from questionnaire

Project (case)	Alpha	Zeta	Delta	Theta	Beta	Tau	Iota	Chi	Rho	
Method used	XP	DSDM	XP Scrum	XP RUP	XP	Ad hoc	RUP	Ad hoc	Ad hoc	
Use of agile method (%)	96	94	67XP 67Scrum	63	56	54	40	30	18	
	High usage		Medium usage			Non-agile				
4a	5	5	4	4	5	3	2	4	3	
4b	5	5	4	4	5	4	1	2	2	
4c	5	5	3	5	4	3	4	3	3	
4d	3	3	5	4	4	1	2	3	4	
5a	5	4	4	5	4	3	3	2	2	
5b	4	5	4	4	4	4	2	3	3	
5c	3	4	3	4	3	2	4	4	1	
5d	3	4	4	4	4	3	2	2	5	
6a	5	5	3	4	4	4	3	3	3	
6b	5	3	3	3	4	4	1	1	2	
6c	3	2	4	5	3	2	4	5	2	
6d	3	5	2	4	2	3	3	4	3	
7a	5	4	4	5	4	4	3	2	3	
7b	4	5	4	3	5	5	1	2	2	
7c	3	5	2	4	3	3	4	3	2	
7d	2	3	2	2	3	1	3	3	5	
8a	4	5	4	4	4	4	2	1	3	
8b	3	5	3	3	4	3	3	2	3	
8c	3	5	2	4	4	3	2	4	3	
8d	3	4	5	3	3	1	4	1	5	
9a	5	3	4	5	5	3	3	1	3	
9b	2	5	1	2	5	5	2	2	2	
9c	3	5	1	3	4	4	1	5	2	
9d	3	*	4	2	4	2	4	4	4	

Key

* no response

1 almost never
2 seldom
3 sometimes
4 often
5 almost always

Appendix O Organisational culture data analysis

This Appendix presents the results of statistical analysis of organisational culture data for each project. The following tables are the output from SPSS. All tests used Spearman's rank correlation coefficient.

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.719(*)
		Sig. (2-tailed)	.	.029
		N	9	9
	4a OC personal place	Correlation Coefficient	.719(*)	1.000
		Sig. (2-tailed)	.029	.
		N	9	9

* Correlation is significant at the 0.05 level (2-tailed).

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.806(**)
		Sig. (2-tailed)	.	.009
		N	9	9
	4b OC dynamic place	Correlation Coefficient	.806(**)	1.000
		Sig. (2-tailed)	.009	.
		N	9	9

** Correlation is significant at the 0.01 level (2-tailed).

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.677(*)
		Sig. (2-tailed)	.	.045
		N	9	9
	4c OC results oriented place	Correlation Coefficient	.677(*)	1.000
		Sig. (2-tailed)	.045	.
		N	9	9

* Correlation is significant at the 0.05 level (2-tailed).

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.138
		Sig. (2-tailed)	.	.723
		N	9	9
	4d OC controlled place	Correlation Coefficient	.138	1.000
		Sig. (2-tailed)	.723	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.842(**)
		Sig. (2-tailed)	.	.004
		N	9	9
	5a OC mentoring leadership	Correlation Coefficient	.842(**)	1.000
		Sig. (2-tailed)	.004	.
		N	9	9

** Correlation is significant at the 0.01 level (2-tailed).

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.853(**)
		Sig. (2-tailed)	.	.003
		N	9	9
	5b OC entrepreneurial leadership	Correlation Coefficient	.853(**)	1.000
		Sig. (2-tailed)	.003	.
		N	9	9

** Correlation is significant at the 0.01 level (2-tailed).

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.248
		Sig. (2-tailed)	.	.520
		N	9	9
	5c OC no nonsense leadership	Correlation Coefficient	.248	1.000
		Sig. (2-tailed)	.520	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.158
		Sig. (2-tailed)	.	.685
		N	9	9
	5d OC coordinated leadership	Correlation Coefficient	.158	1.000
		Sig. (2-tailed)	.685	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.748(*)
		Sig. (2-tailed)	.	.020
		N	9	9
	6a OC teamwork management	Correlation Coefficient	.748(*)	1.000
		Sig. (2-tailed)	.020	.
		N	9	9

* Correlation is significant at the 0.05 level (2-tailed).

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.598
		Sig. (2-tailed)	.	.089
		N	9	9
	6b OC risk taking management	Correlation Coefficient	.598	1.000
		Sig. (2-tailed)	.089	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	-.094
		Sig. (2-tailed)	.	.809
		N	9	9
	6c OC competitive management	Correlation Coefficient	-.094	1.000
		Sig. (2-tailed)	.809	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.105
		Sig. (2-tailed)	.	.787
		N	9	9
	6d OC secure management	Correlation Coefficient	.105	1.000
		Sig. (2-tailed)	.787	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.764(*)
		Sig. (2-tailed)	.	.016
		N	9	9
	7a OC loyalty	Correlation Coefficient	.764(*)	1.000
		Sig. (2-tailed)	.016	.
		N	9	9

- Correlation is significant at the 0.05 level (2-tailed).

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.616
		Sig. (2-tailed)	.	.078
		N	9	9
	7b OC innovation	Correlation Coefficient	.616	1.000
		Sig. (2-tailed)	.078	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.369
		Sig. (2-tailed)	.	.329
		N	9	9
	7c OC achievement	Correlation Coefficient	.369	1.000
		Sig. (2-tailed)	.329	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	-.452
		Sig. (2-tailed)	.	.222
		N	9	9
	7d OC rules	Correlation Coefficient	-.452	1.000
		Sig. (2-tailed)	.222	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.858(**)
		Sig. (2-tailed)	.	.003
		N	9	9
	8a OC openness	Correlation Coefficient	.858(**)	1.000
		Sig. (2-tailed)	.003	.
		N	9	9

** Correlation is significant at the 0.01 level (2-tailed).

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.564
		Sig. (2-tailed)	.	.113
		N	9	9
	8b OC challenges	Correlation Coefficient	.564	1.000
		Sig. (2-tailed)	.113	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.243
		Sig. (2-tailed)	.	.529
		N	9	9
	8c OC competitive	Correlation Coefficient	.243	1.000
		Sig. (2-tailed)	.529	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.103
		Sig. (2-tailed)	.	.792
		N	9	9
	8d OC permanence	Correlation Coefficient	.103	1.000
		Sig. (2-tailed)	.792	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.541
		Sig. (2-tailed)	.	.133
		N	9	9
	9a OC teamwork	Correlation Coefficient	.541	1.000
		Sig. (2-tailed)	.133	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.112
		Sig. (2-tailed)	.	.775
		N	9	9
	9b OC innovator	Correlation Coefficient	.112	1.000
		Sig. (2-tailed)	.775	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.136
		Sig. (2-tailed)	.	.728
		N	9	9
	9c OC market leadership	Correlation Coefficient	.136	1.000
		Sig. (2-tailed)	.728	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	-.412
		Sig. (2-tailed)	.	.310
		N	9	8
	9d OC efficiency	Correlation Coefficient	-.412	1.000
		Sig. (2-tailed)	.310	.
		N	8	8

Appendix P Technique use data

Table 69 presents the data for each of the projects, their nominated method and the techniques they used. Data is from question 43 of the questionnaire. Summed data is used to indicate the most commonly used techniques for agile projects and non-agile projects.

