



Technical Report N° 2007/11

Investigating the Adoption of Agile Software
Development Methodologies in Organisations.

Antony R. Grinyer

9th July 2007
Student Research Proposal

Department of Computing
Faculty of Mathematics and Computing
The Open University
Walton Hall,
Milton Keynes
MK7 6AA
United Kingdom

<http://computing.open.ac.uk>



Technical Report N° 2007/11

Investigating the Adoption of Agile Software
Development Methodologies in Organisations.

Antony R. Grinyer

9th July 2007

***Department of Computing
Faculty of Mathematics and Computing
The Open University
Walton Hall,
Milton Keynes
MK7 6AA
United Kingdom***

<http://computing.open.ac.uk>

PROBATION REPORT



Antony Grinyer (W5573116)
Department of Mathematics & Computing
The Open University
24 March 2006

“Investigating the adoption of agile software development methods in organisations”

Research Supervisors

Dr J.A. Segal
Dr H.C. Sharp

Document submitted in part fulfilment for the
requirements of the research degree
probationary viva.

Table of Contents

PART I	5
CONTEXT	5
MOTIVATION	8
AIMS	9
RESEARCH QUESTIONS	9
CONTRIBUTION OF RESEARCH	9
PART II	10
LITERATURE REVIEW	10
A Brief History of Traditional Software Development Approaches	10
A Brief History of Agile Software Development Approaches	13
The Agile Software Development Manifesto	13
What is Agility?	14
Agile Methodologies	15
The Claimed Benefits and Constraints of Agile Approaches	17
Introduction	17
Extreme Programming (XP) – The Most Popular Agile Methodology	17
Controversial Practices	18
Metaphor	18
Simple Design versus Big Design Up Front	19
Pair Programming	20
On-site Customer	21
Constraints	22
Scalability	23
Locality	24
Skill and Talent	26
Relationships and Trust	27
Customer–Developer Relationship	28
Developer–Developer Relationship	29
Manager-Developer Relationship	30
Culture and Agile Approaches	30
Software Development as a Social Activity	33
The Ongoing Debate	34
Rogers’ Model of Diffusion of Innovations and the Adoption of Agile Methods	37
Methodology	41
Summary	42

PART III	43
PRELIMINARY RESEARCH FINDINGS	43
The Type of Evidence Produced by Empirical Software Engineers	46
Why Do Organisations Adopt Agile Methods? An Initial Study	47
PART IV	49
RESEARCH APPROACH	49
Summary of Research Methodology	49
Data Triangulation	53
Sample Size	55
Scope	56
Cross Organisation Approach versus Sequential Organisation Approach	57
Organisational Context	57
Pilot Studies	58
Data Collection Methods & Analysis	59
Questionnaires	59
Open Questions	60
Closed Questions	60
Semi-structured Interviews	61
Observations	62
Coding	63
Provisional Codes	64
Descriptive Codes	64
Inferential Codes	64
Credibility and Trustworthiness	64
Audio/Video Recording	65
Interpretation	65
Audit Trail	65
Ethical Considerations	66
Potential Candidates for Research	66
RESEARCH TIME PLAN	68
Task Units	68
Task Unit Gantt Chart	74
Task Unit Gantt Chart Data	75

REFERENCES	76
APPENDICES	84
Appendix A Experience Reports of Agile Methodology Adoption	84
Appendix B The Twelve Principles of The Agile Manifesto	87
Appendix C The Twelve Practices of Extreme Programming	88
Appendix D Segal, J., Grinyer, A. and Sharp, H. (2005). <u>The type of evidence produced by empirical software engineers</u> . Proceedings of the REBSE workshop, associated with ICSE 2005, St. Louis, Missouri, USA.	90
Appendix E Grinyer, A., Segal, J. and Sharp, H. (2006). <u>Why do organisations adopt agile methods? An initial study</u> . Submitted to XP2006, Oulu, Finland.	97

PART I

In PART I of this report, I discuss the context of the research; the motivations for and aims of the research; the research questions, and finally the potential contribution of the research.

CONTEXT

The escalating pace of the computer industry and the ever-growing demands from customers to deliver software faster, and to the highest quality, has placed software practitioners under increasing pressure to seek ways of rapidly addressing consumer expectations. Prior to the turn of the century, long-standing *traditional* approaches to software development (see, for example, Royce, 1970; DeMarco, 1979; Boehm, 1988) had provided most of the models and management heuristics for developing software in industry; however, these approaches, often referred to as heavy-weight, rigorous, or plan-driven methods (Lindvall et al., 2004) have been challenged by new and emerging *light* (Yourdon, 2002) approaches to software development. In less than a decade, these light-weight approaches have evolved to become collectively known as *Agile* or *Lean* methodologies (Cockburn, 2002; Highsmith, 2002b; Poppendieck and Poppendieck, 2003). The rise in popularity of agile methods within the software engineering community signifies a reaction to traditional approaches, where speed and responding to change are considered more important than documentation and planning. However, agile methods have met varying resistance from traditionalists and software practitioners who have reacted cautiously to some of the underlying principles, practices, and values offered by agile approaches (see, for example, Boehm, 2002; Elssamadisy and Schalliol, 2003; DeMarco and Boehm, 2002).

A new ideology encapsulated in *The Agile Software Development Manifesto* emerged in 2001 from the shared beliefs between advocates of lighter software development approaches. The principles of the Manifesto captured the commonality of values between proponents of Extreme Programming (XP) (Beck, 1999), Scrum (Schwaber and Beedle, 2001), Dynamic Systems Development Method (DSDM) (Stapleton, 1997), Feature Driven Development (FDD) (Palmer and Felsing, 2002), Crystal Methods (Cockburn, 2004), Adaptive Software Development (Highsmith, 1999), and others seeking alternative approaches to traditional software development methods. The values espoused in the Agile Manifesto have been operationalised in these methods, and consequently, many of the practices in each method have, to some extent, been intended to capture one or more of the values of the Manifesto.

Agile methods may still be viewed as immature with respect to the history of software engineering. At present, most of the current literature on agile methods comes from advocates giving subjective accounts of their own experiences or making claims as to the suitability of agile approaches (see, for example, Cockburn, 2002; Highsmith, 2002). Despite the growing number of agile experience reports emerging from organisations, (see, for example, Appendix 1), very few empirical studies of agile methods within organisations have been conducted. Lindvall et al. state:

'Anecdotal evidence is rising regarding the effectiveness of agile methodologies in certain environments and for specified projects. However, collection and analysis of empirical evidence of this effectiveness and classification of appropriate environments for agile projects has not been conducted' (Lindvall et al., 2002:1).

This paucity of empirical investigation has provoked some speculation among researchers relating to how, and why, organisations choose to adopt agile methods. Segal speculates the growth of agile methods is based on gut reaction:

'...we speculate that the rapid growth of agile methodologies is not due to the presentation of any hard evidence (of which there is currently not very much), but to practitioners' gut reactions on hearing of others experiences...' (Segal, 2003:44).

However, Dyba et al. believe organisations might adopt new technologies (such as new software development methods) as a result of pressure:

'Software companies are often under pressure to adopt immature technologies because of market and management pressures' (Dyba et al., 2005:59).

Despite awareness of the dearth of evidence surrounding adoption of agile methods, Ambler believes it may take the research community many years to gather evidence to support, or indeed refute, the techniques related to agile methods:

'It will take several years, perhaps even a decade, until we have incontrovertible proof that agile software development techniques work in practice. The research community clearly has its work cut out for it, and I

hope that the old guard allows and encourages these new lines of research' (Ambler, 2003).

Agile methods such as XP continue to diffuse throughout the software engineering community, yet little is understood how and why this diffusion is sustained. Moreover, the dearth of empirical evidence elucidating the benefits and constraints of agile methods raises questions as to what sources of information organisations use in order to decide whether to adopt agile methods; the factors which persuade organisations to adopt agile methods, and the type of evidence persuasive to practitioners in those organisations.

Rogers (2003) surveyed many studies of the adoption of many different innovations and found that the subjective evaluations of near-peers, and not scientific evidence, had the most effect; he states:

'Most individuals evaluate an innovation [in deciding whether or not to adopt] not on the basis of scientific research by experts but through the subjective evaluations of near peers who have adopted the innovation' (Rogers, 2003:36).

Furthermore, Rainer et al. state (2005):

'The evidence that practitioners value appears to be very different to the evidence that researchers value. In particular, practitioners seem to value local expertise in contrast to independent empirical evidence...' (Rainer et al., 2005:3).

And with the statement from Zelkowitz (2003) that

'...the industrial community is generally wary of laboratory research results...' (Zelkowitz et al., 2003:255).

MOTIVATION

The motivation for this research arises partly from empirical software engineering. Despite the efforts of empirical software engineers, there is concern (Segal et al., 2005) in the field of empirical software engineering that the investigations and results produced by empirical software engineers are having little impact on practice. The reasons for this are speculative and attempts to understand this shortfall have resulted in empirical software engineers questioning why investigations have little influence on practice. Possible reasons are that empirical software engineers are not looking at those aspects of software engineering that are of interest to practitioners, that results of investigations are not communicated suitably to practitioners, and the type of evidence produced by empirical software engineers is not persuasive to practitioners.

In order to influence practice, empirical software engineers need to understand the decision processes within practice and the type of evidence practitioners' draw upon and accept. The rise in popularity of agile methods in the software development community is an attractive case in point. Agile methods have emerged entirely within the practitioner community, and the adoption of agile methods by organisations has not been as a result of weighing evidence of the type produced by empirical software engineers because, until recently, there hasn't been any.

Therefore, the motivations are further based on the desire to better understand the decision making process of practitioners, and to compare results of the investigation of agile methodology adoption in organisations with the established Rogers' model of diffusion of innovations.

The motivations are summarised as follows:

- 1) To inform the academic and empirical software engineering community of the type of evidence persuasive to practitioners, thus enabling empirical software engineers to conduct investigations and produce evidence more relevant to practitioners' needs.
- 2) To articulate the decision making processes of practitioners.
- 3) To understand and enhance a general model of diffusion in the context of the adoption of agile software development methods, with the hope of enhancing understanding with respect to technology adoption within the software development community.

AIMS

The proposed research aims are as follows:

- 1) To investigate the factors that influence organisations to adopt agile methods, in order to explicate the decision-making processes of practitioners.
- 2) To identify the sources and type of evidence that persuades practitioners to select a new approach.
- 3) To compare the adoption of agile methods with the established Rogers' model of diffusion.

RESEARCH QUESTIONS

The overall research statement - 'Investigating the adoption of agile software development methods in organisations' - can be broken down into three main questions in accordance with the three main aims:

- 1) What are the factors that influence organisations to adopt agile methods?
- 2) What are the sources and types of evidence that practitioners use in order to decide whether or not to adopt agile methods?
- 3) How does the adoption of agile methods compare with the established Rogers' model of diffusion?

CONTRIBUTION OF RESEARCH

It is hoped that the proposed research questions will:

- 1) Inform both the academic and empirical software engineering communities of the decision making processes and types of evidence practitioners draw upon.
- 2) Help to inform empirical software engineers of the type of results and evidence most useful and effective to practitioners.
- 3) Enhance understanding of a general model of diffusion using the adoption of agile methods as a case in point.

PART II

LITERATURE REVIEW

In PART II of this report, I review the current literature with respect to the proposed research area. Firstly, I provide a brief discussion of the history of traditional and agile software development methods; secondly, I describe the claimed benefits and constraints of agile methods in the literature with particular focus on XP; I then describe Rogers' established model of diffusion, and finally, I summarise the findings from the literature.

A Brief History of Traditional Software Development Approaches

Over the last 35 years, different approaches to software development have emerged in an effort to address the complex processes associated with managing the development of software systems. In his seminal paper, 'Managing the Development of Large Software Systems: Concepts and Techniques' (Royce, 1970), Winston Royce proposed an approach to developing software to address the convoluted procedures associated with managing large software systems. The ideas and techniques described in Royce's paper seeded the evolution of a long-standing paradigm that became commonly known as the Waterfall Model. The objectives of the Waterfall Model were to thoroughly analyse, specify, and implement user requirements with a set of clearly defined phases including requirements specification, analysis, design, code, testing, and maintenance (Pressman, 2005:47). These steps included meticulous upfront documentation in all phases of the software development lifecycle, enabling programmers to build software in a predictable and regimented manner. Furthermore, with exhaustive upfront analysis and specification, the Waterfall Model attempted to prevent changing requirements throughout the lifecycle so that requirements were frozen, and functionality signed off by the customer to enable the next phase of the software project to progress.

This continuous, 'no going back' approach to software development provided the analogy to give the Waterfall Model its name, whereby water cascading over a waterfall could not go back on itself; thus, once a phase of the lifecycle was signed off, the phase would not be revisited again. Despite the rigorous approaches enforced by the Waterfall Model, hints of iterative and adaptive techniques more common to modern day agile approaches to software development were evident in Royce's paper (Larman, 2003:48).

In the mid-to-late 1970s, more iterative approaches to software development began to appear in the literature (see, for example, Mills, 1973; Brooks, 1975; Basili and Turner, 1975). These approaches came as a departure from the single pass Waterfall Model, encouraging programmers to revisit phases in the software development lifecycle, establishing an iterative process to software engineering.

In 1977, Tom Gilb proposed an evolutionary approach to software development, placing emphasis on small incremental releases, frequent delivery to users, and dynamic plans and processes (Gilb, 1977). Gilb developed his evolutionary and incremental approach over the next decade (Gilb, 1981 and 1988) during which time, further literature from proponents of incremental software development arose (Boehm, 1981; Booch, 1983). In Boehm's well-known book on software engineering economics, incremental and iterative approaches to developing software were mentioned, although briefly, (Boehm, 1981:41), and in 1988, Boehm developed a milestone approach to software development called the Spiral Model (Boehm, 1988). The Spiral Model evolved over several years based on experience with various refinements of the Waterfall Model as applied to large government software projects (Boehm, 1988:64). The Spiral Model was a risk based process model generator used to guide multi-stakeholder concurrent engineering of software-intensive systems (Boehm, 2000:3). Boehm suggested that the Spiral Model could be used to accommodate previous models and provide guidance as to which models fit a given software situation (Boehm, 1988:65).

The Capability Maturity Model (CMM) (Paulk, 1993) emerged in the early nineties focusing on recommendations for good management and engineering practices, unlike the Waterfall or Spiral models that prescribed methods for software engineering (Paulk, 2001). Organisations were ranked against five levels of the CMM with the lowest level at level one describing a chaotic organisation, to level five being an optimized organisation. The CMM prescribed processes and goals in each of the five levels, with organisations achieving higher levels of capability maturity by adhering to these processes and goals. As Paulk described, the objective of the CMM was process consistency, predictability, and reliability; furthermore, the rigor of the CMM would provide strict goals and guidelines for software development organisations in order to improve their processes.