Table 69: Data for techniques used on projects

	Project	Alpha	Beta	Delta	Zeta	Theta		Iota	Rho	Tau	Chi		
	Method	XP	XP	XP Scrum	DSDM	RUP XP	Sum Agile	RUP	Ad hoc	Ad hoc	Ad hoc	Sum non-agile	Sum All
1	Concurrent development	3	2	0	2	1	8	1	0	2	1	4	12
2	Iterative development	3	2	3	3	3	14	2	0	3	2	7	21
3	Time boxing (iterations of set length)	3	2	2	3	2	12	2	0	0	0	2	14
4	Incremental development	3	2	3	3	1	12	2	0	3	0	5	17
5	Evolutionary prototyping	0	2	1	2	2	7	1	0	3	0	4	11
6	Small releases of software product	3	2	1	3	2	11	1	0	3	0	4	15
7	Component development	2	2	3	2	3	12	2	0	2	0	4	16
8	Test first development	3	2	3	2	1	11	2	3	0	0	5	16
9	Daily builds of complete system	3	2	3	2	2	12	1	0	2	0	3	15
10	Automated regression testing	3	1	2	2	1	9	0	0	0	0	0	9
11	Refactoring of code	3	2	2	3	1	11	2	0	3	0	5	16
12	Testing throughout each iteration	3	1	2	3	1	10	1	1	1	0	3	13
13	Software inspections	3	1	1	3	2	10	1	0	0	1	2	11
14	Customer on-site	3	1	1	3	3	11	1	1	2	1	5	15
15	Method coach on site	3	1	3	2	2	11	3	0	0	0	3	14
16	Tester(s) collocated with team	3	3	2	2	3	13	1	0	0	1	2	14
17	Customer focus groups	0	1	1	3	2	7	0	0	0	1	1	7
18	Rooms organised for pair programming	3	0	0	2	2	7	0	0	0	0	0	7
19	Whole team works in same office/floor	3	3	3	2	3	14	2	0	3	3	8	19
20	Dedicated meeting space	1	2	0	2	3	8	1	0	3	1	5	12

	Project	Alpha	Beta	Delta	Zeta	Theta		Iota	Rho	Tau	Chi		
	Method	XP	XP	XP Scrum	DSDM	RUP XP	Sum Agile	RUP	Ad hoc	Ad hoc	Ad hoc	Sum non-agile	Sum All
21	Pair programming	3	2	1	2	2	10	0	0	0	1	1	10
22	Coding to an agreed standard	3	2	3	3	2	13	1	0	3	1	5	17
23	Collective ownership of code	3	2	3	3	3	14	1	0	3	2	6	18
24	40 hour week	3	1	2	3	2	11	3	2	2	2	9	18
25	Sprint Goal	0	1	1	2	0	4	0	0	0	0	0	4
26	Daily team meetings	1	2	0	3	2	8	1	0	1	1	3	10
27	Iteration planning meeting	3	1	3	3	2	12	2	0	0	1	3	14
28	Planning game	3	1	2	3	1	10	0	0	0	0	0	10
29	Reflective workshops for adaptation	1	0	1	3	2	7	1	0	0	0	1	8
30	User stories	3	2	0	2	1	8	1	0	1	1	3	10
31	System metaphor developed	1	1	0	2	1	5	0	0	0	1	1	5
32	Only what has direct business value	3	2	2	3	2	12	1	0	2	2	5	15
33	Requirements are prioritised	3	2	2	3	2	12	1	3	2	2		18
34	Changes to requirements are negotiated	3	2	3	3	3	14	1	3	0	2	6	18
35	Joint Application Development (JAD)	0	1	2	3	3	9	0	0	0	1	1	9
36	Design is kept as simple as possible	3	3	3	2	3	14	1	1	2	2	6	18
37	Coded solution is kept as simple as possible	3	2	3	2	2	12	1	2	2	2	7	17
38	Risk assessment at each iteration	2	1	2	2	3	10	1	0	1	1	3	12
39	Product Backlog	0	0	3	2	1	6	0	0	0	1	1	6
40	Sprint Backlog	0	0	1	2	0	3	0	0	0	0	0	3
41	Release Backlog	0	0	3	2	1		2	0	0	0	2	8
42	Milestones to track progress	2	3	3	3	3	14	2	0	2	1	5	18
43	Product Backlog Graph metric	0	0	0	2	0	2	0	0	0	0	0	2
44	Sprint Backlog Graph metric	0	0	0	1	0	1	0	0	0	0	0	1
45	Function	0	0	0	2	1	3	1	0	0	0	1	4

	Project	Alpha	Beta	Delta	Zeta	Theta		Iota	Rho	Tau	Chi		
	Method	XP	XP	XP Scrum	DSDM	RUP XP	Sum Agile	RUP	Ad hoc	Ad hoc	Ad hoc	Sum non-agile	Sum All
	point counts												
46	Project post mortem	0	3	3	1	3	10	3	0	3	0	6	16
47	Feasibility study	0	3	0	3	3	9	0	0	0	0	0	9
48	Business study	0	3	3	3	3	12	0	0	3	0	3	15
49	Resource requirements analysis	0	3	3	3	3	12	0	0	0	0	0	12
50	MOSCOW rules	0	0	0	3	3	6	0	0	3	0	3	9
51	Unique methodology at start	0	0	0	0	3	3	0	0	0	0	0	3
52	Tailored existing methodology at start	0	3	0	3	3	9	3	0	0	0	3	12
	TOTALS												
	Number of available techniques = 52												
	Count of techniques used = 156	35	42	39	51	48		35	8	26	26		
	% Count of techniques used	67	81	75	98	92		67	15	50	50		
	Quantity of techniques used	94	80	88	126	103		53	16	60	35		
	% Quantity of techniques used	60	51	56	81	66		34	10	38	22		

Appendix Q Project data analysis

The following tables show the SPSS analysis of project environment data gathered from the questionnaire. Spearman's rank correlation coefficient or Pearson's product moment coefficient was calculated depending on the type of data (ordinal or ratio).

% Agile (extent of techniques used)	Pearson Correlation	1	.040
	Sig. (2-tailed)		.925
	N	9	8
Q14 Number of full-time developers	Pearson Correlation	.040	1
	Sig. (2-tailed)	.925	
	N	8	8

% Agile (extent of techniques used)	Pearson Correlation	1	-.001
	Sig. (2-tailed)		.998
	N	9	9
Q15 Number of part-time developers	Pearson Correlation	-.001	1
	Sig. (2-tailed)	.998	
	N	9	9

% Agile (extent of techniques used)	Pearson Correlation	1	.032
	Sig. (2-tailed)		.952
	N	9	6
Q16 Number of end-users	Pearson Correlation	.032	1
	Sig. (2-tailed)	.952	
	N	6	6

% Agile (extent of techniques used)	Pearson Correlation	1	.032
	Sig. (2-tailed)		.935
	N	9	9
Q21 Number of different tools used	Pearson Correlation	.032	1
	Sig. (2-tailed)	.935	
	N	9	9

% Agile (extent of techniques used)	Pearson Correlation	1	.037
	Sig. (2-tailed)		.924
	N	9	9
Q22 Number of complexity factors	Pearson Correlation	.037	1
	Sig. (2-tailed)	.924	
	N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.075
		Sig. (2-tailed)	.	.849
		N	9	9
	Q23 Requirements defined at start	Correlation Coefficient	.075	1.000
		Sig. (2-tailed)	.849	.
		N	9	9

% Agile (extent of techniques used)	Pearson Correlation	1	.370
	Sig. (2-tailed)		.327
	N	9	9
Q24 New functional requirements	Pearson Correlation	.370	1
	Sig. (2-tailed)	.327	
	N	9	9

% Agile (extent of techniques used)	Pearson Correlation	1	.465
	Sig. (2-tailed)		.207
	N	9	9
Q25 Existing functional requirements	Pearson Correlation	.465	1
	Sig. (2-tailed)	.207	
	N	9	9