The Rational Unified Process (RUP) (Jacobson et al., 1999) emerged fairly recently as an industry standard software development approach. RUP incorporates four phases called Inception, Elaboration, Construction, and Transition, and throughout these phases nine workflows (Krutchen, 1999) run simultaneously. These workflows define processes including business modelling,

requirements specification, systems analysis, design, implementation, testing, configuration and change management, project management, and environment. For each workflow, roles are delegated to address all the processes and documentation required. Furthermore, RUP includes six practices that form the foundation for the workflow phases and roles; these include iterative software development, requirements management, component-based architectures, the use of visual modelling software, verification of software quality, and software change control. Given the meticulous nature of the RUP workflow-driven phases and allocated roles, RUP is considered a relatively heavyweight approach; however, according to Krutchen (1999) organisations can utilize some parts of RUP without having to follow all aspects of the method out-of-the-box, making it an attractive choice.

Software development organisations continue to embrace principles from traditional approaches such as The Waterfall and Spiral Models; however, there is increasing interest within the practitioner community towards more recent approaches to developing software, in particular, agile software development methods.

Agile methods have emerged as an effort to meet the increasing demands during software development projects, focusing on aspects such as adapting to changing customer requirements during software development projects, frequent releases of software, and strong communication and collaboration using frequent customer feedback. More and more organisations appear to be adopting agile methods from the increasing number of experience reports emerging (see, for example, Appendix A); however, few investigations have been conducted to explicate the benefits and constraints of these approaches.

A Brief History of Agile Software Development Approaches

In 2001, a group of seventeen proponents of non-traditional, lighter software development approaches assembled to discuss common beliefs and alternatives to traditional methods. At this meeting, representative proponents of XP, Scrum, DSDM, FDD, Crystal, Adaptive Software Development, and other non traditional methods, shared their opinions and attitudes towards software development methods. The outcome of the meeting resulted in the formation of a consortium called the Agile Alliance (Agile Alliance, 2001) along with a policy based upon the common values and beliefs within the Alliance; this policy was named The Agile Software Development Manifesto.

The Agile Software Development Manifesto

The Agile Software Development Manifesto describes the shared values and principles between members of the Agile Alliance. They say:

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

1. **Individuals and interactions** over processes and tools.
2. **Working software** over comprehensive documentation.
3. **Customer collaboration** over contract negotiation.
4. **Responding to change** over following a plan' (Agile Alliance, 2001).

Furthermore, the Agile Alliance proposes twelve principles to support these values (see Appendix B). The values in the manifesto are, as Cockburn says, '...an outcome of our direct experience and reflection on that experience' (Cockburn, 2002: 216). The values in the manifesto emphasize people, working software, collaboration, and the ability to respond to change. The reduced emphasis on traditional values such as strict guidelines, processes, documentation, and more focus on people, is what proponents of agile methods claim to make agile software development teams more able to adapt to change. Highsmith and Cockburn claim:

'Agility, ultimately, is about creating and responding to change. What is new about agile methods is not the practices they use, but their recognition of people as the primary drivers of project success, coupled with an intense focus on effectiveness and manoeuvrability. This yields a combination of values and principles that define an agile world view' (Highsmith and Cockburn, 2001:122).

The values and principles of the manifesto have provided the underpinnings for many of the practices and values of agile methodologies. Agile methods embrace the values of the manifesto through practices based upon the communication, collaboration, and adaptive aspects of software development.

What is Agility?

According to the Oxford English Dictionary *agile* means 'nimbleness and quick moving' (Oxford English Dictionary, 2001:6). In the context of software development, Highsmith describes agility as:

'*Agility* means quickness, lightness, and nimbleness – the ability to act rapidly, the ability to do the minimum necessary to get the job done, and the ability to adapt to changing conditions' (Highsmith, 2002b:30).

Moreover, Cockburn says:

'Agile implies being effective and maneuverable. An agile process is both light and sufficient. The lightness is a means of staying maneuverable. The sufficiency is a matter of staying in the game' (Cockburn, 2002:178).

Krutchen describes agility in an organisational context:

'Agility, for a software development organisation, is the ability to adapt and react expeditiously and appropriately to changes in its environment and to demands imposed by this environment. An agile process is one that readily embraces and supports this degree of adaptability. So, it is not simply about the size of the process or speed of delivery; it is mainly about flexibility' (Krutchen, 2001:27) .

Thus, according to these statements, being agile in a software development context means the ability to adapt and move quickly to the changing demands in the environment.

Agile approaches to software development claim to be more people-oriented, less bureaucratic and process-oriented than heavyweight methods that place more emphasis on process and documentation. Fowler believes the attraction of agile methods is a reaction to bureaucracy in traditional methods:

'For many people the appeal of these agile methods is their reaction to the bureaucracy of the monumental methods. These new methods attempt a

useful compromise between no process and too much process, providing just enough process to gain a reasonable payoff' (Fowler, 2001).

Despite the claimed benefits of agile methods, most of the literature stating these claims has been written by proponents of agile methods; thus, these claims have presented a strong level of subjectivity towards the efficacy of agile methods. However, regardless of this subjectivity there are a growing number of experience reports emerging from organisations finding success with agile methods; noticeably, there is a particular abundance of literature afforded to XP from the practitioner community (see, for example, Appendix A) and some literature from the academic community (see, for example, Rasmusson, 2003; Williams, 2003; Sharp and Robinson, 2004).

A significant proportion of the literature addresses areas of controversy between agile and traditional approaches (see, for example, Glass, 2001; Beck and Boehm, 2002; DeMarco and Boehm, 2002), in particular, the conflicts between the values and practices of agile versus traditional software development methods. Moreover, there is an acknowledgement of the difficulties traditional software engineers have in accepting such a divergence to software development in agile methods such as XP. Beck states:

'The XP practices are so sideways to what we have heard and said and maybe even been successful within the past. One of the big difficulties is just how contrary XP sounds' (Beck, 1999:153).

Many of the criticisms of agile methods have related specifically to practices found in XP; consequently, the current literature concerning agile methods focuses strongly on XP.

Agile Methodologies

There are a number of software development methodologies that have been categorized under the Manifesto umbrella. XP, SCRUM, DSDM, FDD, and Crystal Methods are among the most popular examples of these agile methods. Increasing numbers of software development organisations have adopted agile methods in an attempt to change and improve their development practices in order to react and adapt to constant change during software development projects. I shall now briefly describe each of these methods.

Of all the agile approaches, XP has gained more attention and acceptance than any other agile methodology (DeMarco in Beck and Fowler, 2000). Beck states

that 'XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software' (Beck, 1999: xvii). XP requires adherence to values and practices that endeavour to address the needs of software development teams facing vague and changing requirements (Beck, 1999). XP differs dramatically from traditional software development tenets by dispensing with up-front analysis, simplifying design, encouraging programming in pairs, enforcing frequent release of software, adopting close customer collaboration, and enabling teams to react to constant change. The focus on flexibility and dispensing with unnecessary bureaucracy may make XP an attractive methodology for software teams, and indeed may help explain its popularity and acceptance in the software engineering community.

The Scrum methodology (Schwaber and Beedle, 2001) differs to XP in that 'XP has a definite programming flavour (for example, pair programming, coding standards, refactoring), Scrum has a project management emphasis' (Highsmith, 2002b: 242).

According to Schwaber

'Scrum addresses the complexity of software development projects by implementing the inspection, adaptation and visibility requirements of empirical process control with a set of simple practices and rules...'

(Schwaber, 2004: 5).

Scrum relies on self organising teams as opposed to the strict authoritarian measures associated with many traditional software development methods (Schwaber in Highsmith, 2002).

Dynamic Systems Development Method (DSDM) (Stapleton, 1997) is considered the most mature agile methodology and has enjoyed much success since the early 1990s (DSDM Consortium, 2003). At that time, the DSDM Consortium formalized an existing software development approach called RAD (Rapid Application Development), which aimed to deliver software quickly through prototyping and iterative design and testing. DSDM is based on nine principles which align with the principles of the Agile Manifesto (Highsmith, 2002b). These principles include self-steering teams, frequent delivery of products through iterative and incremental development, and strong cooperation and collaboration between all stakeholders.

Feature Driven Development (FDD) (Coad et al. 2000; Palmer and Felsing, 2002) is a process-oriented software development method for developing business critical systems (Abrahamsson et al., 2003). FDD is a software development approach that primarily focuses on design and implementation of high quality deliverables on a frequent basis. Of all the agile methods, FDD is the most process-oriented method requiring practitioners to adhere to strict practices; however, like many agile approaches FDD follows an iterative approach to software development.

Crystal Methods (Cockburn, 2004) are a collection or *family* of methods that can be tailored depending on the characteristics of the software development project. Individual methods in Crystal are selected depending on the size and criticality of the project. Crystal is a very flexible approach to software development allowing practices and methods from other software development methodologies to be incorporated within its own family of methods and processes.

The Claimed Benefits and Constraints of Agile Approaches

Introduction

In the first section I discuss the most popular agile method - Extreme Programming - summarising the current areas of debate; the next sections address the human factors of agile methods - relationships and trust, culture, and social aspects; finally, I discuss the ongoing debate between proponents of agile and traditional approaches to software development.

Extreme Programming (XP) – The Most Popular Agile Methodology

XP was created by Kent Beck during his time working on a software development project in the mid-1990s. Experiences drawn from this project encouraged Beck to develop and refine a methodology that was later published in his book 'Extreme Programming Explained' (Beck, 1999).

Since that time, XP has seen an increase in popularity and acceptance within the software development community; furthermore, it has gained a reputation as the most recognised agile methodology of all that currently exist. DeMarco (2002) claims that:

'XP is the most important movement in our field today. I predict that it will be as essential to the present generation as the S.E.I. and its Capability

Maturity Model were to the last' (DeMarco in Beck and Fowler, 2000: *in Forward*).

Fowler claims that of all the methodologies that fit the agile methods banner, XP has gained the most attention. He believes this is partly because of the 'remarkable ability of the leaders of XP' (Fowler, 2001) to gain attention. Fowler claims that XP has gained so much attention that

'...its popularity has become a problem, as it has rather crowded out the other methodologies and their valuable ideas' (Fowler, 2001).

Cockburn believes that the popularity of XP lies with how its practices reflect being agile and that XP is 'effective, well documented, and controversial' (Cockburn, 2002:165).

Despite these claims, XP has been a centre of debate between traditional and agile camps in the software engineering community (see, for example, Glass, 2001). For advocates of more traditional methodologies, practices such as: programming in pairs; simple design as opposed big design up front (BDUF); adapting to changing requirements as opposed to pre-determined requirements, and XP's on-site customer, are amongst the most controversial practices of XP. Moreover, issues concerning the interdependency of XP, as stated by Beck, '...any one [XP] practice doesn't stand well on its own (with the possible exception of testing). They require the other practices to keep them in balance...' (Beck, 1999:69) has provoked others to criticize XP and suggest ways to improve it (see, for example, Stephens, M. and Rosenberg, 2003; Boehm and Turner, 2003d).

The next section will summarise the most controversial practices and constraints of XP that have arisen from in the literature.

Controversial Practices

Of the twelve practices of XP, (see Appendix C), four practices in particular frequently emerge as controversial. These practices are Metaphor, Simple Design, Pair Programming, and On-site Customer.

Metaphor

Architecture is not a term used very often with the XP methodology, in its place, XP uses an all-encompassing *story* (Beck, 1999) called the metaphor that describes and controls software development projects. The metaphor describes a comprehensible story shared amongst all stakeholders involved in a project. XP

discards architectural documentation such as meticulous system design specifications seen in more common traditional approaches. Instead, XP places more emphasis on strong communication and collaboration between the development team and customer using the metaphor to guide progression. According to Beck, the metaphor replaces much of what people call architecture, with the aim 'to guide all development with a simple shared story of how the whole systems works' (Beck, 1999:54). Furthermore, he states:

'Part of the architecture is captured by the system metaphor. If you have a good metaphor in place, everyone on the team can tell about how the system as a whole works' (Beck, 1999:113).

Williams describes the system metaphor as a story that everyone such as customers, programmers, and managers understand, and that it 'shapes the system'; furthermore it is seen as the system architecture in common development vocabulary (Williams, 2003:18). However, the use of a metaphor has met criticism, for example, Murru et al. claim they were unable to make the metaphor work in place of a clearly defined architecture arguing that 'XP has no clear recipe for developing an architecture' (Murru et al., 2003:39). Moreover, Rasmusson experienced difficulties when trying to explicate the XP system metaphor while introducing XP into new projects (Rasmusson, 2003). However, Beck argues that in order for the metaphor to work, a story should be developed and agreed between all stakeholders in order for the metaphor to be derived and understood (Beck, 1999).

Simple Design versus Big Design Up Front

Beck claims XP's simple design practice provides a mechanism to adapt to rapid change when uncertainty is a major factor within a project. He claims that if people believe the future is uncertain, and that you can cheaply change your mind, then putting in functionality speculatively, that is, big design up front, is crazy. Beck states software teams should only put in what is needed and to 'do the simplest thing that could possibly work' (Beck, 1999: 103).

Williams claims that XP developers move hastily and are agile through every aspect of software development including the design phase. She claims XP programmers use few-minute cycles for practices such as simple design:

'the purpose of this "every few minutes" cycle is to get feedback early [from stakeholders]' (Williams, 2003:16).

Wood and Kleb (2003) found XP's simple design practice helped break down large tasks into smaller more manageable parts; however, they also found that simple design became a problem due to the complex algorithms needed for optimal performance within the context of their project.

In his recent book, Beck states how difficult simplicity is:

'Simplicity is the most intensely intellectual of the XP values. To make a system simple enough to gracefully solve only today's problem is hard work' (Beck, 2004:18).

However, he argues that:

'XP is making a bet. It is betting that it is better to do a simple thing today and pay a little more tomorrow to change it if it needs it, than to do a more complicated thing today that may never be used anyway' (Beck, 1999:31).

Pair Programming

Pair programming is a practice of XP that has enjoyed much attention in literature (see, for example, Williams and Kessler, 2000 and 2002; Jensen, 2003; Parish et al., 2004). According to the claims on a popular XP web resource:

'All production software in XP is built by two programmers, sitting side by side, at the same machine. This practice ensures that all production code is reviewed by at least one other programmer, and results in better design, better testing, and better code' (Extreme Programming, <http://www.xprogramming.com/xpmag/whatisxp.htm>, 2002).

Pair programming encapsulates the communicative and collaborative values of agile approaches and encourages developers to share knowledge and to work together towards a common development goal.

Rasmusson (2003) describes how pair programming was the most powerful XP practice adopted by the team, and how the practice contributed largely to the overall success of the project. He describes how pairing provided a mechanism for effective communication and consistency throughout the development team; however, in contrast, he described the drawbacks encountered when pairing junior and senior developers because the 'the junior felt like a passenger travelling on a high-speed train' (Rasmusson, 2003:22). Furthermore, he states

that pair programming was a critical factor during the introduction of test driven development to the team and that without pair programming, individual software developers tended to revert back to traditional methods of working alone (Rasmusson, 2003).

According to Murru, pair programming disseminated skills across the development team and 'disciplined programmers by averaging individual idiosyncrasies' (Murru, 2003:40). Furthermore, Wood and Kleb claimed they achieved productivity gains through pairing because it created a 'pressure effect to intense sessions that discouraged cutting corners' (Wood and Kleb, 2003:33).

According to these viewpoints, it appears the coupling of experienced and less experienced developers through pair programming promotes a skill-averaging affect, balancing the mixed competencies found in software development teams. Moreover, pair programming appears to force programmers to communicate closely while coding, an aspect seldom experienced in the isolation of traditional programming practices.

Beck highlights the benefits of pair programming and states:

'Pair programming to communicate, get feedback, simplify the system, catch errors, and bolster your courage makes a lot of sense' (Beck, 2004:35).

On-site Customer

The on-site customer practice of XP aims to encourage rapid feedback and decision making, with face-to-face communication between the customer and development team. Beck believes that the business domain knowledge and skills brought by the on-site customer are critical to the success of the project:

'A real customer must sit with the team, available to answer questions, resolve deputes, and set small-scale priorities. By "real customer" I [Beck] mean someone who will really use the system when it is in production' (Beck, 1999:60).

The on-site customer practice of XP has been one of the central issues of debate between agile and traditional software engineers. According to results of a survey conducted by Rumpe and Schroder in 2001, of the 45 organisations using XP that took part, the on-site customer was one of the most difficult XP practices to satisfy (Rumpe and Schroder in Cohen et al., 2003). Furthermore, Bailey et al.

believe that the on-site customer could cause an adverse effect on the XP development team:

'An on-site customer is rapidly aware of exactly how fast development proceeds, and the planning game makes them aware of how it is likely to proceed in future. If this rate is not sufficient to meet deadlines, customers are prone to fall back on old habits and demand increased development speed by various flawed practices' (Bailey et al., 2002).

Despite the obvious benefits of an on-site customer providing immediate domain knowledge and feedback, the literature suggests organisations have found it very difficult to utilize this practice. The main reason seems to be that customers have jobs and therefore their own responsibilities, so having an on-site customer is an unrealistic practice. Moreover, some customers could find it difficult to fit into an XP software development team. In addition, from the perspective of the XP team, managers might feel uncomfortable allowing customers see the 'inner workings' (Murru et al. 2003:41) of the organisation, and developers might feel uncomfortable with the constant presence and pressure from customers.

According to Beck, the main advantages of the on-site customer are the communication, collaboration, and frequent feedback during projects. In his recent book, Beck states:

'The point of customer involvement is to reduce wasted effort by putting the people with the needs in direct contact with the people who can fill those needs' (Beck, 2004:61).

However, despite this argument, it appears there is little evidence in the literature to suggest the on-site customer practice has been adopted or utilised successfully by organisations.

Constraints

Some agile methods have been criticised on the grounds that the practices might constrain software development teams, for example, how can XP pair programming be adopted by distributed teams? How can teams employ stand-up meetings, such as in the Scrum agile methodology, if teams are distributed? And how can large teams adopt XP if it is said to be better suited to smaller teams? (Beck, 1999).

Scalability

According to Beck, the size of an XP team is crucial and is better suited to a small number of developers:

'Size clearly matters. You probably couldn't run an XP project with a hundred programmers. Nor fifty. Nor twenty, probably. Ten is definitely doable' (Beck, 1999:157).

In order to address scalability issues, Boehm and Turner believe agile methods may be complemented by more traditional approaches. In response to a 50-person XP case study, they claim that:

'...a larger agile project needs to adopt traditional plans and specifications in order to deal with the increasingly complex, multidimensional interactions among the project's elements' (Boehm and Turner, 2003d:28).

Furthermore, in the debate between Boehm and Beck, Boehm argues that XP needed to draw upon more disciplined approaches in order to scale (Beck and Boehm, 2003).

Communication is believed by some to be a critical factor affecting scalability. For example, having large teams might adversely affect face-to-face meetings due to the greater numbers of individuals involved. According to Eckstein (2004), lack of communication may affect how well software development projects scale, however she believes agile methods might alleviate this problem because of the value agile methods place on communication.

In Fredrick's view, in order for XP to scale a team should be grown, such that developers are added to the project to handle the increasing number of stories (Fredrick, 2003).

There have been few reports of how effective agile methods are in large teams and on large projects; this point is raised by Abrahamsson et al. who believe this might negatively impact the acceptance of agile methods as a whole:

'The majority of the software in large organisations is not produced by co-located teams of less than ten engineers. If, for example, scalability problems are not solved, agile thinking will not gain the serious attention it deserves' (Abrahamsson et al., 2002:99).

However, in Beck's recent book, he claims XP can scale-up successfully if the necessary awareness and appropriate adaptations are considered:

'With awareness and appropriate adaptations, XP does scale. Some problems can be simplified to be easily handled by a small XP team. For others, XP must be augmented. The basic values and principles apply at all scales' (Beck, 2004:117).

Locality

Locality constraints with some agile practices have caused some debate with respect to the suitability of agile methods in certain team environments, in particular teams that are geographically dispersed. Williams notes the importance that agile teams place on co-location:

'XP teams value face-to-face communication and spend essentially the whole day, every day communicating' (Williams, 2003:18).

Rasmusson amplifies this claim from experiences of an XP project:

'This [XP] project's optimal ecosystem was when the team was co-located, working within earshot of one another. Big open spaces with large tables where developers could sit side by side and pair-program were most conducive to a productive work environment' (Rasmusson, 2003:24).

According to Cohn and Ford, co-location might not be an issue as long as distributed development teams:

'...resolve political and cultural issues before developers can succeed with a project shared across cities. ...they [the XP teams] must be combined, bringing them together in the first one or two weeks of the project [to] increase the likelihood of success' (Cohn and Ford, 2003:75).

From experiences on a recent XP project, Braithwaite and Joyce found XP was successful in a distributed environment as long as the team made '...a commitment to the value judgments that characterise the core of all agile methods, the Agile Manifesto' (Braithwaite and Joyce, 2005:3). Furthermore, they state that:

'We have found that you can stay true to the principles and not compromise the practices of XP in a distributed environment. Thus, business can realize

both the benefits of distributed and of truly agile development' (Braithwaite and Joyce, 2005:1).

In order to address locality constraints of some agile practices, Stotts et al. (2003) describe the concept of a virtual team, whereby a group of people work towards a common objective across a number of dimensions including distance:

'A virtual team can be defined as a group of people who work together towards a common goal but operate across time, distance, culture and organisational boundaries' (Stotts et al., 2003:2).

Stotts et al. (2003) claim the constraints of co-location in practices such as XP's pair programming would be 'a limitation that ostensibly hinders use of XP for distributed development of software' (Stotts et al. 2003:1) In order to investigate this issue, Stotts et al. carried out two investigations using student participants to test the feasibility of pair programming between distributed pairs, a concept Stotts called 'Distributed Pair Programming'. The results of these initial studies claimed that distributed pair programming in virtual teams was a feasible way of developing software, and that distributed pairs could maintain many of the benefits (pair pressure, pair learning, two brains) seen in co-located pairs (Stotts et al., 2003); however, these claims were based upon single study using students in a controlled environment, and not in an industrial 'real-life' context.

In support of distributed teams running agile projects, Sepulveda states that:

'We have been successful delivering quality software in timely and reliable manner for about a year. We are actually more successful in the remote team arrangement than the co-located one' (Sepulveda, 2003:6).

And in an experiment conducted by Baheti et al., they found that:

'...co-located teams did not achieve statistically significantly better results than the distributed teams' (Baheti et al., 2002:9).

Little is understood concerning the benefits of virtual teams and practices such as distributed pair programming due to the paucity of evidence supporting these concepts. However, Highsmith claims that a number of XP practitioners are already exploring ways to expand the use of agile methods into other problem domains, particularly those involving larger projects and distributed teams (Highsmith, 2002c). Williams and Cockburn state that researchers are currently

trying to explicate a suitable agile value set and practices as a list of constraints relating to team size, skill, and co-location (Williams and Cockburn, 2003).

Skill and Talent

The requirement for strong skills and talent within agile software development teams is a topical area of debate. Some writers (see, for example, Boehm, 2002; Boehm and Turner, 2002a; Paulk, 2002) claim the success of agile methods is dependant on highly competent individuals and intrinsic elements such as tacit knowledge. It might be argued that some agile practices enhance the competency of an individual, for example, advocates of agile methods believe some practices, such as XP's pair programming, raise competency levels within software teams by encouraging skill and knowledge distribution amongst individuals.

Cockburn and Highsmith claim agile methods place more importance on amicability, talent, skill, and communication in software development projects than traditional approaches (Cockburn and Highsmith, 2001). However, Boehm claims agile methods place more reliance on skill and talent and quotes Larry Constantine's assessment of XP:

'...there are only so many Kent Becks in the world to lead the team. All of the agile methods put a premium on having premium people' (Constantine 2001 in Boehm, 2002:65).

From their experiences of introducing an agile method into an organisation, Cohn and Ford found that agile development teams moved very quickly and that too many slow, less skilled workers slowed the entire team (Cohn and Ford, 2003). Moreover, when introducing XP into a project, Rasmusson found that test driven development took time because of the amount of skill required (Rasmusson, 2003). Furthermore, Murru et al. (2003) reported that XP's refactoring practice required senior programmers because inexperienced programmers often lacked the experience to judge quality during code updates. However, in contrast, Murru et al. applauded the flexibility XP brought to an inexperienced team and claimed that XP compensated for a development team lacking in programming experience; however, they argue that an inexperienced team would probably be poor at applying XP (Murru et al., 2003).

Highsmith argues that individual skills are a requirement in all software teams, and that having skilled individuals is not only a prerequisite for agile methods, but also for more rigorous methods:

“Others have cautioned about the necessity of having skilled individuals in order for agile approaches to work. To summarize, agile approaches require skilled leadership and team members. If this statement is true, then the flip side should be true also; namely, that rigorous approaches can be utilized by unskilled leaders and team members. The arguments for and against certain methodologies often founder on this confusion between process and skill’ (Highsmith, 2002:97).

Furthermore, he claims that an agile process helps teams to deliver the best software they are capable of producing:

‘No amount of additional process will make the members of the team capable of delivering a product beyond their talent and skill. While process and structure can support and leverage skill, increasing structure does not make up for skill deficiency’ (Highsmith, 2002:97).

During a discussion with a practitioner using FDD, Highsmith found that:

‘The key, he [the practitioner] said, is having good people – good domain experts, good developers, good chief programmers. No process makes up for a lack of talent and skill’ (Highsmith, 2002b:6).

Cockburn and Highsmith claim that:

‘If the people on the project are good enough, they can use almost any process and accomplish their assignment. If they are not good enough, no process will repair their inadequacy – “people trump process” is one way to say this’ (Cockburn and Highsmith, 2001:131).

Relationships and Trust

As yet, little has been written in the literature concerning the relationships between members of agile teams (Sharp and Robinson, 2004). The customer - developer relationship has received some attention (see, for example, Deursden, 2001; Highsmith, 2002), however there is a dearth of literature discussing the relationships between members of agile teams. Beck states:

‘Value in software is created not just by what people know and do but also by their relationships and what they accomplish together. Ignoring the value

of relationships and trust just to simplify the scheduling problem is false economy' (Beck, 2004:63).

Beck believes a critical aspect to software development is relationships and trust, and that trust allows teams to be more productive:

'When you act trustworthy and have nothing to hide, you are more productive. When you are ready with accurate estimates and low defect rates, including customers in the development process fosters trust and encourages continued improvement' (Beck, 2004:62).

Customer-Developer Relationship

Although there is little available literature addressing relationships between people on agile related projects, the customer-developer relationship has undoubtedly drawn some attention. For example, according to some writers, the on-site customer's relationship and collaboration with XP developers is crucial for XP to succeed. Deursen claims:

'... the success of an XP project depends on the way in which the on-site customer is able to fulfil these [collaborative] roles [with the team]' (Deursen, 2001:1).

According to some advocates of agile methods, in order for the customer-developer relationship to work effectively, a level of trust must exist between these team participants (Highsmith, 2002b). From the XP developer perspective, the customer must be trusted to have the qualities of an authorised decision maker and be a committed participant to drive the project, providing constant feedback through testing, and contributing to the development of stories. Conversely, from the customer viewpoint, they must trust the developers' abilities to progress the project to a high standard. Fowler believes the issue of trust is a key factor of XP:

'A key piece of XP is that you have to trust your developers, and the coach's role is to teach them how to do it well, but in the end you have to trust them. And if you haven't got that trust, XP can be threatening' (Fowler in Highsmith, 2002:144).

Moreover, Highsmith says:

'Value people. Trust them. Support them. These values are the essence of an agile ecosystem. Mistrust communications. Put practices in place to bridge the difficulties of communication and collaboration between trusted, valuable people' (Highsmith, 2003:103).

However, Murru et al. believe that XP practices do not necessarily improve trust:

'XP might suffer from a catch-22: It requires a high level of trust in the customer-developer relationship, especially in the initial project phase, but it [XP] eliminates activities (requirements sign-off, architecture drafting, and so on) that help increase the trust level during start-up' (Murru et al., 2003:41).

Highsmith believes that:

'If you fundamentally trust people, you provide a crisp definition of the desired outcome, a light framework within which to work, a feedback mechanism to ensure communications haven't snarled, and you then let people perform. Maybe this is one reason XP has attracted such a large following – the practices are based on trust and quality work, two characteristics that appeal to many developers' (Highsmith, 2002:306).

However, he suggests that trust may be one of the few factors that could form a barrier to the overall acceptance of XP:

'Maybe embracing trust, communications, and collaboration will ultimately prove too great a barrier to a wide adoption of Agile approaches within corporations' (Highsmith, 2002:380).

Developer-Developer Relationship

Strong levels of trust do not only apply to the relationships between developers and customers, but also to relationships among the developers themselves. In order for projects to succeed developers should trust and depend on each other to reliably share project goals in order to promote confidence in collectively meeting those goals. A case in point is XP's pair programming; developers work together in close collaboration on programming tasks to complete an objective. If there is mistrust between these developers, meeting programming tasks might be much harder to achieve. Thus, having the confidence and trust in each other enables the developers to work collectively towards a common goal.

The claimed benefits of pair programming and developers working in close collaboration have been studied (see, for example, Williams and Kessler, 2000 and 2002), however there is very little, if any, literature addressing human aspects in agile teams, such as trust, relationships, and personality. For example, what is the impact on pair programming of different developer personalities? Are certain personality traits better suited for programming pairs? These are among many human related questions that are yet to be addressed.

Manager–Developer Relationship

Agile proponents believe trust should exist between all team members; in particular, that managers should trust their development teams when taking risks to introduce new agile practices into the development process. Furthermore, developers should trust managers to support and show commitment when agile methods are adopted.

Williams and Cockburn emphasise the importance that agile approaches place on decision making across the whole software development team including managers and developers; however, they question how managerial staff might handle this collaborative decision making, and that

‘...just how to handle this power-shift remains an open question’ (Williams and Cockburn, 2003:41).