% Agile (extent of techniques used)	Pearson Correlation	1	.552
	Sig. (2-tailed)		.123
	N	9	9
Q26 Technical requirements	Pearson Correlation	.552	1
	Sig. (2-tailed)	.123	
	N	9	9

% Agile (extent of techniques used)	Pearson Correlation	1	.428
	Sig. (2-tailed)		.251
	N	9	9
Q27 Other changes	Pearson Correlation	.428	1
	Sig. (2-tailed)	.251	
	N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.060
		Sig. (2-tailed)	.	.899
		N	9	7
	Q28 Budgetry constraints	Correlation Coefficient	.060	1.000
		Sig. (2-tailed)	.899	.
		N	7	7

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.241
		Sig. (2-tailed)	.	.533
		N	9	9
	Q29 Time pressure	Correlation Coefficient	.241	1.000
		Sig. (2-tailed)	.533	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.207
		Sig. (2-tailed)	.	.593
		N	9	9
	Q30 Software quality level	Correlation Coefficient	.207	1.000
		Sig. (2-tailed)	.593	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.173
		Sig. (2-tailed)	.	.656
		N	9	9
	Q31 Criticality for your organisation	Correlation Coefficient	.173	1.000
		Sig. (2-tailed)	.656	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	.296
		Sig. (2-tailed)	.	.476
		N	9	8
	Q32 Criticality for client organisation	Correlation Coefficient	.296	1.000
		Sig. (2-tailed)	.476	.
		N	8	8

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	-.105
		Sig. (2-tailed)	.	.788
		N	9	9
	Q33 Experience level of team	Correlation Coefficient	-.105	1.000
		Sig. (2-tailed)	.788	.
		N	9	9

Spearman's rho	Q34 Team empowerment	Correlation Coefficient	1.000	.780(*)
		Sig. (2-tailed)	.	.013
		N	9	9
	% Agile (extent of techniques used)	Correlation Coefficient	.780(*)	1.000
		Sig. (2-tailed)	.013	.
		N	9	9

* Correlation is significant at the 0.05 level (2-tailed).

% Agile (extent of techniques used)	Pearson Correlation	1	.258
	Sig. (2-tailed)		.576
	N	9	7
Q35 Customer contacts	Pearson Correlation	.258	1
	Sig. (2-tailed)	.576	
	N	7	7

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	-.443
		Sig. (2-tailed)	.	.232
		N	9	9
	Q36 Communication style within-team	Correlation Coefficient	-.443	1.000
		Sig. (2-tailed)	.232	.
		N	9	9

Spearman's rho	% Agile (extent of techniques used)	Correlation Coefficient	1.000	-.183
		Sig. (2-tailed)	.	.638
		N	9	9
	Q37 Communication style team-management	Correlation Coefficient	-.183	1.000
		Sig. (2-tailed)	.638	.
		N	9	9

% Agile (extent of techniques used)	Pearson Correlation	1	.412
	Sig. (2-tailed)		.271
	N	9	9
Q39 Experience in main role	Pearson Correlation	.412	1
	Sig. (2-tailed)	.271	
	N	9	9

% Agile (extent of techniques used)	Pearson Correlation	1	.411
	Sig. (2-tailed)		.272
	N	9	9
Q41 Experience in business domain	Pearson Correlation	.411	1
	Sig. (2-tailed)	.272	
	N	9	9

% Agile (extent of techniques used)	Pearson Correlation	1	.107
	Sig. (2-tailed)		.784
	N	9	9
Q42 Experience in software development	Pearson Correlation	.107	1
	Sig. (2-tailed)	.784	
	N	9	9

% Agile (extent of techniques used)	Pearson Correlation	1	-.101
	Sig. (2-tailed)		.872
	N	9	5
A Total years of experience	Pearson Correlation	-.101	1
	Sig. (2-tailed)	.872	
	N	5	5

% Agile (extent of techniques used)	Pearson Correlation	1	.372
	Sig. (2-tailed)		.324
	N	9	9
B Total years of experience	Pearson Correlation	.372	1
	Sig. (2-tailed)	.324	
	N	9	9

SD Staff - agile only	Pearson Correlation	1	.940
	Sig. (2-tailed)		.018
	N	5	5
Team size - agile only	Pearson Correlation	.940	1
	Sig. (2-tailed)	.018	
	N	5	5

* Correlation is significant at the 0.05 level (2 tailed).

Appendix R List of Abbreviations

AM	Agile Modeling
ART	Automated Regression Testing
ASD	Adaptive System Development
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
DSDM	Dynamic System Development Method
ETHICS	Effective Technical and Human Implementation of Computer-based systems
FIE	Frontiers in Education Conference
IE	Information Engineering
ISAC	Information Systems work and Analysis of Change
ISD	Internet Speed Development
ISO	International Organization for Standardization
IT	Information Technology
ITiCSE	Innovation and Technology in Computer Science Education
JSD	Jackson Systems Development
KAELOC	Thousands of Assembler Equivalent Lines of Code
KPA	Key Process Area
LD	Lean Development
OOA	Object-oriented Analysis
PP	Pragmatic Programming
RUP	Rational Unified Process
SDLC	System Development Life Cycle
SIGCSE	Association of Computing Machinery's Special Interest Group on Computer Science Education
SSADM	Structured Systems Analysis and Design Method
SSM	Soft Systems Methodology
STRADIS	Structured Analysis, Design and Implementation of Information Systems
SW-CMM	Capability Maturity Model for Software
TDD	Test Driven Development
XP	Extreme Programming
YSM	Yourdan Systems Method

BIBLIOGRAPHY

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods: Review and analysis. *VTT Publications 478.* Retrieved 1 January, 2004, from http:/www.inf.vtt.fi/pdf

Abrahamsson, P., Warsta, J., Siponen, M. K., & Ronkainen, J. (2003). New directions on agile methods: A comparative analysis. In *Proceedings of the 25th International Conference on Software Engineering, ICSE'03* (pp. 244-254). Washington, DC, USA: IEEE Computer Society.

Agile Development Conference. (2005). Retrieved 1 June, 2005, from http://www.agileuniverse.com/home

Agile methodologies: Survey results. (2003). Retrieved 1 March, 2003, from www.ShineTech.com/agile_survey.jsp

Agile Open. (2005). Retrieved 22 June, 2005, from http://www.agileopen.net/

AgileAlliance. (2001). Manifesto for agile software development. Retrieved 17 February, 2003, from http://www.agilemanifesto.org

Aiken, J. (2004). Technical and human perspectives on pair programming. *ACM SIGSOFT Software Engineering Notes, 29*(5), 1-14.

Ambler, S. (2002). *Agile modeling: Effective practices for Extreme Programming and the Unified Process*. New York: John Wiley & Sons, Inc.

Ambu, W., & Gianneschi, F. (2003). Extreme programming at work. In M. Marchesi & G. Succi (Eds.), *4th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2003* (Vol. 2675 Lecture Notes in Computer Science, pp. 298-306). Berlin: Springer-Verlag.

Astrachan, O. L., Duvall, R. C., & Wallingford, E. (2003). Bringing extreme programming to the classroom. In M. Marchesi, G. Succi, D. Wells & L. Williams (Eds.), *Extreme programming perspectives* (pp. 237-250). Boston: Addison-Wesley.

Avison, D. E., & Fitzgerald, G. (1995a). Chapter 7 Methodologies: Issues and frameworks. In *Information systems development: methodologies, techniques and tools* (2 ed., pp. 417-478). London: McGraw-Hill.

Avison, D. E., & Fitzgerald, G. (1995b). *Information systems development: Methodologies, techniques and tools* (2 ed.). London: McGraw-Hill.