Developing a strong trust relationship is a gradual process. The collaborative aspects valued by agile methods, in particular XP, assume that trust exists in agile projects. This may not be the case if customers and developers are unacquainted; on the contrary, lack of familiarity might even have an adverse affect and exacerbate the progress of agile projects. The assumption that a strong level of trust exists immediately throughout agile teams, in particular between the developer and customer, makes for a controversial assumption by the proponents of agile methods such as XP.

Culture and Agile Approaches

One intrinsic characteristic of software development is risk (Boehm and Turner, 2003a). The risk of trying new approaches and new technologies may erode the ability of software development teams to experiment and ultimately progress and improve their practice. Lycett et al. (2003) claim that mature organisations may not accept sudden migration to agile practices, due to the risks that may be involved. Williams and Cockburn echo Lycett’s claim, considering the consternation organisations might feel adopting agile practices; they claim that

management might 'consider agile development to be a licence for developers to hack' (Williams and Cockburn, 2003:41).

The culture of software engineering organisations has an influence on whether new technologies may be adopted, for example, how much the organisation is willing to risk adopting an agile method; yet much of the software engineering literature continues to address technical aspects of software development as opposed to cultural aspects (see, for example, Glass et al., 2002; John et al., 2005; Segal et al., 2005). A possible reason for this trend may be because software engineering is seen as a technical field, therefore investigations tend to focus more on the technological processes and artefacts of software engineering, rather than the people involved. It appears some attention has been given to the importance of investigating human factors with respect to software engineering; Sharp et al. state that:

'...while our [software engineering] community is constantly redefining and expanding the notion of human factors in software engineering, a broader understanding of how the human and social environment affects software engineering escapes us. In working to gain this better understanding, the three of us have found that software engineering's very nature is mutably shaped by the human and social world in which it exists, and that a distinct culture of software engineering transcends national, regional, and organizational cultures' (Sharp et al., 2000:40).

According to Lindvall et al., agile software development is characterized as a cultural activity rather than a process-driven activity:

'To be agile is a cultural thing. If the culture is not right, the organisation cannot be agile' (Lindvall et al., 2002:7).

Moreover, Beck highlights the importance culture has on XP:

'...the biggest barrier to the success of an XP project is culture' (Beck 1999:156).

According to Cockburn, not understanding the cultural aspects of methodology, process, and practice implementation leads to failure (Cockburn in Highsmith, 2002). And Highsmith claims:

'More methodology implementations flounder from failing to adequately account for an organisation's cultural factors than for any other reason. A particular culture is not necessarily change tolerant or change resistant – but it may resist certain types of changes and embrace others. Culture provides both the strength to persist in the face of adversity and the weakness to persist in the face of folly. Any attempt to implement a methodology requires that an organisation analyze its own fundamental culture – its core values' (Highsmith, 2002:326).

From their review of adoption of technology (such as agile methods), Sultan and Chan found that:

'...the literature suggests that a favourable company culture and climate is one of the most important resources and that such a culture promotes adoption of new technologies' (Sultan and Chan, 2000:110).

With respect to XP, Sharp and Robinson believe that:

'The [XP] practices are significant when developing and building a community with the appropriate culture (Level 0). However, once the community is established and the values are embedded in the culture, then the practices can be modified. Eventually, an alternative set of practices can be found, if required, because the community's culture is so strong and the values so deeply embedded' (Sharp and Robinson, 2003:9).

In this quote, Level 0 means to follow all 12 XP practices all the time according to the so-called 'XP Maturity Model' (reported by Alan Francis at XP2003 quoting Kent Beck).

West believes XP is not a methodology, but rather a culture, he claims that:

'XP is a culture, not a method, not a tool, not a set of techniques' (West, 2001:5).

The underlying message from these viewpoints is the significance of human factors in software engineering: the way people work; their behaviours; their attitudes; their beliefs, and the organisational culture as a whole. Highsmith uses the term 'Agile Software Development Ecosystem' (ASDE) to define the culture of an agile team, he states:

'Agility isn't defined by a set of practices; it is defined by a set of cultural beliefs. Understanding your organization's culture is, therefore, the first step in transitioning to an agile ecosystem' (Highsmith, 2002b:205).

Software Development as a Social Activity

According to Dittrich et al. (2005), 'Computing and especially software engineering is a social activity' (Dittrich et al., 2005:1). The values of the Agile Manifesto encourage these social activities through communication, collaboration, feedback, interaction, reflection and adaptation. Despite this, it appears much of the research literature continues to address technical aspects of software engineering. According to John et al.:

'Human and social factors have a very strong impact on the success of software development endeavours and the resulting system. Surprisingly, much of software engineering research in the last decade is technical, quantitative and deemphasizes the people aspect' (John et al, 2005:1).

Rasmusson highlights the social aspect of XP:

'XP's social side really shone through when people got together, shared experiences, and collectively looked for solutions' (Rasmusson, 2003:22).

And he concludes that 'XP is a very social way of developing software' (Rasmusson, 2003:27).

Moreover, Aur et al. assert the importance of social interaction on XP projects and how '...the [XP] team must act as an Effective Social Network' (Auer et al., 2003:2).

Reporting on an ethnographic study of a small company using XP to develop web-based intelligent advertisements, Sharp and Robinson argue:

'...that learning XP is not just learning the technology of the practices but also the learning of that social activity' (Sharp and Robinson, 2004:371).

In Beck's most recent book on XP, he claims, before anything else, that 'Extreme Programming (XP) is about social change' (Beck, 2004:1). He claims 'Good, safe, social interaction is as necessary to successful XP development as good technical skills' (Beck, 2004:4). He continues by stating how the sound of conversation,

and therefore social interaction, is a sign of good health in software development teams:

'People evaluating XP teams should understand what an effective team looks like. This may differ from other teams they have seen. For example, talking and working go together on an XP team. The hum of conversation is a sign of health. Silence is the sound of risk pulling up' (Beck, 2004:79).

Agile methods place more focus on social aspects than traditional methods of software development, for example, through XP's pair programming. However, it appears little research effort has been placed on the social aspects of agile methods and software engineering as a whole. Given the viewpoints of agile proponents, if indeed software engineering is a social activity, then surely empirical investigations of the social aspects of software development would provide useful insights?

Although there have been some efforts to investigate the human factors of agile methods (see, for example, Sharp and Robinson, 2004), there is a lack of investigations addressing human aspects across the field of software engineering research in general (Segal et al., 2005).

It is believed by some (see, for example, Sim et al., 2001; Pfleeger, 2005) that drawing upon research techniques from other fields, such as social science, may advance software engineering research and may help understand human related aspects:

'Just as it makes sense to re-use code within software engineering, it also makes sense to harness theories and methods from other fields to advance software engineering research' (Sim et al., 2000:1).

The Ongoing Debate

The agile versus traditional methods debate continues within the software engineering community (see, for example, Charette, 2001a, 2001b, 2001c, 2001d; Glass, 2001; Beck and Boehm, 2003). Concern regarding the suitability of agile methods in certain environments has given rise to proposals for mixing agile approaches with more traditional plan-driven approaches. Boehm believes combining agile and plan-driven approaches based upon a risk analysis may provide a beneficial approach:

'I believe that both agile and plan-driven approaches have a responsible center and overinterpreting radical fringes. Although each approach has a

home ground of project characteristics within which it performs very well, and much better than the other, outside each approach's home ground, a combined approach is feasible and preferable' (Boehm, 2002:64).

Critics of agile methods have questioned the level of discipline in agile methods (see, for example, Boehm, 2002), however Beck argues that, 'given a broader view of discipline, XP is far more disciplined than most processes' (Beck and Boehm, 2003:44). Moreover, Beck challenges Boehm's view of software development as a plan-oriented interpretation compared to what Beck calls his own reality-oriented interpretation; Beck believes that plan-oriented and reality-oriented should not be treated separately, and that agile methods require significant amounts of discipline to succeed (Beck and Boehm, 2003). Boehm concludes the debate, claiming what is needed are

'...efforts to synthesize the best from agile and plan-driven methods to address our future challenges of simultaneously achieving high software dependability, agility, and scalability' (Beck and Boehm, 2003:46).

Boehm and Turner believe there are five critical factors which enable organisations to plan how to balance agility and discipline: 'size, criticality, personnel, dynamism, and culture' (Boehm and Turner, 2003c:1). Moreover, they predict future projects may soon place a high premium on methods that draw upon a hybrid approach combining agility and discipline:

'Large projects can no longer count on low rates of changes, and their extensive process and product plans will become expensive sources of rework and delay. As the use of agile methods progresses from individual early-adopter projects to enterprise-coupled mainstream applications, the Brooksian software development werewolves of complexity and conformity will be waiting for them. Thus there will be a higher premium on having methods that combine agility and discipline in situation-tailorable ways' (Boehm and Turner, 2003b:3).

They conclude that:

'Methods are important, but potential silver bullets are more likely to be found in areas dealing with people, values, communications, and expectations management' (Boehm and Turner, 2003d:148).

According to Williams and Cockburn, software development teams are already attempting to combine agile practices with their existing practices by adopting a few practices at a time:

'Development groups are trying pair programming without any other agile practices, test-first development without pair programming, and short delivery cycles without pair programming or test-first development, and trying to determine how to get any of these practices to work with internationally distributed teams. Researchers worldwide are gathering these agile subset experiences and designing and staging additional experiments to empirically assess the efficacy of isolated practices or agile subsets' (Williams and Cockburn, 2003:41).

However, Highsmith believes the important questions of the debate are not so much to do with the practices drawn from both agile and traditional methods, but rather, the cultural aspects:

'The core issue is not whether CMM-like or Agile-like practices are *best*, the core issue is building an organizational culture with a balance of practices that support innovation, discipline, and adaptability... Articulating an organisation's values and principles is key to understanding its culture and to successful methodology implementation' (Highsmith, 2002:333).

Rogers' Model of Diffusion of Innovations and the Adoption of Agile Methods

In this section, I summarise the established model of the diffusion of innovations as described by Rogers (2003), and comment on how this model is appropriate to the investigation of adoption of agile methods in organisations.

According to Rogers, an innovation is

'...an idea, practice, or object that is perceived as new by an individual or other unit of adoption' (Rogers, 2003:12).

And, that diffusion is

'...the process by which an innovation is communicated through certain channels over time among the members of a social system' (Rogers, 2003:35).

Agile methods such as XP are considered to be innovations since they are relatively new and evolving; however, not all practices of agile methods are particularly innovative. For example, small incremental releases of software and frequent delivery to users are practices that have been reported almost 30 years ago (Gilb, 1977). Yet, XP as a whole for example, is considered an innovation since its approach is new in contrast with traditional, long-standing methods of software development.

Rogers' established model of diffusion of innovations is based upon the synthesis of results from many different studies of diffusion of many innovations, drawing upon diffusion studies as early as 1943, as Raghavan and Chand state:

'Rogers's framework for diffusion of innovations was originally based on extensive study of agricultural innovations [by Ryan and Gross in 1943]. It was later enriched and validated through many empirical studies in other settings. It is a comprehensive framework that can be used descriptively or prescriptively' (Raghavan and Chand, 1989:82).

Furthermore, Raghavan and Chand state:

'We strongly believe that studying and adapting Rogers's framework to the software-engineering context could significantly enhance our overall understanding of the diffusion process and help us develop effective

strategies to successfully diffuse software engineering innovations' (Raghavan and Chand, 1989:82).

Rogers' established model is now summarised:

Rogers states that innovations require significant time to be widely accepted and adopted. He believes the important factors influencing the rate of adoption of new innovations are the values, beliefs, experiences, and interpersonal networks of individuals who are in the social system exposed to the innovation, and that the diffusion of innovations is a 'kind of social change' (Rogers, 2003:6).

Rogers introduces the concept of the *innovation-decision* process in his model, whereby an individual (or group) actively seek to determine the advantages and disadvantages of an innovation in order to reduce *uncertainty* before adoption. During this time, individuals evaluate the *relative advantage* and *compatibility* of an innovation to determine the degree at which the innovation is perceived to be better than the idea it supersedes, and how the innovation is consistent with the existing values and needs of these potential adopters. Furthermore, he states that individual perceptions of *complexity*, *trialability*, and *observability* of a new innovation are viewed as the 'most important characteristics in explaining the rate of adoption' (Rogers, 2003:16). He states, individuals first acquire knowledge about an innovation, formulate an attitude towards that innovation, and decide to adopt or reject that innovation, then implement that innovation through to confirmation of the decision to adopt. These five-stages of (1) knowledge, (2) persuasion, (3) decision, (4) implementation, and (5) confirmation, define the innovation-decision process. Furthermore, he describes five categories of a social system based upon their level of innovativeness, these categories are (1) innovators, (2) early adopters, (3) early majority, (4) late majority, and (5) laggards. These categories signify the groups of individuals respective of their innovativeness and likely rate of adoption. Innovators are the most likely to develop or adopt new innovations, whereas at the farther end of the scale the laggards are the last or most resistant individuals to adopt an innovation. Rogers states the member classifications and structure within a social system can 'facilitate or impede the diffuse of innovations' (Rogers, 2003:25).

Despite the emphasis placed on social systems Rogers says that, compared to other aspects of diffusion research, relatively few studies have addressed how diffusion and adoption is affected by social or communication structure in a system. Drawing upon results of many diffusion investigations, he concludes that

the diffusion process is sustained mostly by peer networks and not as a result of scientific studies:

'Diffusion investigations show that most individuals do not evaluate an innovation on the basis of scientific studies of its consequences, although such objective evaluations are not entirely irrelevant, especially to the very first individuals who adopt. Instead, most people depend mainly upon a subjective evaluation of an innovation that is conveyed to them from other individuals who have already adopted the innovation. This dependence on experience of near peers suggests that the heart of the diffusion process consists of the modelling and imitation by potential adopters of their network partners who have previously adopted. Diffusion is a very social process that involves interpersonal communication relationships' (Rogers, 2003:18).

In an organisational context, Rogers' model of the diffusion process can be summarised in five stages based upon data from a large number of studies of diffusion of different types of innovation. Consideration is given to how these stages may relate to the adoption of agile methods by a software team in an organisation:

- *Agenda setting*, where needs and problems are identified and prioritized, and the environment is searched for innovations which might meet these needs and problems. Agenda setting in the context of a software development team might involve individuals looking for alternative software development approaches in order to address needs or problems within the software team's current process. At this point, individuals might considered agile methods as a solution.
- *Matching*, where the fit between the problem and the innovation is investigated. This might involve a consideration of risks and benefits and a feasibility test. At this point, individuals in a software development team might attempt to match the problem against a perceived solution, for example, from specific practices found in agile methods.
- *Redefining/restructuring*, where 'the innovation is re-invented so as to accommodate the organization's needs and structure more closely, and when the organization's structure is modified to fit with the innovation. Both the innovation and the organization are expected to change, at least to some degree' (Rogers, 2003:424). At this point, the software

development team might attempt to adapt or adjust certain practices from agile approaches in order to address their needs and fit with their current working environment.

- *Clarifying*, where the innovation is put into more widespread use in an organisation and its meaning becomes clearer to the members of the organisation. This is the point at which the software development team gain full confidence in the agile practices they have adopted, and the agile method becomes widely accepted.
- *Routinizing*, where the innovation becomes totally integrated within the organisation. This is the point at which the agile method is implemented and accepted by all members organisation - individuals are fully committed to its usage. The agile method is fully accepted as the software development methodology.

In these stages above, the consideration given to the adoption and acceptance by a software development team is purely speculative; this speculation therefore leads to the third research question:

- How does the adoption of agile methods compare with the established Rogers' model of diffusion?

It is the first three stages of this model in particular (agenda setting, matching, and redefining/restructuring) that is, in part, the focus of this research in order to explore and compare Rogers' model with the adoption of agile approaches to software development in organisations.

Methodology

As part of the literature review, I have investigated research methods to determine what I believe (and through discussions with my supervisors) are appropriate methods in order to address the research questions. I have reviewed a number of well-known sources of literature that describe research methods, however, three texts have provided particular insight into the methods that can be employed; these are:

1. Robson, C. (2002). Real World Research. Oxford, Blackwell Publishing Ltd.
2. Denzin, K. N. and Lincoln, S. Y. (2000). Handbook of Qualitative Research. London, SAGE Publications.
3. Miles, M. and Huberman, M. (1994). Qualitative Data Analysis. London, SAGE Publications.

It is expected that the final thesis will expand on this section and describe in greater detail the research methods examined and considered; however, PART IV of this report focuses on the research approach and explains the reasons and justifications for choosing particular research methods.

Summary

It was found during the review of the literature, that a significant proportion of literature regarding agile methods focused on the agile versus traditional methodology debate, in particular, the conflicting views surrounding the benefits and constraints raised by proponents of both approaches. Furthermore, the agile approach which has received the most attention by far has been XP.

What is certainly apparent from reviewing the state-of-the-art is that most of literature concerning agile methods is strongly subjective, that is, most of the evidence supporting agile methods consists mainly of claims and assertions based upon anecdotal evidence from proponents of agile methods. This point, coupled with the fact that there is not much hard evidence from empirical or academic literature regarding agile methods, raises questions as to why organisations adopt agile methods. Nonetheless, with the increasing number of reports emerging from organisations adopting agile approaches (see, for example, Appendix A) it is reasonable to infer that agile methods continue to gain popularity and sustain a steady rate of diffusion within the practitioner community despite the dearth of hard evidence.

It is this phenomenon that this research wishes to explore.

PART III

In Part III of this report, the results of two preliminary investigations in collaboration with my supervisors are described. I begin by presenting some background to the investigations; I then summarise the preliminary studies and the motivations for each; finally, I relate these investigations to the proposed research questions. The full details of the preliminary investigations are found in the appendices.

PRELIMINARY RESEARCH FINDINGS

The type of evidence which persuades organisations to adopt new technologies has provoked interest in the empirical software engineering research community (Segal et al., 2005). Despite the growing body of evidence produced by empirical software engineers, there is concern that industry is taking little notice (see, for example, Kitchenham, 2002; Zelkowitz et al., 2003; Glass, 2005). According to Zelkowitz et al. (2003), practitioners believe it is necessary to adopt new technologies to improve their practice but take little notice of academic results in so doing. Moreover, Zelkowitz et al. believe the foremost reason is the apparent 'disconnect' between the research and practitioner communities, and the lack of respect and trust between both. Glass et al. amplifies this point:

'There is a severe decoupling between research in the computing field and the state of practice in the field' (Glass et al., 2002:505).

Lanubile (1997) states that the research community has attempted to advance technology with proposals to the software industry, however few technologies have been widely accepted:

'In recent years, many new technologies have been proposed [by the research community] for developing and maintaining better software, on time and within budget. Some of these technologies have been introduced into production environments, but only a few have been widely accepted in the software industry' (Lanubile, 1997:97).

According to Vallet (1997), in order for the results of empirical research to have impact on practitioners, researchers need to collaborate more closely with those practitioners:

'Another barrier to the use of empirical studies in practice is that the current model for transitioning empirical studies to practice is incorrect. The implicit

model is that researchers will perform research studies, publish the results, and industry will then pick up the results and use them. This paradigm calls for too big of a jump between the research study and the industrial application. In fact, a better model would be for industry to work with research organizations to experiment with new techniques in industrial settings before attempting wide-spread dissemination within an organization. These pilot studies within organizations would allow the research community to work with and understand the impacts of the techniques in practice, while allowing the industrial community to tailor the techniques and provide feedback to the research community' (Jon Valett, 1997:140).

Supporting Vallet's viewpoint, Dittrich et al. (2005) suggests a research method called Co-operative Method Development (CMD) to engage practitioners during research:

'CMD takes the existing practice of software development in concrete industrial settings as a starting point. This enables us to address the actual problems that had been encountered, and it allows us to generalise technical and methodological recommendations that are rooted in successful practices. The research is implemented as evolutionary cycles consisting of qualitative empirical research, technical and methodological innovation in co-operation with the involved practitioners...' (Dittrich et al., 2005:1).

Segal believes the gap between empirical software engineering and practice might be lessened if more attention is paid to two important aspects of evidence:

'The first is that evidence from case or field studies of actual software engineering practice is essential in order to understand and inform that practice. The second is that the nature of evidence should fit the purpose to which the evidence is going to be put' (Segal, 2003:1).

Furthermore, Pfleeger and Winifred argue that:

'Researchers must learn to produce evidence that is useful for practitioners and credible to both' (Pfleeger and Winifred, 2000:31).

After conducting empirical investigations into 13 organisations, Rainer et al. found that practitioners wanted to see more evidence, however they did not necessarily accept this evidence:

'Our main finding is that there is an apparent contradiction between developers saying that they want evidence for software process improvement, and what developers will accept as evidence. This presents a serious problem for research: even if researchers could demonstrate a strong, reliable relationship between software process improvement and improved organisational performance, there would still be the problem of convincing practitioners that the evidence applies to their particular situation' (Rainer et al., 2003:1).

Robinson (2001) believes that more empirical investigations of practice are required in order to gain enough information with respect to the impact of empirical studies on practice:

'...we need considerably more empirical studies of practice. We simply don't have enough information about the actuality of practice to be certain that our research efforts are addressing the significant problems of a practice-oriented discipline' (Robinson, 2001:111).

These viewpoints significantly highlight the point that empirical software engineers need to engage more closely with practitioners in order to:

- develop research investigations that produce evidence persuasive and useful to practitioners;
- address areas of software engineering that are of interest to practitioners;
- communicate the results and evidence of empirical software engineering investigations suitably to practitioners;
- promote a collaborative approach between empirical software engineers and practitioners to benefit both communities, and to lessen the gap between research and practice.

With respect to agile methods, Jeffries (2005) claims that:

'To really take an important role in practice, ideas need to get built into people's heads. Since most practitioners don't read much, they get most of their ideas from people they talk to, things they can read quickly, and most of all, from ... practice' (Jeffries, 2005).

According to Jeffries' quote, practitioners get most of their ideas from talking to other people around them. This point is further highlighted by Rogers (2003) who states:

'Most individuals evaluate an innovation not on the basis of scientific research by experts but through the subjective evaluations of near peers who have adopted the innovation' (Rogers, 2003:36).

These statements suggest practitioners might be more persuaded by the evidence produced by empirical software engineers, if investigations such as field or in-depth case studies report experiences similar to those experiences described by near peers. As Segal states:

'Richly contextual case studies, where practitioners can recognise a situation similar to their own, might evoke the gut instinct "it worked there; it will work here. I must try it". For example, we speculate that the rapid growth in agile methodologies is not due to the presentation of any hard empirical evidence (of which there is currently not very much), but to practitioners' gut reactions on hearing of other's experiences: "yes, this makes sense"' (Segal, 2003:43).

Given the suggestions above as to the type of studies which might help reduce the gap between empirical software engineering and practice, we decided to investigate the type of studies and evidence published in the Journal of Empirical Software Engineering.

Furthermore, given that agile methods emerged from the practitioner community, and that there is a dearth of empirical evidence elucidating the benefits and constraints of agile methods, we investigated why organisations adopt agile methods.

The following two sections summarise these investigations.

The Type of Evidence Produced by Empirical Software Engineers

In order to investigate the type of studies and evidence reported by empirical software engineers, we examined the research published between the years 1997 and 2003 inclusive in the journal of Empirical Software Engineering, drawing on the taxonomy developed by Glass et al. (2002). Using this taxonomy, our investigation classified the journal papers by topic, research method, research approach, reference disciplines, units of analysis, and type of author(s). We found that the research in the journal was somewhat narrow in topic with around half the papers focusing on measurement/metrics, review and inspection; that researchers were almost as interested in formulating as in evaluating; that hypothesis testing and laboratory experiments dominated evaluations; that

research was not very likely to focus on people and extremely unlikely to refer to other disciplines.

This investigation partially addresses the second research question in this proposal:

- What are the sources and types of evidence that practitioners use in order to decide whether or not to adopt agile methods?

With the arguments presented at the beginning of section, that is, that practitioners do not appear to look at scientific and academic studies as their source of evidence, we believe that our investigation begins to provide some explanation as to why. For example, we found many papers in the journal describe experiments conducted in artificial lab-type settings, therefore, practitioners might view evidence from these experiments as inappropriate as they were not carried out within a real-life context. Furthermore, we found the journal papers address a narrow range of topics, therefore the areas empirical software engineers investigate might not be of interest to practitioners. Therefore, in view of these findings, the second research question is considered to be appropriate in order to determine the sources of information and types of evidence practitioners use to decide whether to adopt agile methods.

Full details of this investigation can be found in the paper (Segal et al., 2005) in Appendix D.

Why Do Organisations Adopt Agile Methods? An Initial Study

The aim of this investigation was to begin to identify the factors influencing organisations to adopt agile approaches. The study was based upon literature analysis and analysis of responses to an informal question posed to three popular agile related mailing lists. We examined literature from the Agile Alliance Article Library which contained a vast collection of articles relating to agile development practices. These articles included publications from academic journals and less formal experience reports from software development practitioners.

In order to obtain further experience reports we posted a simple question to three popular agile methods online discussion mailing lists. These three mailing lists were selected on the prevalence and frequency of agile related online discussions posted on a daily basis.

Our findings reveal that responding to recurrent project failure, and the influence of internal and external individuals, are the major factors in persuading organisations to adopt agile methodologies.

This investigation partially addresses the first research question in this proposal:

- What are the factors that influence organisations to adopt agile methods?

By examining literature describing experiences of adopting agile methods, we were able to begin to identify the factors that influence organisations to adopt agile methods; however, our approach was limited to literature analysis. Therefore, using other methods for data collection it is hoped that a greater insight can be gained into the factors persuading organisations to adopt agile methods.

Full details of this investigation can be found in the paper (Grinyer et al., 2006) in Appendix E.

PART IV

In PART IV, I summarise the intended research approach; I then describe the data collection methods in more detail; next, I list the expected work to be completed as individual task units, and finally, I present a graphical representation of the research task units as a Gantt chart.

RESEARCH APPROACH

The questions raised in this proposal are, in part, as a result of gaps identified during the review of the literature and the preliminary research findings described in PART III. In order to address these questions, a number of data collection methods are incorporated into the research approach: these are, survey questionnaires, semi-structured interviews, and observations.

A convention is used in the following sections to distinguish between pilot investigations and the actual investigations to be conducted. Piloting will simply be referred to as a 'pilot', however actual data collection activities conducted after the piloting stages will be referred to as the 'real' studies. This convention differentiates the data collection testing stages from the real-life data collection of the proposed research.

Summary of Research Methodology

The research methodology is a flexible qualitative design approach of an exploratory nature. Such a design, according to Robson aims to find out what is happening, particularly in little-understood situations (Robson, 2002:59). He states:

'If, for example, the main purpose [of the research] is exploratory, trying to get some feeling as to what is going on in a novel situation where there is little to guide what one should be looking for, then your initial approach will be highly flexible' (Robson, 2002:182).

The data to be collected is expected to be qualitative from responses to open and closed questions in survey questionnaires, semi-structured interviews, and observations. According to Creswell:

'The qualitative study approach is considered an appropriate method when little is known about the phenomenon under investigation and the concepts are immature due to the lack of theory and previous research and a need exists to explore and describe the phenomena' (Creswell, 1994:145).

It is believed Creswell's statement is particularly appropriate given that agile methods are new in relation to the evolution of software engineering, and the paucity of empirical evidence supporting agile methods in organisations (Lindvall, 2002).

The research approach can be modelled as shown in figure 1:

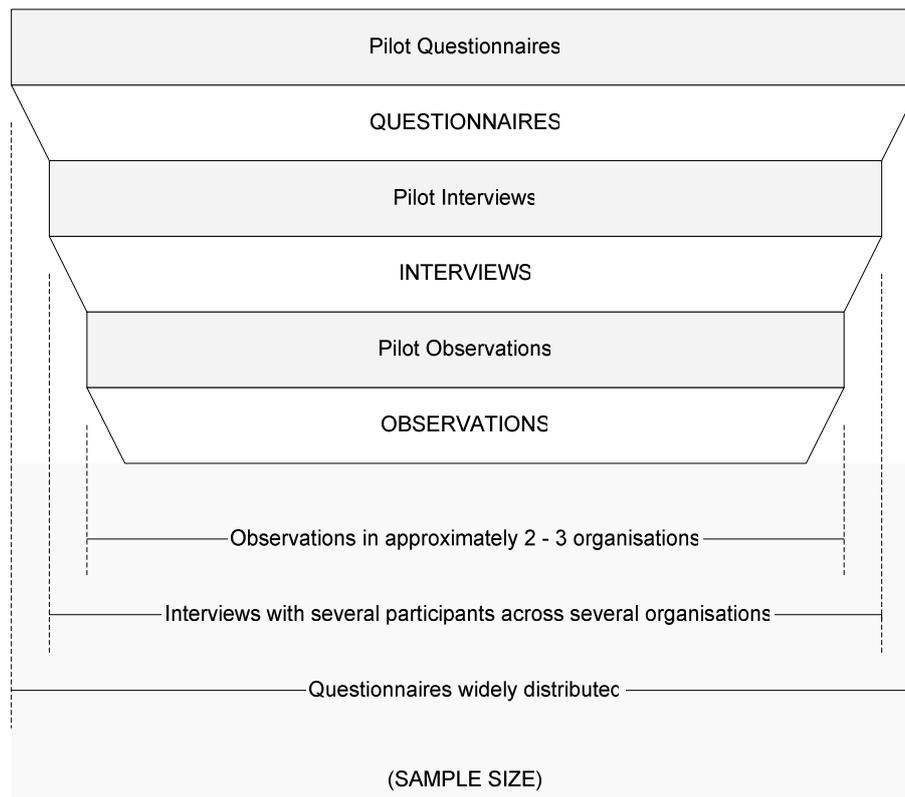


Figure 1. Model of Research Approach.

The findings from the preliminary studies described in PART III, in particular the investigation into the adoption of agile methods in organisations, elicit a number of unanswered questions. These questions and the review of the literature provide, in part, the basis for the research model shown in figure 1, and will be used to inform the design of the questionnaires for the first data collection activity.