Avison, D. E., & Fitzgerald, G. (2003a). Chapter 26: Issues. In *Information systems development: methodologies, tools and techniques* (3 ed.). London: McGraw-Hill.

Avison, D. E., & Fitzgerald, G. (2003b). Chapter 27: Methodology comparisons. In *Information systems development: methodologies, tools and techniques* (3 ed.). London: McGraw-Hill.

Avison, D. E., & Fitzgerald, G. (2003c). *Information systems development: Methodologies, techniques and tools* (3 ed.). London: McGraw-Hill.

Avison, D. E., & Fitzgerald, G. (2003d). Where now for development methodologies? *Communications of the ACM, 46*(1), 78-82.

Baskerville, R. L., Levine, L., Pries-Heje, J., Ramash, B., & Slaughter, S. (2001). How internet companies negotiate quality. *Computer, 34*(5), 51-57.

Baskerville, R. L., & Pries-Heje, J. (2001). Racing the E-bomb: How the Internet is redefining information systems development methodology. In B. Fitzgerald, N. Russo & J. DeGross (Eds.), *Realigning research and practice in IS development* (pp. 49-68). New York: Kluwer.

Baskerville, R. L., & Pries-Heje, J. (2004). Short cycle time systems development. *Information Systems Journal, 14*(3), 237-264.

Baskerville, R. L., Travis, J., & Truex, D. P. (1992). Systems without method: The impact of new technologies on information systems development projects. In K. E. Kendall, K. Lyytinen & J. DeGross (Eds.), *Transactions on the impact of computer supported technologies in information systems development* (pp. 241-260). Amsterdam: Elsevier Science Publications.

Beck, K. (1999). Embracing change with Extreme Programming. *Computer, 32*(10), 70-77.

Beck, K. (2000). *Extreme programming explained: Embrace change*. Boston: Addison-Wesley.

Beck, K., & Fowler, M. (2001). *Planning extreme programming*. Boston: Addison-Wesley.

Becker-Pechau, P., Breitling, H., Lippert, M., & Schmolitzky, A. (2003). Teaching team work: An extreme week for first-year programmers. In M. Marchesi & G. Succi (Eds.), *4th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2003* (Vol. 2675 Lecture Notes in Computer Science, pp. 286-293). Berlin: Springer-Verlag.

Bedoll, R. (2003). A tail of two projects: How 'agile' methods succeeded after 'traditional' methods had failed in a critical system development project, *Extreme Programming and Agile Methods - XP/Agile Universe 2003* (Vol. 2753 Lecture Notes in Computer Science, pp. 25-34). Berlin: Springer-Verlag.

Beedle, M., Devos, M., Sharon, Y., Schwaber, K., & Sutherland, J. (1999). Scrum: A pattern language for hyperproductive software development. In N. Harrison, B. Foote & H. Rohnert (Eds.), *Pattern languages of program design* (pp. 637-651). New York: Addison-Wesley.

Beynon-Davies, P., MacKay, H., & Tudhope, D. (2000). 'It's lots of bits of paper and ticks and post-it notes and things.' a case study of a rapid application development project. *Information Systems Journal, 10*(3), 195-216.

Boehm, B. (1988). A spiral model of software development and enhancement. *Computer, 21*(5), 61-72.

Boehm, B., & Turner, R. (2003). Observations on balancing discipline and agility. *Proceedings of the Agile Development Conference, ADC 2003.* Retrieved 1 June, 2005, from IEEE Xplore database.

Boehm, B., & Turner, R. (2004). *Balancing agility and discipline*. Boston: Addison-Wesley.

Bos, E., & Vriens, C. (2004). An agile CMM. In C. Zannier, H. Erdogmus & L. Lindstrom (Eds.), *4th Conference on Extreme Programming and Agile Methods - XP/Agile Universe 2004* (Vol. 3134 Lecture Notes in Computer Science, pp. 129-138). Berlin: Springer-Verlag.

Boudreau, M., Gefen, D., & Straub, D. W. (2001). Validation in information systems research: A state-of-the-art assessment. *MIS Quarterly, 25*(1), 1.

Brinkkemper, S. (1996). Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology, 38*(4), 275-280.

Broza, G. (2004). Adapting extreme programming to research, development and production environments. In C. Zannier, H. Erdogmas & L. Lindstrom (Eds.), *4th Conference on Extreme Programming and Agile Methods, XP/Agile Universe 2004* (Vol. 3134 Lecture Notes in Computer Science, pp. 139-146). Berlin: Springer-Verlag.

Bunse, C., Feldmann, R. L., & Dorr, J. (2004). Agile methods in software engineering education. In J. Eckstein & H. Baumeister (Eds.), *5th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP2004* (Vol. 3092 Lecture Notes in Computer Science, pp. 284-293). Berlin: Springer-Verlag.

Butler, T., & Fitzgerald, B. (1997). A case study of user participation in the information systems development process, *Proceedings of the eighteenth international conference on Information Systems. Atlanta, Georgia, United States* (pp. 411-426). Atlanta, GA, USA: Association for Information Systems.

Butler, T., & Fitzgerald, B. (2001). The relationship between user participation and the management of change surrounding the development of information systems: A European perspective. *Journal of End User Computing, 13*(1), 12-25.

Cameron, K. S., & Quinn, R. E. (1999). *Diagnosing and changing organizational culture: Based on the competing values framework*. Reading MA: Addison-Wesley.

Cavaye, A. L. M. (1995). User participation in system development revisited. *Information and Management, 28*(5), 311-323.

Charette, R. (2001). The decision is in: Agile versus heavy methodologies. *e-Project Management Advisory Service, 2*(19).

Charette, R. N. (2002). *Foundations of Lean Development: The Lean Development manager's guide* (Vol. 2). Spotsylvania, Va.: ITABHI Corporation.

Checkland, P. (1999). *Systems Thinking, Systems Practice. Soft systems methodology: A 30-year retrospective*. Chichester: John Wiley & Sons, Ltd.

Cockburn, A. (1998). *Surviving object-oriented projects: A manager's guide*. Reading MA: Addison Wesley Longman.

Cockburn, A. (2002). *Agile software development*. Boston: Addison-Wesley.

Cockburn, A. (2003). People and methodologies in software development. Retrieved 17 January, 2004, from http://www.peopleandmethodologiesinsoftwaredevelopment.pdf

Collis, J., & Hussey, R. (2003). *Business research: A practical guide for undergraduate and postgraduate students* (2 ed.). Basingstoke: Palgrave Macmillan.

Conboy, K., & Fitzgerald, B. (2004a). Toward a conceptual framework of agile methods. In C. Zannier, H. Erdogmus & L. Lindstrom (Eds.), *4th Conference on Extreme Programming and Agile Methods XP/Agile Universe 2004* (Vol. 3134 Lecture Notes in Computer Science, pp. 105-116). Berlin: Springer-Verlag.

Conboy, K., & Fitzgerald, B. (2004b). Toward a conceptual framework of agile methods: A study of agility in different disciplines, *Proceedings of the 2004 ACM workshop on Interdisciplinary Software Engineering Research* (pp. 37-44). New York: ACM Press.

Coplien, J., Hutz, S., & Marykuca, B. (1992). Iterative development/OO: The bottom line. *OOPS Messenger, 4*(2), 101-108.

Coplien, J., Hutz, S., & Winder, R. L. (1994). The object paradigm and development process standards. *OOPS Messenger, 5*(2), 99-102.

Crinnion, J. (1992). The evolutionary development of business systems. *IEE Colloquium on Software Prototyping and Evolutionary Development.* Retrieved 1 June, 2005, from IEEE Xplore database.