The data collection methods shown in figure 1 are represented in an upturned pyramid structure governed by the sample size at each stage of the investigation. The sample size is applicable 'only' to the white layers in the model. The grey layers represent the piloting stages and are limited to a small number of

participants, and as the investigation progresses, so too will the sample size narrow.

The proposed course of the investigation will begin with the design and piloting of a survey questionnaire based on the questions raised in the preliminary studies in PART III, and the review of the literature. The purpose of piloting the questionnaires will be to provide details of any ambiguities in the questions that are asked, and to establish whether the answers provided the required information. After the questionnaires have been piloted and the responses have been returned, the results will be analysed and the questionnaire adjusted accordingly. These modifications will address aspects of the questionnaire that might prove problematic, for example, questions that might be unclear, that are misinterpreted, or questions that might not provide responses of much relevance. The design of the questionnaire will be modified where necessary in response to the analysis of the piloting stage; the questionnaires will then be distributed widely in order to target a large population of individuals who use or have used agile methods. This sample will include individuals on agile software development related mailing lists, discussion forums, organisations that use agile methods, and other sources that are identified. The potential sources for this investigation are addressed in a later section.

On receiving the responses from the questionnaire survey, the results will be analysed in order to disambiguate the questions raised in the preliminary studies and furthermore, to determine the extent in which the questionnaires answer the questions elicited in the investigations in PART III. This analysis of the questionnaires will provide the grounding for interview design for the next stage of the data collection. The main aims of the interviews will be to disambiguate the responses of the questionnaires and address questions that are not, or cannot be easily answered, by questionnaires. Furthermore, the interviews will expand on the questionnaires to provide a richer data collection method as more focus and time will be placed on individual questions with a narrower sample of participants.

The interviews will be piloted with around two to three participants that are 'close' to the research area, that is, participants that may have partial exposure to agile methods in organisations or have used agile approaches in the past. Like the procedures followed with questionnaires, the pilot interviews will be conducted with a small sample of participants. The results of the interviews will be analysed in order to address problematic areas with the questions asked, for example, if interviewees find questions difficult to understand. It is expected that piloting of the interviews will elucidate problems with the questions asked; therefore, the

interview design can be modified accordingly. For the real interviewing data collection activity, it is expected that several interviews will be conducted across several organisations, for example, three or four members of software development team within an organisation may be interviewed across several organisations. The interviewing technique will be semi-structured therefore the questions asked may deviate from the core set of questions, depending on the response to answers. After the interviewing data collection activity is complete the data will be transcribed, analysed, and then aggregated with the data collected from the questionnaires.

Following the analysis of the interviews and aggregation with the data collected from the questionnaires, observations are the final data collection method proposed. The aims of the observations are similar to the aims of the interviews in response to the questionnaires, that is, the observations will be conducted to address areas of the questionnaires and interviews that remain unanswered or incomplete; furthermore, observations will be conducted in order to expand on the findings of the previous data collection activities. The observations will also help to address discrepancies that might be found in both questionnaires and interviews, as Robson (2002) says:

'Interview and questionnaire responses are notorious for discrepancies between what people say that they have done, or will do, and what they actually did, or will do' (Robson, 2002:310).

Moreover, Robson states:

'A major advantage of observation as a technique is its directness. You do not ask people about their views, feelings or attitudes; you watch what they do and listen to what they say' And that, 'Data from direct observations contrasts with, and can often usefully complement, information obtained by virtually any other technique' (Robson, 2002:310).

In view of these statements, the triangulated data from the questionnaires and interviews may reveal that a particular phenomenon is ambiguous, and that observations might help explicate this phenomenon. For example, it may be that a number of interviews are conducted across several organisations and it is discovered that responses to a particular set of questions returns a very diverse result that is not disambiguated through interviewing. To clarify this point, a scenario is presented: A question in an interview asks 'Who in the software development team decides what software development practices are adopted?'

Responses to this question in the questionnaires and interviews might reveal that interviewees give very diverse answers which may not necessarily provide a clear result, however observations of the teams in their environment might provide evidence that project managers, for example, make most of the decisions in a particular software team. However, this is just arbitrary example.

The piloting of the observations will be with a small sample of one or possibly two organisations. Once the data from the observation pilots is collected, it will be analysed in order to inform the real observations. The final stage of the data collection will be the real observations across a few organisations. These observations will be 'unobtrusive observations' (Robson, 2002:310) whereby the research will be conducted from an 'observer-as-participant' position as described by Gold (1958) in Robson:

'This [the observer-as-participant] is someone who takes no part in the activity, but whose status as researcher is known to the participants' (Gold 1958, in Robson 2002:319).

The data collected from the observations will be analysed and triangulated with the data collected from both the questionnaires and interviews.

Due to the flexible nature of the research approach, the order in which the data collection methods are conducted is not necessarily final. For example, it is possible that observations might follow questionnaires depending on the responses of questionnaires; the results of observations may require further investigation using interviews, and more questionnaires might also be distributed as a result of interviews or observations. According to Robson, adapting the research approach according to the findings from the data already collected is an acceptable approach in flexible research designs:

'...in these [flexible] designs you don't have to foreclose on options about methods. Ideas for changing your approach may arise from your involvement and early data collection' (Robson, 2002:167).

Data Triangulation

According to Robson (2002), Triangulation

'...is a valuable and widely used strategy' And '...it involves the use of multiple sources to enhance the rigour of the research' (Robson 2002:174).

Specifically, data triangulation is defined as

'...the use of more than one method of data collection (e.g. observation, interviews, documents)' (Robson 2002:174).

One of the main benefits of data triangulation is that analysis of data is not based on a single data collection method; therefore, it is believed the uncertainty of the results is lessened by drawing upon multiple data collection methods. Robson (2002) states:

'One important benefit of multiple methods is in the reduction of *inappropriate certainty*. Using a single method and finding a pretty clear-cut result may delude investigators into thinking that they have found the "right" answer. Using other, additional, methods may point to different answers which remove specious certainty' (Robson, 2002: 370).

Furthermore, Stake (2000) considers data triangulation as a

'...process of using multiple perceptions to clarify meaning, verifying the repeatability of an observation or interpretation' (Stake 2000, in Denzin and Lincoln 2000:443).

As summarised in the previous model of the research approach, the research approach will incorporate data triangulation by drawing upon three data collection methods; these include:

- self-completion questionnaire-based survey,
- semi-structured interviews, and
- unobtrusive observations.

After each of the data collection activities, the data will be analysed independently and then compared with data collected in previous collection activities. For example, the responses to interview questions will be transcribed from audio recordings and then coded (Coding is described later in this section) for analysis; the results of this analysis will then be triangulated with the results of the questionnaires in a 'complementary fashion to enhance interpretability' (Robson, 2002:371).

In the context of empirical studies of software engineering, Seaman states that triangulation is an important tool for collecting evidence by drawing upon different methods; she says that, using triangulation

'The evidence might come from different sources, be collected using different methods, be analyzed using different methods, have different forms (interviews, observations, documents, etc.), or come from a different study altogether' (Seaman, 1999:569).

Data triangulation will be used throughout the data collection and analysis process to inform the next activities in the model of the research approach illustrated in the figure 1.

Sample Size

According to Robson (2002)

'It is difficult to pre-specify the number of observation sessions, interviews, etc. required in a flexible design study. Although the terminology differs in the different traditions, the basic notion is that you keep going until you reach "saturation". This is when further data collection appears to add little or nothing to what you have already learnt' (Robson, 2002:198).

This viewpoint takes into account that the data collected is analysed and interpreted as the research is conducted, such that

'...the data you have already collected throws up conjectures, suggests new themes, etc. which may call for further data collection' (Robson, 2002:199).

This is, in part, the motivation to adopt a pyramid approach whereby each data collection method informs the next data collection design and implementation, for example, the responses and results of the questionnaires inform the design of the interview questions.

It is proposed that the questionnaires will be distributed widely, in order to gather as much data as possible, and to help counter problems with potential low response rates notorious in survey questionnaires (Robson, 2002:238). The sample used for interviewing will be narrower and will focus on several interviews across several organisations. For example, five interviews may be conducted in each of four or five organisations participating; however, depending on

accessibility to the organisations involved in the study, there may be more interviews conducted in fewer organisations, or conversely, fewer interviews conducted in each organisation but across a greater number of organisations. The sample for the observation data collection activity will be considerably smaller using two or three organisations, as the scope of the research narrows. It is possible that observations need only address a few aspects of the initial findings from the previous analysis of the questionnaires and semi-structured interviews; therefore, it is deemed appropriate within the scope of the study that the sample for observations will focus on two or three organisations.

Scope

The data collection activities will target individuals who are exposed to, or have been exposed to, agile practices in organisations. The questionnaires will be distributed widely and may produce results from respondents across different organisational domains; however, it is expected that interviews and observations will target software engineering teams in an industrial and/or commercial setting. It is believed the advantage of this approach is to be as close as possible to a real-life setting. This viewpoint is in agreement with Seaman who states:

'Empiricists in software engineering often complain about the lack of opportunities to study software development and maintenance in real settings. This really implies that we must exploit to the fullest every opportunity we do have, by collecting and analyzing as much data of as many different types as possible' (Seaman, 1999:572).

Therefore, the scope of the investigation will be software engineering practitioners within an industrial and/or commercial setting. It is expected that the data collection activities will target organisations that are developing software using an agile method such as XP, DSDM, Scrum, etc. The potential organisations for this research are described in a later section.

Depending on the involvement of the participants, collaboration between the researcher and target organisations may incorporate levels of co-operative method development (Dittrich et al., 2005) to benefit both parties. For example, within the scope of the research methodology, the investigation might include feedback presentations and review meetings with the organisations involved, in order for the organisations to make use of the research findings.

Cross Organisation Approach versus Sequential Organisation Approach

An important aspect of the proposed research is how the data collection methods will be conducted with the organisations involved. The following two approaches have been considered:

1. The research methods are conducted across organisations in a concurrent manner. For example, interviews across several organisations are conducted around the same time during the investigation.
2. The research methods focus on one organisation at a time in a sequential manner. Therefore, questionnaires, interviews, and observations are conducted in one organisation until all the data is gathered, at which time, the next organisation is engaged for the next data collection activity.

It was decided that in order to gain a broader sample of responses from software engineering practitioners in organisations, (for example, a broad collection of responses to interview questions), the first approach listed above would be more suitable. It is believed that if approach 2) were selected the investigation could potentially become too focused on a specific organisation and therefore reflect more of a case study approach.

The aim of data collection methods, in particular the survey questionnaires, is to gather responses from practitioners across a wide distribution sample. Therefore, throughout the study, the data collection activities will be carried out across organisations, as opposed to concentrating on specific organisations in a sequential manner.

Organisational Context

A very important aspect to consider in this research proposal is the organisational context of the organisations that are included during the investigation. The organisational context can be viewed as the characteristics of the organisation, examples of which are the organisation's size or goals.

In this research, the organisational context may have a great influence on the adoption of agile methods; therefore, it is very important to analyse and relate the data collected back to the organisational context. To a limited extent, the questionnaires may provide some details about the organisational context of the respondents who have participated, depending on the questions asked; however,

it is more likely that the interviews and observations will provide much more detail regarding the organisational context.

According to Rogers' established model of diffusion (2003) characteristics of organisations, and therefore the organisational context, greatly affect the organisations' innovativeness, and thus might greatly affect the organisation's ability to adopt a new innovation such as an agile approach. In relation to structural characteristics of organisations, Rogers argues:

'...organization's structural characteristics such as *openness* (defined as the degree to which the members of a system are linked to other individuals who are external to the system) and *formalization* (defined as the degree to which an organization emphasizes following rules and procedures in the role performance of its members) were found to be related positively and negatively, respectively, to organizational innovativeness' (Rogers, 2003:408).

Furthermore, Boehm and Turner (2003d) identify five critical factors that they claim determine the suitability of agile or plan-driven methods in a particular project situation and indeed an organisation - size, criticality, dynamism, personnel, and culture. They place these five dimensions along five axes to characterise agile and plan-driven home grounds; they state:

'By rating a project along each of the five axes, you can visibly evaluate its home ground relationships. If all the ratings are near the centre, you are in an agile territory. If they are at the periphery, you will best succeed with a plan-driven approach' (Boehm and Turner, 2003d:57).

With these points in mind, for each participating organisation in this study, the organisational context, that is, the characteristics of the organisation, will be considered during the data analysis, in particularly when analysing the data collected from interviews and observations.

Pilot Studies

As shown in the research approach model in figure 1, pilot studies will feature strongly throughout the investigation. According to Robson:

'A pilot is a mini-version of the study carried out before committing yourself to the big one. This is done in part so you can sort out technical matters to do with methods of data collection to ensure that, say, the questions in the

questionnaire are understandable and unambiguous. Just as importantly, it gives you a chance to ensure you are on the right lines conceptually' (Robson, 2002:97).

Yin (1994, in Robson 2002), believe pilot studies help

'...investigators to refine their data collection plans with respect to both the content of the data and the procedures to be followed'. And, that pilots are a 'laboratory for the investigators, allowing them to observe different phenomena from many angles or to try different approaches on a trial basis' (Yin 1994:74 in Robson, 2002:185).

It is intended that pilot studies will precede each of the three data collection activities during the investigation, therefore three pilots will be conducted. Each pilot will focus on a small sample to test the design of each data collection method; therefore, the pilots will inform the design of the questionnaires, the semi-structured interviews, and the observations.

Data Collection and Analysis

The data collection techniques for the proposed research are self-completion questionnaires, semi-structured interviews, and observations.

Questionnaires

The purpose of the questionnaires is to address, and expand on, the findings and questions raised during the preliminary studies described in PART III and the review of the literature. A questionnaire survey will be the first data collection activity of the research approach, and the responses returned will inform the design of the semi-structured interviews in the next phase of data collection.

The questionnaires will be of an exploratory nature investigating the attitudes and beliefs of individuals using agile methods in software development organisations. Self completion questionnaires have been selected as they provide an '...approach to the study of attitudes, values, beliefs and motives' (Robson, 2002:234).

The sample size for the questionnaires will be large in order to collect as much data as possible, and to account for a potentially low response rate. According to Robson, low response rates can be a serious problem with self completion questionnaires:

'A low response rate is a serious and common problem with self-completion questionnaires...every effort should be made to get the rate up to an acceptable level' (Robson, 2002:238).

It is intended that questionnaires will be distributed widely to individuals using agile methods, and in particular, will be completed by participants in the organisations targeted for the semi-structured interviews and observations. The questionnaires will be distributed electronically to mailing lists, discussion forums, through emails, and also through letters to organisations; the aim of this is to achieve as high a return rate as possible.

Initially, it is proposed that the questions will be a combination of open and closed questions. These type of questions are now described.

Open Questions

An open question in a questionnaire is one where there are 'no restrictions on the content or manner of reply other than on the subject area' (Robson, 2002: 275). The open questions will allow the participant to answer questions in greater depth, in the hope to draw rich data for analysis.

To analyse the data retrieved from the open questions, a coding system will be adopted in order to categorise responses to questions, in order to explicate any phenomena that may arise, and to disambiguate and structure the responses.

Closed Questions

A closed question is one which forces the participant 'to choose from two or more fixed alternatives' (Robson, 2002: 275); therefore, the closed questions will relate directly to the research questions. Robson states:

'The survey questions should be designed to help achieve the goals of the research and, in particular, to answer the research questions' (Robson, 2002: 241).

Initially, the closed questions may be rated using an analysis scheme such as a Likert scale (Likert 1932 in Robson, 2002: 293). A Likert scale is a 'summated rating approach' to measuring attitudes and is considered a suitable approach for measuring attitudes in relation to statements (Robson 2002:293). However, suitable coding approaches for closed questions will also be considered during the questionnaire design.

Semi-structured Interviews

The purpose of the semi-structured interviews is to address the potential ambiguity and incomplete information that may arise from the questionnaires. Furthermore, the semi-structured interviews will expand on what is found during the analysis of the questionnaire responses. Semi-structured interviews will only be carried out in specific organisations, therefore the scope of the research will become more focused. The organisations used for this study, in particular for interviews and observations, will depend on accessibility and willingness of organisations to participate in the research.

It is expected that interview participants are part of a software development team; however, depending on the research findings, interview participants may include other non-technical personnel in the organisation. For example, the questionnaire responses from organisations included in the research study may indicate that other departments were involved in the decision to adopt agile methods (for example marketing or operations managers), therefore participants in other departments may be included in interviews.

There are two reasons for the sample size of semi-structured interviews being small, the first being that 'interviewing is time consuming' (Robson 2002: 273). For ethical reasons, the researcher will respect the time of the participants involved. Secondly, using a smaller sample size for the semi-structured interviews allows more time can be spent with each respondent; therefore, questions can be discussed and answered in greater depth with the intent to provide richer data (Robson, 2002: 199).

The choice to use semi-structured interviews was mostly driven by the exploratory nature of the research questions, and in part, by the literature supporting the use of interviewing techniques. According to Fontana and Frey (2000), interviewing is

'...one of the most common and powerful ways in which we try to understand our fellow human beings' (Fontana and Frey 2000, in Denzin and Lincoln, 2000:645).

Furthermore, Bell states that semi-structured interviews will retrieve information 'which can be analysed and patterns extracted and comparisons made' (Bell, 2002:13).

A semi-structured interview is one where the interviewer has a set of predetermined questions that are asked; however, the nature of the interview allows the interviewer to deviate from the core questions and indeed the order of the questions, to ask other questions to gain a richer insight, depending on the responses from the interviewees (Robson, 2002:270). This flexible approach to interviewing provided the motivation to select semi-structured interviews over other styles of interview such as unstructured or fully structured interviews (Robson, 2002:270).

The interviews will be audio recorded with informed consent from participants. This will enable the interviewer to concentrate on the interview responses and make notes without missing important information captured on tape. According to Lincoln and Guba, recording of data mechanically protects data in its 'raw' form eliminating the selective effect of researchers' perceptual skills (Lincoln and Guba 1985 in Seale, 1999: 148). The importance of audio recording is amplified by Patton, who considers the audio recorder as the 'indispensable equipment of researchers using qualitative methods' (Patton, 1990:348).

The data collected from the interviews will be coded and analysed. Furthermore, the data will be triangulated with the results of the questionnaires to elicit any phenomena that might occur. The data analysed from the interviews and questionnaires will be used to inform the design of the observation data collection method.

Observations

In this research, the purpose of the observations are to address any discrepancies from the analysis and findings in the data already collected, and to further expand on the findings. Moreover, the observations will be used to capture the behavioural aspects of people in the organisations in the study, in order to articulate the decision making processes by software engineering practitioners. It is proposed that observations will be conducted at decision making meetings that occur in organisations; however, depending on the course of the research other observations in different organisational settings may be conducted.

At this point, it is worth mentioning that observations in organisational settings, in particular decision meetings, may raise problems. For example, meetings in organisations may be confidential therefore only certain meetings can be attended. Meetings may not occur very frequently, therefore it is imperative that the researcher is aware of the meetings that can be attended. Furthermore, the design of the observations must consider the type of participants to observe, for

example, whether the observation should be of programmers alone, programmers and project managers, the whole team, and so forth. These considerations are among some of the issues that will be addressed when consulting the organisations willing to participate in the research.

Initially, it is intended that *unobtrusive* observations (Robson, 2002:310) will be conducted with the research being an observer-as-participant (Robson, 2002, 319). Robson states:

'As the actions and behaviour of people are central aspects in virtually any enquiry, a natural and obvious technique is to watch what they do, to record this in some way and then to describe, analyse and interpret what we have observed' (Robson, 2002:309).

Furthermore, Adler and Adler, characterize observations 'as the fundamental base of all research methods' (Adler and Adler 1994:389 in Denzin and Lincoln, 2002:673). Therefore, observations have been chosen to complement the data triangulation aspects of the research and to address discrepancies that might occur from questionnaires and interviews. Robson states:

'Data from direct observation contrasts with, and can often usefully complement, information obtained by virtually any other technique' (Robson, 2002:310).

The data gathered in the observations will be captured in field notes (Seaman, 1999:559) and coded during the analysis. Moreover, the observation data will be triangulated with the data from the previous data collection methods for further interpretability.

Coding

According to Robson, a code is

'...a symbol applied to a section of text to classify or categorise it' (Robson, 2002:477).

Miles and Huberman (1994) say codes are labels or tags

'...assigning units of meaning to the descriptive or inferential information compiled during a study' (Miles and Huberman, 1994:56).

A coding system will be created in order to understand meaningfully the qualitative data derived during the data collection activities. There are a number of classifications for coding, which are now summarised in the following sections.

Provisional Codes

As recommended by Miles and Huberman (1994:58) a provisional list of codes can be created from the research questions and problem area. It is expected that the provisional list of codes will evolve and change as the analysis of data progresses; however, it is also believed that codes may be used from other categorisation systems that are found and are pertinent to the analysis of the data collected. For example, Rogers' (2003) classification of adopters (innovators, early adopters, early majority, late majority, and laggards) may be used as codes like INNT, ELAD, EMAJ, LMAJ, and LAGG to classify organisations in this study by their adopter characterisation i.e. when they first adopted an agile approach to software engineering.

Descriptive Codes

Miles and Huberman (1994:57) describe 'descriptive codes' as codes which require little interpretation. For example, the code 'ELAD' as shown in the previous section, would signify that an organisation was an **EarLy AD**opter of agile methods. These descriptive codes will be used for the data analysis throughout the investigation.

Inferential Codes

Inferential codes are used *interpretively* in order to discover inferential, complex and thematic links in the data (Miles and Huberman, 1994).

Inferential coding requires handling the same piece of data more interpretively (Miles and Huberman, 1994:57). It requires the researcher to reread coded data in order to see if another code becomes apparent that may suggest a theme that links data together (Miles and Huberman, 1994:57).

It is expected that inferential codes might emerge as a result of data triangulation when the data collected from each of the collection methods are combined and compared.

Credibility and Trustworthiness

The trustworthiness of findings from flexible, qualitative research is criticised by fixed design experimentalists for the lack of 'standard' means of assuring reliability and validity (Robson, 2002:168).

However, Kirk and Miller (1986, in Robson 2002:169) suggest the methods and techniques used in fixed design research are inappropriate to flexible qualitative research, therefore different measures for ensuring trustworthiness are called for.

As this study is of a flexible qualitative design, investigations will follow the procedures described below in order to reduce threats to the credibility and trustworthiness of the findings. Moreover, with the use of triangulation it is hoped threats of validity are further reduced. Robson says:

'Triangulation can help to counter all of the threats to validity' (Robson, 2002:175).

Audio/Video Recording

Audio recording will be used when conducting semi-structured interviews and video taping for observations where appropriate, in order to reduce the threat of providing inaccurate and incomplete data. The use of audio and video taping will enable the author to provide a valid description of what is seen and heard (Maxwell 1992 in Robson, 2002:171).

Interpretation

The interpretations of the data will be described in detail in order to clearly demonstrate how interpretations are reached (Mason 1996 in Robson, 2002:171). This will include checking at regular intervals the appropriateness of the approach used (Maxwell 1992 in Robson, 2002:171) and continually noting and justifying the steps through which the interpretation will be made (Mason 1996 in Robson, 2002:171). Furthermore, the interpretations will be checked with my supervisors who will independently interpret samples of data to see whether their interpretations match mine.

Audit Trail

A full up-to-date record of the activities carried out during the research will be made and kept. This will include data from the self completion questionnaires, transcripts of the semi-structured interviews, and field notes from observations in order to reduce the effect of researcher bias (Robson, 2002:176).

Particularly, the aim of the audit trail will be to improve the analysis task and to deepen confidence in the final conclusions (Miles and Huberman, 1994:286).

Ethical considerations

The ethical considerations during this research will address concerns with respect to the participants and the target organisations. Denzin and Lincoln (2000) insist that when the objects of enquiry are human beings 'researchers must take extreme care to avoid any harm to them' (Denzin and Lincoln, 2000:662).

Moreover, Stake (in Denzin and Lincoln, 2000:447) suggests that an informal contract should exist between the researcher and the researched that is protective to the well being of the participants. Therefore, it is proposed that a formal non-disclosure agreement between the researcher and participants will be sought before collecting data. Organisations will also be consulted and involved in the reporting of the evaluation in order not to harm the organisation.

The informed consent of participants will be sought and participants will be informed that they are free to withdraw consent and withdraw from partaking in the study at any time, without prejudice to them. Participants will have the option of remaining anonymous when completing the self completion questionnaires, when participating in the semi-structured interviews, and during observations. Informed consent from participants concerning audio recording will be sought during interviews and observations.

If it is deemed necessary to use quotes and comments in the presentation of results, then authorisation will be sought. Confidentiality will be maintained at all times. Feedback will be given to participants regarding the findings and the participants will be made aware of this fact from the outset. Participants will receive a copy of the findings if requested. The dissemination of findings will be negotiated with the organisations involved, therefore, participants will be informed of how the evaluation findings will be disseminated and to whom.

Potential Candidates for Research

It is important to identify candidate participants for this research, in particular for the proposed interview and observation data collection methods; therefore, I present a summary of potential candidates for the research:

I am treasurer and a committee member for an agile methods interest group in the north-west of England called 'AgileNorth' – a branch of the British Computer Society's (BCS) Software Practice Advancement (SPA) group. The interest group meet on a monthly basis to discuss all aspects of agile approaches which include activities such as book reviews, presentations, invitations of guest speakers, coding dojos (dojo is a martial arts training area), general debates on agile

practices, and other interesting issues relating to agile methods. In 2005, the AgileNorth committee successfully organised a conference called AgileNorth 2005 which attracted considerable interest. Within the constraints of the location, we received over 60 conference delegates from around 30 organisations, including reputable companies such as Microsoft, British Aerospace, and BT. Furthermore, we received delegates from well known organisations who support agile methods, including ThoughtWorks and Exoftware. Another conference is planned for 2006.

The AgileNorth group currently boasts between 40 – 50 members, and with the success of the conference in 2005, AgileNorth has made many more contacts through networking. These include other agile interest groups such as 'Agile Scotland' and 'Agile North Wales and Liverpool'.

It is hoped that my involvement with AgileNorth will enable me to find participants for this research, given that 30 organisations attended the conference; however, it is not necessarily correct to assume that all these organisations are using agile methods. As stated in my research approach, significant time will be set aside to identify and confirm the participants and organisations that are accessible and willing to partake.

Other possible sources are my contacts at work - I am a senior software engineer and consultant for a software development company which means, as part of my role, I sometimes work in other software development companies. During this time, I have encountered some individuals and organisations that use or have used agile methods. Through my work, I have found two organisations in UK and one in Helsinki, Finland, that use agile methods. One of the UK companies use DSDM, the other company uses XP. The organisation in Helsinki, Finland, uses agile software development practices drawn from both the XP and Scrum. Furthermore, in the company for which I work there are a few employees that have used and are using some agile practices during software development – these individuals might be useful candidates for pilot studies; however, consideration of the participants involved at each stage of the research will be addressed during the design of each data collection method (such as the questionnaire).

It is hoped, using the sources described above, that a number of participants and organisations will be willing to partake in this study in order to conduct the data collection methods described in the research approach.

RESEARCH TIME PLAN

The following section describes the research work broken down into individual task units reflecting the milestones and outcomes. A graphical representation of the task units is then presented in a Gantt chart at the end of this section.

Task Units

The tasks are split into phases:

- Ongoing Phase
- Data Collection & Analysis Phase
- Interpretation & Writing Phase

The *Ongoing Phase* describes the tasks that continue throughout the lifetime of the research; the *Data Collection & Analysis Phase* describes the tasks relating to the data collection methods and the analysis of the collected data, and the *Interpretation & Writing Phase* describes those tasks related to interpretation of the data and the major writing tasks.

The task units do not account for time allocated for conference or seminar attendances, or writing and submission of papers to conferences and/or journals that might occur during the research. Furthermore, the task units reflect the work to be completed and not the work that has been done already; however, Task Unit 4 in the *Data Collection & Analysis Phase* is now complete (described below) and precedes the questionnaire data collection task, therefore it is included.

The task units and in particular the time estimates for planning the research tasks are, in part, based upon Section 3 'Planning and Organizing a Research Project' of the U500 Open University module booklet: 'Doing Academic Research'

Note - task units are abbreviated and cross referenced using the following convention:

- Task Unit 1 = TU1;
- Task Unit 2 = TU2;
- Task Unit 3 = TU3, and so forth.

Furthermore, the time/duration unit used for the task units is months.

Ongoing Phase

The Ongoing Phase describes the tasks that are continued throughout the lifetime of the research process.

Task Unit 1

Description: Reviewing of literature.

Duration: Until thesis submission.

Dependencies: n/a

Outcome: A chapter or chapters in the thesis.

Status: Ongoing.

Task Unit 2

Description: Write early drafts of thesis chapters.

Duration: Until thesis submission.

Dependencies: n/a

Outcome: Thesis chapters.

Status: Ongoing.

Task Unit 3

Description: Analysis and interpretation.

Duration: Until thesis submission.

Dependencies: n/a

Outcome: Reporting of results, findings, and conclusions.

Status: Ongoing.

Data Collection & Analysis Phase

The Data Collection and Analysis Phase describes the tasks addressing the data collection and analysis.

Task Unit 4

Description: Explore agile methodology adoption stories in the literature.

Duration: n/a

Dependencies: n/a

Outcome: A paper reporting the findings. (see Appendix E)

Status: Complete.

Task Unit 5

Description: Design questionnaires for piloting, including reviews and revisions.

Duration: 2 months.

Outcome: Questionnaire for pilot study.

Dependencies: TU4

Status: To do.

Task Unit 6

Description: Identify and confirm candidate participants for questionnaires.

Duration: 2 months.

Outcome: A collection of sources to distribute the questionnaire, such as mailing lists, discussion forums, and participating organisations.

Dependencies: TU5

Status: To do.

Task Unit 7

Description: Pilot questionnaire.

Duration: 1 month.