Cusumano, M., & Selby, R. W. (1997). How Microsoft builds software. *Communications of the ACM, 40*(6), 53-61.

Cusumano, M. A., & Yoffie, D. B. (1999). Software development on Internet time. *Computer, 32*(10), 60-69.

Darke, P., Shanks, G., & Broadbent, M. (1998). Successfully completing case study research: Combining rigour, relevance and pragmatism. *Information Systems Journal, 8*(4), 273-289.

Doll, W. J., & Deng, X. (2001). The colloborative use of information technology: End-user participation and system success. *Information Resources Management Journal, 12*(2), 6-16.

Domino, M., Hevner, A., & Collins, R. W. (2002). Applying agile software development processes to global virtual teams: A study of communication modalities, *Proceedings of the Annual ACM Special Interest Group on Computer Personnel Research Conference, SIGCPR 2002* (pp. 76-78). New York, USA: ACM Press.

Drobka, J., Noftz, D., & Raghu, R. (2004). Piloting XP on four mission-critical projects. *IEEE Software, 21*(6), 70-75.

Dube, L., & Pare, G. (2003). Rigor in information systems positivist case research: Current practice, trends, and recommendations. *MIS Quarterly, 27*(4), 597-635.

Dynamic Systems Development Method, Version 2. (1995). (2 ed.). Ashford: Tesseract Publishing.

Elssamadisy, A. (2003). XP on a large project - a developer's view. In M. Marchesi, G. Succi, D. Wells & L. Williams (Eds.), *Extreme programming perspectives* (pp. 387-397). Boston: Addison-Wesley.

Embley, D. W., Kurtz, B. D., & Woodfield, S. N. (1992). *Object-oriented systems analysis: A model-driven approach*. Englewood Cliffs, NJ.: Yourdan Press/Prentice-Hall.

Eva, M. (1994). *SSADM Version 4: A user's guide* (2 ed.). London: McGraw-Hill Book Company.

Fitzgerald, B. (1997). The use of systems development methodologies in practice: A field study. *Information Systems Journal, 7*(3), 201-212.

Fitzgerald, B. (1998). An empirically-grounded framework for the information systems development process, *Proceedings of the International Conference on Information Systems* (pp. 103-114). Atlanta, GA, USA: Association for Information Systems.

Fitzgerald, B. (2000). Systems development methodologies: The problem of tenses. *Information Technology and People, 13*(3), 174-185.

Fitzgerald, B., & Howcroft, D. (1998). Competing dichotomies in IS research and possible strategies for resolution, *Proceedings of the International Conference on Information Systems* (pp. 155-164). Atlanta, GA, USA: Association for Information Systems.

Fowler, M. (1999). *Refactoring: Improving the design of existing programs*. Boston: Addison-Wesley.

Fraser, S., Astels, D., Beck, K., Boehm, B., McGregor, J., Newkirk, J., et al. (2003). Discipline and practices or TDD (Test Driven Development). In R. Crocker & J. G. L. Steele (Eds.), *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications OOPSLA'03,* (pp. 268-270). New York: ACM.

Fuqua, A. M., & Hammer, J. M. (2003). Embracing change: An XP experience report, *4th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2003* (Vol. 2675 Lecture Notes in Computer Science, pp. 298-306). Berlin: Springer-Verlag.

Gallivan, M. J., & Keil, M. (2003). The user-developer communication process: A critical case study. *Information Systems Journal, 13*(1), 37-68.

George, B., & Williams, L. (2004). A structured experiment of test-driven development. *Information and Software Technology, 46*(5), 337-342.

Glass, R. L. (2004). Practical programmer: Matching methodology to problem domain. *Communications of the ACM, 47*(5), 19-21.

Glass, R. L., & Vessey, I. (1995). Contemporary application domain taxonomies. *IEEE Software, 12*(4), 63-76.

Goulielmos, M. (2004). Systems development approach: Transcending methodology. *Information Systems Journal, 14*(4), 363-386.

Graham, I. (1991). *Object oriented methods*. Harlow, UK: Addison-Wesley.

Graham, I., Henderson-Sellers, B., & Younessi, H. (1997). *The OPEN process specification*. Harlow, England: Addison-Wesley.

Grossman, F., Bergin, J., Leip, D., Merrit, S., & Gotel, O. (2004). One XP experience: Introducing agile (XP) software development into a culture that is willing but not ready. *Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative Research, 252-254.* Retrieved 1 June, 2005, from The ACM Digital Library database.

Groves, L., Nickson, R., Reeve, G., Reeves, S., & Utting, M. (2000). A survey of software development practices in the New Zealand software industry. *Proceedings of the Australian Software Engineering Conference.* Retrieved 1 June 2005, from IEEE Xplore database.

Hevner, A., Collins, R. W., & Garfield, M. J. (2002). Product and project challenges in electronic commerce software development. *ACM SIGMIS Database, 33*(4), 10-22.

Highsmith, J. (2002). *Agile software development ecosystems*. Boston: Addison-Wesley.

Highsmith, J. (2003). Agile software development - why it is hot! In M. Marchesi, G. Succi, D. Wells & L. Williams (Eds.), *Extreme programming perspectives* (pp. 9-16). Boston: Addison-Wesley.

Highsmith, J. A. (2000). *Adaptive software development: A collaborative approach to managing complex systems*. New York, NY: Dorset House Publishing.

Hirschheim, R., & Klein, H. K. (1989). Four paradigms of information systems development. *Communications of the ACM, 32*(10), 1199-1216.

Hofstede, G., Neuijen, B., Daval Ohayv, D., & Sanders, G. (1990). Measuring organizational cultures: A qualitative and quantitative study across twenty cases. *Administrative Science Quarterly, 35*(2), 286-316.

Holcombe, M., Gheorge, M., & Macias, F. (2003). Teaching XP for real: Some initial observations and plans. In M. Marchesi, G. Succi, D. Wells & L. Williams (Eds.), *Extreme programming perspectives* (pp. 251-260). Boston: Addison-Wesley.

Howard, D. (2003). Swimming around the waterfall: Introducing and using agile development in a data centric, traditional software engineering company. In M. Marchesi & G. Succi (Eds.), *4th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2003* (Vol. 2675 Lecture Notes in Computer Science, pp. 138-145). Berlin: Springer-Verlag.

Hulkko, H., & Abrahamsson, P. (2005). A multiple case study on the impact of pair programming on product quality, *Proceedings of the 27th International Conference on Software Engineering* (pp. 495-504). New York: ACM Press.

Hunt, A., & Thomas, D. (2000). *The pragmatic programmer*. Boston: Addison Wesley.

Ibba, A., & Ohlemeyer, C. (2003). A framework for testing at the edge - an experience report. In M. Marchesi & G. Succi (Eds.), *Proceedings of the 4th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2003* (Vol. 2675 Lecture Notes in Computer Science, pp. 198-204). Berlin: Springer-Verlag.

Iivari, J., Hirschheim, R., & Klein, H. K. (1999). Beyond methodologies: Keeping up with information systems development approaches through dynamic classification. *Proceedings of the 32nd Hawaii International Conference on System Sciences, HICSS-32*. Retrieved 1 June, 2005, from IEEE Xplore database.

Iivari, J., Hirschheim, R., & Klein, H. K. (2001). A dynamic framework for classifying information systems development methodologies and approaches. *Journal of Management Information Systems, 17*(3), 179-218.

Iivari, J., Hirschheim, R., & Klein, H. K. (2004). Towards a distinct body of knowledge for Information Systems experts: Coding ISD process knowledge in two IS journals. *Information Systems Journal, 14*(4), 313-342.

Iivari, J., & Kerola, P. (1983). A sociocybernetic framework for the feature analysis of information systems design methodologies. In T. W. Olle, H. C. Sol & C. Tully (Eds.), *Information systems design methodologies: A feature analysis. Proceedings of the IFIP WB 8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies* (pp. 87-139). Amsterdam: North-Holland.

International Conference on eXtreme Programming and Agile Processes in Software Engineering. (2005). Retrieved 22 June, 2005, from http://www.xp2005.org/

Jackowski, Z. (2003). Metamodel of a system development method. Retrieved 23 November, 2003, from www.agilealliance.com/articles/articles/metamodelOfSDM.pdf

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The unified software development process*. Reading, Massachusetts: Addison-Wesley.

Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. (1992). *Object-oriented software engineering: A use case driven approach*. Harlow, UK: Addison-Wesley.

Jayaratna, N. (1994). *Understanding and evaluating methodologies NIMSAD: A systematic framework*. London: McGraw-Hill.

Johnson, D. H., & Caristi, J. (2003). Extreme programming and the software design course. In M. Marchesi, G. Succi, D. Wells & L. Williams (Eds.), *Extreme programming perspectives* (pp. 273-285). Boston: Addison-Wesley.

Kahkonen, T., & Abrahamson, P. (2003). Digging into the fundamentals of extreme programming: Building the theoretical base for agile methods. *Proceedings of the 29th Euromicro Conference, 1-6 Sept.* Retrieved 1 June, 2005, from IEEE Xplore database.

Keil, M., & Carmel, E. (1995). Customer-developer links in software development. *Communications of the ACM, 38*(5), 33-44.

Koskela, J., & Abrahamson, P. (2004). On-site customer in an XP project: Empirical results from a case study. In T. Dingsoyr (Ed.), *11th European Conference on Software Process Improvement, EuroSPI 2004* (Vol. 3281 Lecture Notes in Computer Science, pp. 1-11). Berlin: Springer-Verlag.

Krebs, J. (2005). RUP in the dialogue with Scrum. *developerWorks.* Retrieved 6 June, 2005, from http://www-106.ibm.com/developerworks/rational/library/feb05/krebs/index.html

Kroenke, D. (1992). *Management Information Systems*. New York: McGraw-Hill.

Kruchten, P. (2000). *The Rational Unified Process: An introduction* (2 ed.). Boston: Addison-Wesley Longman.

Kuhn, S., & Muller, M. J. (1993). Participatory design. *Communications of the ACM, 36*(4), 25-28.

Kumar, K., & Welke, R. J. (1992). Methodology engineering: A proposal for situation-specific methodology construction. In W. W. Cotterman & J. A. Senn (Eds.), *Challenges and strategies for research in systems development* (pp. 257-269). New York: John Wiley & Sons.

Kuppuswami, S., Vivekanandan, K., Ramaswamy, P., & Rodrigues, P. (2003). The effects of individual XP practices on software development effort. *ACM SIGSOFT Software Engineering Notes, 28*(6), 1-6.

Larman, C. (2002). *Applying UML and patterns: An introduction to object-oriented analysis and design and the Unified Process*. Upper Saddle River, NJ: Prentice Hall.

Layman, L., Williams, L., & Cunninghan, L. (2004). Exploring extreme programming in context: An industrial case study. *Agile Development Conference, ADC 2004.* Retrieved 1 June, 2005, from IEEE Xplore database.

Lengel, R. H., & Daft, R. (1988). The selection of communication media as an executive skill. *The Academy of Management Executive, 2*(3), 225-232.

Lewis, P. J. (1994). *Information systems development: Systems thinking in the field of IS*. London: Pitman.

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., et al. (2004). Agile software development in large organizations. *Computer, 37*(12), 26-33.

Lytinnen, K., & Rose, G. M. (2003a). Disruptive information system innovation: The case of internet computing. *Information Systems Journal, 13*(4), 301-330.

Lytinnen, K., & Rose, G. M. (2003b). The disruptive nature of information technology innovations: The case of internet computing in systems development organisations. *MIS Quarterly, 27*(4), 557-595.

Maddison, R. N. (Ed.). (1983). *Information System Methodologies*. Chichester: Wiley Heyden.

Mar, K., & Schwaber, K. (2002). Scrum with XP. Retrieved 6 June, 2005, from http://www.informit.com/articles/article.asp?p=26057

Marchesi, M., & Succi, G. (Eds.). (2003). *Proceedings of the 4th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2003* (Vol. 2675 Lecture Notes in Computer Science). Berlin: Springer-Verlag.

Markus, M. L., & Bjorn-Anderson, N. (1987). Power over users: Its exercise by system professionals. *Communications of the ACM, 30*(6), 498-504.

Martin, A., Biddle, R., & Noble, J. (2004). The XP customer role in practice: Three studies. *Agile Development Conference, ADC2004.* Retrieved 1 June, 2005, from IEEE Xplore database.

Martin, A., Noble, J., & Biddle, R. (2003). Being Jane Malkovich: A look into the world of an XP customer. In M. Marchesi & G. Succi (Eds.), *4th International Conference on Extreme Programming and Agile Processes, XP 2003* (Vol. 2675 Lecture Notes in Computer Science, pp. 234-243). Berlin: Springer.

Martin, J. (1991). *Rapid Application Development*. Englewood Cliffs, NJ: Prentice Hall.

Martin, J., & Finkelstein, C. (1981). *Information Engineering. Vol 1 and 2*. Englewood Cliffs, New Jersey: Prentice Hall.

Martin, R. C. (1998). *The process: Object oriented analysis and design with applications*. Boston: Addison-Wesley.

Martinsson, J. (2003). Maturing XP through the CMM. In M. Marchesi & G. Succi (Eds.), *Proceedings of the 4th International Conference on Extreme Programming and Agile Processes, XP 2003* (Vol. 2675 Lecture Notes in Computer Science). Berlin: Springer-Verlag.

Mathison, S. (1988). Why triangulate? *Educational Researcher, 17*(2), 18-17.

Maurer, F. (2002). Supporting distributed extreme programming, *Extreme Programming and Agile Methods, XP/Agile Universe 2002: Second XP Universe and First Agile Universe Conference, August 4-7* (Vol. 2418 Lecture Notes in Computer Science, pp. 13-22). Berlin: Springer-Verlag.

Maximilien, E. M., & Williams, L. (2003). Assessing test-driven development at IBM, *Proceedings of the 25th International Conference on Software Engineering* (pp. 564-569). Washington DC: IEEE Computer Society.

McKeen, J. D., Guimaraes, T., & Wetherbe, J. C. (1994). The relationship between user participation and user satisfaction: An investigation of four contingency factors. *MIS Quarterly, 18*(4), 427-451.

McKinney, D., Froeseth, J., Robertson, J., Denton, L. F., & Ensminger, D. (2004). Agile CS1 labs: Extreme programming practices in an introductory programming course. In C. Zannier, H. Erdogmas & L. Lindstrom (Eds.), *4th Conference on Extreme Programming and Agile Methods, XP/Agile Universe 2004* (Vol. 3134 Lecture Notes in Computer Science, pp. 164-174). Berlin: Springer-Verlag.

Melnik, G., & Maurer, F. (2005). A cross-program investigation of student's perceptions of agile methods, *27th International Conference on Software Engineering* (pp. 481-488). New York: ACM Press.

Mens, T., & Tourwe, T. (2004). A survey of software refactoring. *IEEE Transactions on Software Engineering, 30*(2), 126-139.

Morkel Theunissen, W. H., Kourie, D. K., & Watson, B. W. (2003). Standards and agile software development, *Proceedings of the 2003 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement through Technology (SAICSIT), ACM International Conference Proceeding Series; Vol. 47* (pp. 178-188). Republic of South Africa: South African Institute for Computer Scientists and Information Technologists.

Mugridge, R., MacDonald, B., & Roop, P. (2003a). A customer test generator for web-based systems. In M. Marchesi & G. Succi (Eds.), *4th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2003* (Vol. 2675 Lecture Notes in Computer Science, pp. 189-197). Berlin: Springer-Verlag.

Mugridge, R., MacDonald, B., Roop, P., & Tempero, E. (2003b). Five challenges in teaching XP. In M. Marchesi & G. Succi (Eds.), *4th International Conference on Extreme Programming and Agile Processes, XP 2003* (Vol. 2675 Lecture Notes in Computer Science, pp. 406-409). Berlin: Springer-Verlag.

Muller, M. M. (2004). Are reviews an alternative to pair programming? *Empirical Software Engineering, 9*(4), 335-351.

Muller, M. M., & Hagner, O. (2002). Experiment about test-first programming. *IEE Proceedings Software, 149(5), 131-136.* Retrieved 1 June, 2005, from IEEE Xplore database.

Muller, M. M., & Padberg, F. (2003). On the economic evaluation of XP projects, *Proceedings of the 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ESEC/FSE-11* (Vol. 28(5), pp. 168-177). New York: ACM Press.

Murru, O., Deias, R., & Mugheddu, G. (2003). Assessing XP at a European Internet company. *IEEE Software, 20*(3), 37-43.

Nawrocki, J., Jasinski, M., Walter, B., & Wojciechowski, A. (2002a). Combining extreme programming with ISO 9000. In M. H. Shafazand & A. M. Tjoa (Eds.), *Proceedings of the First EurAsian Conference on Information and Communication Technology, EurAsia-ICT 2002* (Vol. 2510 Lecture Notes in Computer Science, pp. 786-794). Berlin: Springer-Verlag.

Nawrocki, J., Jasinski, M., Walter, B., & Wojciechowski, A. (2002b). Extreme programming modified: Embrace requirements engineering practices. *IEEE Joint International Conference on Requirements Engineering, RE'02, 303-310.* Retrieved 3 June, 2005, from IEEE Xplore database.

Nawrocki, J., Walter, B., & Wojciechowski, A. (2001). Toward maturity model for extreme programming. *Proceedings of the 27th Euromicro Conference, 233-239.* Retrieved 2 June, 2005, from IEEE Xplore database.

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM, 48*(5), 73-78.

Nickell, E., & Smith, I. (2003). Three patterns in Java unit testing. In M. Marchesi & G. Succi (Eds.), *4th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2003* (Vol. 2675 Lecture Notes in Computer Science, pp. 170-179). Berlin: Springer-Verlag.

Noble, J., Marshell, S., Marshall, S., & Biddle, R. (2004). Less Extreme Programming, *Proceedings of the Sixth Conference on Australasian Computing Education - ACE'04* (Vol. 30, pp. 217-226). Darlinghurst, Australia: Australian Computer Society.

Olle, T. W., Sol, H. C., & Tully, C. (Eds.). (1983). *Information systems design methodologies: A feature analysis. Proceedings of the IFIP WB 8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies.* Amsterdam: North-Holland.

Paige, R. F., Chivers, H., McDermid, J. A., & Stephenson, Z. R. (2005). High-integrity extreme programming, *Proceedings of the 2005 ACM Symposium on Applied Computing* (pp. 1518-1523). New York: ACM Press.

Palmer, S. R., & Felsing, J. M. (2002). *A practical guide to Feature-Driven Development*. Upper Saddle River: Prentice Hall.

Paparone, C. R. (2003). *Applying the competing values framework to study organizational subcultures and system-wide planning efforts in a military university.* Unpublished Doctor of Philosophy, The

Pennsylvania State University. Retrieved 1 June, 2006, from http://etda.libraries.psu.edu/theses/approved/WorldWideIndex/ETD-316/.

Paulk, M. C. (2001). Extreme programming from a CMM perspective. *IEEE Software, 18*(6), 19-26.

Pollice, G. (2001). RUP and XP, Part 1: Finding common ground. *The Rational Edge.* Retrieved 6 June, 2005, from http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/archives/rup.html

Poppendiek, M., & Poppendiek, T. (2003). *Lean software development an agile toolkit*. Boston: Addison-Wesley.

Pressman, R. S. (2005). *Software engineering: A practitioner's approach* (6 ed.). Boston: McGraw-Hill Higher Education.

Reeves, M., & Zhu, J. (2004). Moomba - a collaborative environment for supporting distributed extreme programming in global software development. In J. Eckstein & H. Baumeister (Eds.), *Extreme Programming and Agile Processes in Software Engineering, XP 2004* (Vol. 3092 Lecture Notes in Computer Science, pp. 38-50). Berlin: Springer-Verlag.

Reifer, D. J. (2002a). How good are agile methods? *IEEE Software, 19*(4), 16-18.

Reifer, D. J. (2002b). How to get the most out of extreme programming/agile methods. In D. Wells & L. Williams (Eds.), *Extreme Programming and Agile Methods - XP/Agile Universe 2002: Second XP Universe and First Agile Universe Conference* (Vol. 2418 Lecture Notes in Computer Science, pp. 185-196). Berlin: Springer-Verlag.

Reifer, D. J. (2003). XP and the CMM. *IEEE Software, 20*(3), 14-15.

Reifer, D. J., Maurer, F., & Erdogmus, H. (2003). Scaling agile methods. *IEEE Software, 20*(4), 12-14.

Rising, L., & Janoff, N. S. (2000). The Scrum software development process for small teams. *IEEE Software, 17*(4), 26-32.

Roberts, T. L., Leigh, W., & Purvis, R. L. (2000). Perceptions on stakeholder involvement in the implementation of system development methodologies. *The Journal of Computer Information Systems, 40*(3), 78-83.

Royce, W. W. (1987). Managing the development of large software systems, *Proceedings of the 9th international conference on Software Engineering*. Los Alamitos, CA: IEEE Computer Society Press (Reprinted from Proceedings, IEEE WESCON, August 1970 p. 1-9. Originally published by TRW).

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The Unified Modeling Language reference manual*. Massachusetts: Addison-Wesley.

Rzevski, G. (1983). On the comparison of design methodologies. In T. W. Olle, H. G. Sol & C. J. Tully (Eds.), *Information systems design methodologies: A feature analysis. Proceedings of the IFIP WB 8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies* (pp. 258-266). Amsterdam: North-Holland.

Salo, O., & Abrahamson, P. (2004). Empirical evaluation of agile software development: The controlled case study approach. In F. Bomarius & H. Iida (Eds.), *5th International Conference on Product Focused Software Process Improvement, PROFES 2004* (Vol. 3009 Lecture Notes in Computer Science, pp. 408-423). Berlin: Springer-Verlag.

Sanders, D. (2003). Student perceptions of the suitability of extreme and pair programming. In M. Marchesi, G. Succi, D. Wells & L. Williams (Eds.), *Extreme programming perspectives* (pp. 261-272). Boston: Addison-Wesley.

Schalliol, G. (2003). Challenges for analysts on a large XP project. In M. Marchesi, G. Succi, D. Wells & L. Williams (Eds.), *Extreme programming perspectives* (pp. 375-385). Boston: Addison-Wesley.

Schneider, J., & Johnston, L. (2003). Extreme programming in universities: An educational perspective, *Proceedings of the 25th International Conference on Software Engineering* (pp. 594-599). Washington, DC: IEEE Computer Society.

Schummer, T., & Schummer, J. (2001). Support for distributed teams in extreme programming. In G. Succi & M. Marchesi (Eds.), *Extreme programming examined* (pp. 355-377). Boston: Addison-Wesley Longman.

Schwaber, K. (1995). SCRUM software development process. Retrieved 3 October, 2004, from http://www.controlchaos.com/old-site/scrumwp.htm

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*. Upper Saddle River, New Jersey: Prentice Hall.

Sharma, R., Sugumaran, V., & Rajagopalan, B. (2002). A framework for creating hybrid-open source software communities. *Information Systems Journal, 12*(1), 7-25.

Simmonds, M., & Fazackerley, B. (2003). EnterpriseXP: Can the combination of XP and DSDM improve the appeal of XP to the business community? In M. Marchesi & G. Succi (Eds.), *4th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2003* (Vol. 2675 Lecture Notes and Computer Science, pp. 337-339). Berlin: Springer-Verlag.

Smrtic, M. B., & Grinstein, G. (2004). A case study in the use of extreme programming in an academic environment. In C. Zannier, H. Erdogmas & L. Lindstrom (Eds.), *4th Conference on Extreme Programming and Agile Methods, XP/Agile Universe 2004:* (Vol. 3134 Lecture Notes in Computer Science, pp. 175-182). Berlin: Springer-Verlag.

Sol, H. C. (1983). A feature analysis of information systems design methodologies: Methodological considerations. In W. Olle, H. G. Sol & C. Tully (Eds.), *Information systems design methodologies: A feature analysis. Proceedings of the IFIP WB 8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies*. Amsterdam: North-Holland.

Spayd, M. K. (2003). Evolving agile in the enterprise: Implementing XP on a grand scale. *Proceedings of the Agile Development Conference, ADC 2003, 60-70.* Retrieved 4 June, 2005, from IEEE Xplore database.

Stapleton, J. (1997). *DSDM Dynamic Systems Development Method*. Harlow, England: Addison-Wesley.

Trimmer, K. J., Collins, R. W., Will, R., & Blanton, J. E. (2000). Information systems development: Can there "good" conflict? *Proceedings of the ACM SIGCPR conference on Computer Personnel Research* (pp. 174-179). New York, NY, USA: ACM Press.

Truex, D. P., Baskerville, R. L., & Klein, H. K. (1999). Growing systems in emergent organisations. *Communications of the ACM, 42*(8), 117-123.

Truex, D. P., Baskerville, R. L., & Travis, J. (2001). Amethodological systems development: The deferred meaning of systems development methods. *Accounting, Management and Information Technology, 10*, 53-79.

Turk, D., France, R., & Rumpe, B. (2002). Limitations of Agile Software Processes. *Proceedings of the Third International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2002.* Retrieved 1 June, 2005, from http://www.agilealliance.org/articles/turkdanfrancerobertru/file

Vanhanen, J., Jartti, J., & Kahkonen, T. (2003). Practical experiences of agility in the Telecom industry. In M. Marchesi & G. Succi (Eds.), *4th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2003* (Vol. 2675 Lecture Notes in Computer Science, pp. 279-287). Berlin: Springer-Verlag.

Visconti, M., & Cook, C. R. (2004). An ideal process model for agile methods. In F. Bomarius & H. Iida (Eds.), *5th International Conference on Product Focused Software Process Improvement, PROFES 2004* (pp. 431-441). Berlin: Springer-Verlag.

Wake, W., & Wake, S. (2003). The system metaphor explored. In M. Marchesi, G. Succi, D. Wells & L. Williams (Eds.), *Extreme programming perspectives* (pp. 35-39). Boston: Addison-Wesley.

Wallace, J., Hunt, J., & Richards, C. (1999). The relationship between organisational culture, organisational climate and managerial values. *The International Journal of Public Sector Management, 12*(7), 548-564.

Wasserman, A. I., Freeman, P., & Porcella, M. (1983). Characteristics of software development methodologies. In T. W. Olle, H. G. Sol & C. J. Tully (Eds.), *Information systems design methodologies: A feature analysis. Proceedings of the IFIP WB 8.1 Working Conference on Feature Analysis of Information Systems Design Methodologies* (pp. 37-62). Amsterdam: North-Holland.

Wendorff, P. (2002). Organisational culture in agile software development. In M. Oivo & K. Komi-Sirvio (Eds.), *14th International Conference on Product Focused Software Process Improvement, PROFES 2002* (Vol. 2559 Lecture Notes in Computer Science, pp. 145-157). Berlin: Springer-Verlag.

Wengraf, T. (2003). *Qualitative Research Interviewing*. London: Sage Publications.

Williams, L., Kerbs, W., & Layman, L. (2004a). Extreme programming evaluation framework for object-oriented languages version 1.3. *Technical Report TR-2004-11.* Retrieved 20 May, 2004, from http://www.csc.ncsu.edu/research/tech/reports.php

Williams, L., & Kessler, R. (2001). Experimenting with industry's "pair programming" model in the computer science classroom. *Computer Science Education, 11*(1), 7-20.

Williams, L., Krebs, W., Layman, L., & Anton, A. (2004b). Toward a framework for evaluating Extreme Programming. *Technical Report TR-2004-2.* Retrieved 15 January, 2004, from http://www.csc.ncsu.edu/research/tech/reports.php

Williams, L., Layman, L., Krebs, W., & Anton, A. I. (2004c). Exploring the use of a "Safe Subset" of extreme programming: An industrial case study. *Technical Report TR-2004-3.* Retrieved 20 May, 2005, from http://www.csc.ncsu.edu/research/tech/reports.php

Wirfs-Brock, R. J., & Johnson, R. E. (1990). Surveying current research in object-oriented design. *Communications of the ACM, 3*(9), 104-124.

Wirfs-Brock, R. J., & Wilkerson, B. (1989). Object-oriented design: A responsibility-driven approach, *Object Oriented Programming, Systems, Languages and Applications* (pp. 71-75). New York: ACM Press.

Wohlin, C., Host, M., & Henningsson, K. (2003). Empirical research methods in software engineering. In *Empirical Methods and Studies in Software Engineering* (Vol. 2765 Lecture Notes in Computer Science, pp. 7-23). Berlin: Springer-Verlag.

Woit, D. M. (2005). Requirements interaction management in an extreme programming environment: A case study, *27th International Conference on Software Engineering, ICSE 2005* (pp. 489-494). New York: ACM Press.

Wood-Harper, A. T., & Fitzgerald, G. A. (1982). A taxonomy of current approaches to systems analysis. *The Computer Journal, 25*(1), 12-16.

Wood, W. A., & Kleb, W. L. (2003). Exploring XP for scientific research. *IEEE Software, 20*(3), 30-36.

Wright, G. (2003). Achieving ISO 9001 certification for an XP company, *Extreme Programming and Agile Methods - XP/Agile Universe 2003* (Vol. 2753 Lecture Notes in Computer Science, pp. 43-50). Berlin: Springer-Verlag.

Wynekoop, J. L., & Russo, N. L. (1995). Systems development methodologies: Unanswered questions. *Journal of Information Technology, 10*(2), 65-73.

XP @ Scrum. Retrieved 6 June, 2005, from http://www.controlchaos.com/about/xp.php

XP Agile Universe. (2005). Retrieved 22 June, 2005, from http://www.agileuniverse.com/home

Yin, R. K. (2003). *Case study research* (3 ed.). Thousand Oaks: Sage Publications.

Yuan, J., Holcombe, M., & Gheorghe, M. (2003). Where do unit tests come from? In M. Marchesi & G. Succi (Eds.), *4th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2003* (Vol. 2675 Lecture Notes in Computer Science, pp. 161-179). Berlin: Springer-Verlag.