Outcome: Responses from questionnaire returned.

Dependencies: TU6

Status: To do.

Task Unit 8

Description: Analyse and revise questionnaire design with respect to outcome of pilot.

Duration: 1.5 months.

Outcome: Revised questionnaire for real study.

Dependencies: TU7

Status: To do.

Task Unit 9

Description: Distribute questionnaires for real data collection activity.

Duration: 1.5 months (for data collection).

Outcome: Data from real study questionnaires returned.

Dependencies: TU8

Status: To do.

Task Unit 10

Description: Data analysis of questionnaires.

Duration: 4 months.

Outcome: A report analysing the results of the questionnaires.

Dependencies: TU9

Status: To do.

Task Unit 11

Description: Design semi-structured interview for piloting, including reviews and revisions.

Duration: 2 months.

Outcome: Semi-structured interview design for pilot study.

Dependencies: Outcome and findings from TU10 to inform design.

Status: To do.

Task Unit 12

Description: Identify and confirm candidate organisations & participants for interviews.

Duration: 2 months.

Outcome: Several participants identified across several organisations.

Dependencies: TU11

Status: To do.

Task Unit 13

Description: Pilot interviews.

Duration: 1 month.

Outcome: Data collected from interviews.

Dependencies: TU12

Status: To do.

Task Unit 14

Description: Analyse and revise interview design with respect to outcome of pilot.

Duration: 1.5 months.

Outcome: Revised interview.

Dependencies: TU13

Status: To do.

Task Unit 15

Description: Conduct interviews in organisations for real data collection activity.

Duration: 1.5 months.

Outcome: Data from real interviews transcribed.

Dependencies: TU14

Status: To do.

Task Unit 16

Description: Data analysis of interviews & triangulation with questionnaires.

Duration: 4 months.

Outcome: A report analysing the results of the interviews and TU10.

Dependencies: TU10, TU15

Status: To do.

Task Unit 17

Description: Design observations for piloting, including reviews and revisions.

Duration: 2 months.

Outcome: Observation design for real study.

Dependencies: Outcome and findings from TU16 to inform design.

Status: To do.

Task Unit 18

Description: Confirm candidate organisations & participants for observations.

Duration: 2 months.

Outcome: Observable organisations identified and confirmed.

Dependencies: Observations will be in the same organisations used in TU12.

Status: To do.

Task Unit 19

Description: Pilot observations.

Duration: 1 month.

Outcome: Data collected from observation.

Dependencies: TU18

Status: To do.

Task Unit 20

Description: Analyse and revise observation design with respect to outcome of pilot.

Duration: 1.5 months.

Outcome: Revised observation method for real study.

Dependencies: TU19

Status: To do.

Task Unit 21

Description: Conduct observations in organisations for real data collection activity.

Duration: 1 month.

Outcome: Data from observations collected.

Dependencies: TU21

Status: To do.

Task Unit 22

Description: Data analysis of observations & triangulating with questionnaires & interviews.

Duration: 4 months.

Outcome: A report analysing the results of the interviews and TU10.

Dependencies: TU10, TU15, TU21

Status: To do.

Interpretation and Writing Phase

The interpretation and writing phase describes the interpretation and writing tasks after the data collection and analysis activities. However, data analysis will be conducted throughout the data collection period and not just at the end of the data collection phase i.e. within this phase only.

Task Unit 23

Description: Consolidate data analysis and interpretation of findings.

Duration: 4 months.

Outcome: A report describing interpretations of the data and findings.

Dependencies: T10, TU15, TU21

Status: To do.

Task Unit 24

Description: Relate findings to concepts/theories/research questions.

Duration: 2.5 months.

Outcome: A report of the results and findings section in the thesis.

Dependencies: T10, TU15, TU22, TU24 (if more research conducted)

Status: To do.

Task Unit 25

Description: Complete final chapters, results, and conclusions.

Duration: 12 months.

Outcome: Final draft of thesis ready for review.

Dependencies: TU25

Status: To do.

Task Unit 26

Description: Review and edit thesis iteratively to reach required standard – submit.

Duration: 2.5 months.

Outcome: Thesis ready for submission

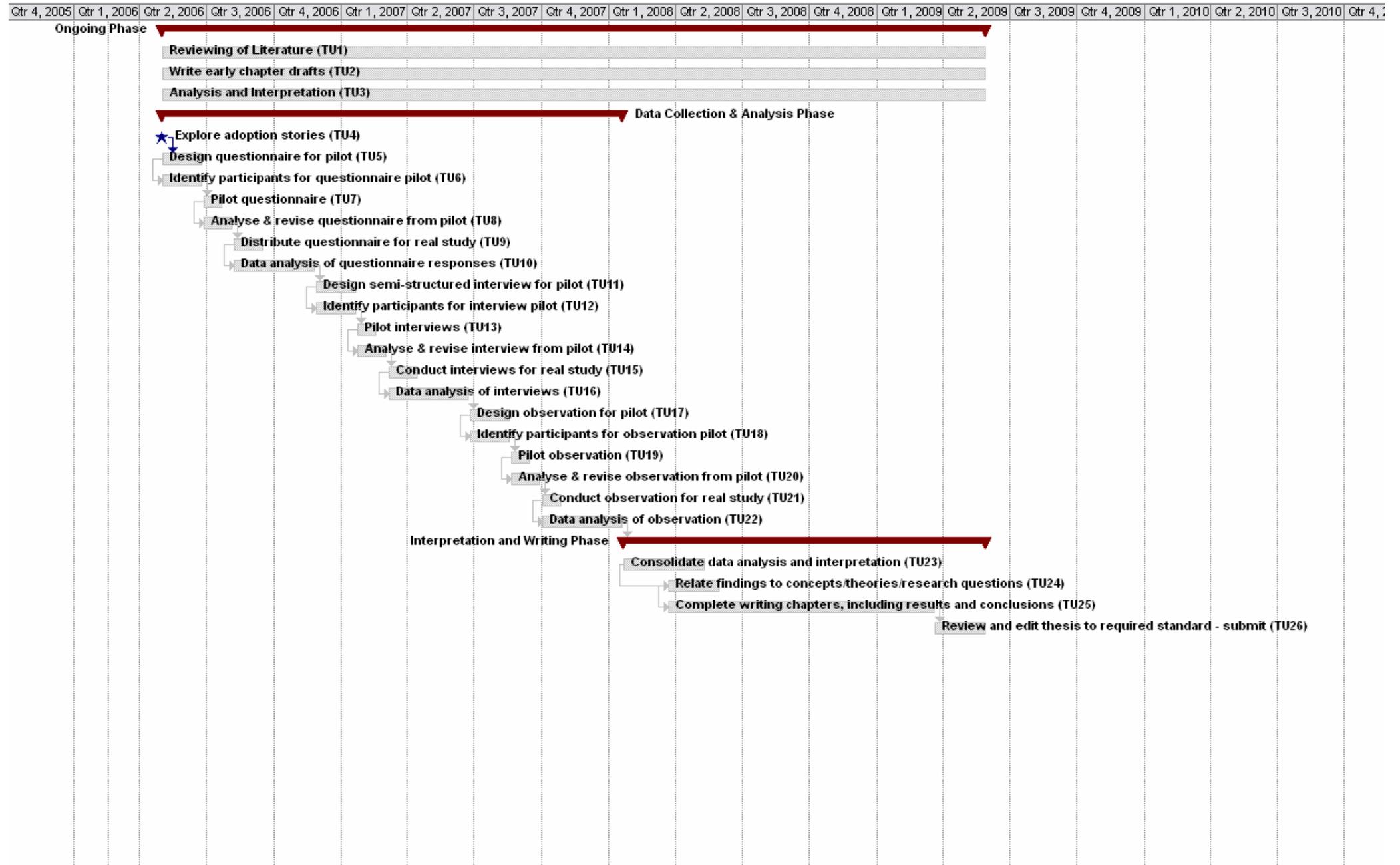
Dependencies: TU26

Status: To do.

Note – Reviews, revisions, and edits will continue throughout the research and development of the thesis, therefore TU26 above relates to the final reviews and edits that will be conducted.

Furthermore, analysis and interpretation will continue during the course of the research, therefore some tasks will overlap and some tasks will be conducted in parallel with each other.

Task Unit Gantt Chart



Task Unit Gantt Chart Data

Task Name	Duration	Start	Finish
- Ongoing Phase	160.8 wks	Mon 01/05/06	Thu 28/05/09
Reviewing of Literature (TU1)	160.8 wks	Mon 01/05/06	Thu 28/05/09
Write early chapter drafts (TU2)	160.8 wks	Mon 01/05/06	Thu 28/05/09
Analysis and Interpretation (TU3)	160.8 wks	Mon 01/05/06	Thu 28/05/09
- Data Collection & Analysis Phase	90 wks	Mon 01/05/06	Fri 18/01/08
Explore adoption stories (TU4)	0 wks	Mon 01/05/06	Mon 01/05/06
Design questionnaire for pilot (TU5)	8 wks	Mon 01/05/06	Fri 23/06/06
Identify participants for questionnaire pilot (TU6)	8 wks	Mon 01/05/06	Fri 23/06/06
Pilot questionnaire (TU7)	4 wks	Mon 26/06/06	Fri 21/07/06
Analyse & revise questionnaire from pilot (TU8)	6 wks	Mon 26/06/06	Fri 04/08/06
Distribute questionnaire for real study (TU9)	6 wks	Mon 07/08/06	Fri 15/09/06
Data analysis of questionnaire responses (TU10)	16 wks	Mon 07/08/06	Fri 24/11/06
Design semi-structured interview for pilot (TU11)	8 wks	Mon 27/11/06	Fri 19/01/07
Identify participants for interview pilot (TU12)	8 wks	Mon 27/11/06	Fri 19/01/07
Pilot interviews (TU13)	4 wks	Mon 22/01/07	Fri 16/02/07
Analyse & revise interview from pilot (TU14)	6 wks	Mon 22/01/07	Fri 02/03/07
Conduct interviews for real study (TU15)	6 wks	Mon 05/03/07	Fri 13/04/07
Data analysis of interviews (TU16)	16 wks	Mon 05/03/07	Fri 22/06/07
Design observation for pilot (TU17)	8 wks	Mon 25/06/07	Fri 17/08/07
Identify participants for observation pilot (TU18)	8 wks	Mon 25/06/07	Fri 17/08/07
Pilot observation (TU19)	4 wks	Mon 20/08/07	Fri 14/09/07
Analyse & revise observation from pilot (TU20)	6 wks	Mon 20/08/07	Fri 28/09/07
Conduct observation for real study (TU21)	4 wks	Mon 01/10/07	Fri 26/10/07
Data analysis of observation (TU22)	16 wks	Mon 01/10/07	Fri 18/01/08
- Interpretation and Writing Phase	70.8 wks	Mon 21/01/08	Thu 28/05/09
Consolidate data analysis and interpretation (TU23)	16 wks	Mon 21/01/08	Fri 09/05/08
Relate findings to concepts/theories/research questions (TU24)	10 wks	Fri 21/03/08	Thu 29/05/08
Complete writing chapters, including results and conclusions (TU25)	52 wks	Fri 21/03/08	Thu 19/03/09
Review and edit thesis to required standard - submit (TU26)	10 wks	Fri 20/03/09	Thu 28/05/09

PART V

Part V includes the references and appendices.

REFERENCES

Abrahamsson, P., Salo, O. and Ronkainen, J. (2002). Agile software development methods - review and analysis. Finland, Technical Research Centre of Finland.

Abrahamsson, P., Warsta, J., Siponen, M., et al. (2003). New directions on agile methods: a comparative analysis. Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon.

Agile Alliance (2001). Manifesto for Agile Software Development. <http://www.agilemanifesto.org/> (accessed 10/03/2005)

Ambler, S. (2003). Proof Positive. <http://www.sdmagazine.com/documents/s=8939/sdm0311f/sdm0311f.html> (accessed 10/01/2005)

Auer, K., Meade, E. and Reeves, G. (2003). The Rules of the Game. Proceedings of XP Agile Universe, New Orleans.

Baheti, P., Gehringer, E. and Stotts, D. (2002). Exploring the Efficacy of Distributed Pair Programming. Proceedings of XP Agile Universe, Chicago, IL.

Bailey, P., Ashworth, N. and Wallace, N. (2002). Challenges for Stakeholders in Adopting XP. <http://www.xpdreamteam.com/Default.aspx?tabId=90> (accessed 27/04/2004)

Basili, V. R. and Turner, J. A. (1975). "Iterative Enhancement: A Practical Technique for Software Development." IEEE Transactions on Software Engineering, December 1975, **1**(4): 390 - 396.

Beck, K., Cockburn, A., Highsmith, J., et al. (2001). Manifesto for Agile Software Development. <http://www.agilemanifesto.org/> (accessed 10/03/2005)

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

Beck, K. and Fowler, M. (2000). Planning Extreme Programming. Boston, MA., Addison-Wesley.

Beck, K. (2001). One Team. *Report*. <http://www.threeriversinstitute.org> (accessed 16/10/2003)

Beck, K. and Boehm, B. (2003). "Agility through Discipline: A Debate." *Computer*, June 2003, **36**(6): 44-46.

Beck, K. and Andres, C. (2004). Extreme Programming Explained: Embrace Change, Second Edition. Boston, USA, Addison-Wesley.

Bell, J. (2000). *Doing Your Research Project*. Buckingham, UK., Oxford University Press.

Boehm, B. (1981). Software Engineering Economics. NJ, USA, Prentice Hall.

Boehm, B. (1988). "A spiral model of software development and enhancement." Computer, May 1988, **21**(6): 61-72.

- Boehm, B. (2000). *Spiral Development: Experience, Principles, and Refinements*, *Special Report*, CMU/SEI-00-SR-08, ESC-SR-00-08.
- Boehm, B. (2002). "Get Ready for Agile Methods, with Care." *Computer*, January 2002, **35**(1): 64-69.
- Boehm, B. and Turner, R. (2003a). "Using Risk to Balance Agile and Plan-Driven Methods." *Computer*, June 2003, 36(6): 57-66.
- Boehm, B. and Turner, R. (2003b). Observations on Balancing Discipline and Agility. Agile Development Conference, Salt Lake City, Utah.
- Boehm, B. and Turner, R. (2003c). Rebalancing Your Organisation's Agility and Discipline. Proceedings of XP Agile Universe, New Orleans.
- Boehm, B. and Turner, R. (2003d). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, Addison Wesley.
- Booch, G. (1983). *Software Engineering with Ada*. US, Benjamin/Cummings.
- Braithwaite, K. and Joyce, T. (2005). XP Expanded: Distributed Extreme Programming. 6th International Conference, XP 2005, Sheffield, UK.
- Bratthall, L. and Jorgensen, M. (2002). "Can you Trust a Single Data Source Exploratory Software Engineering Case Study?" *Empirical Software Engineering*, 2002, 7(1): 9-26.
- Brewer, J. (2001). Extreme Programming FAQ. <http://www.jera.com/techinfo/xpfaq.html> (accessed on 12/06/2005)
- Brooks, P. F. (1975). *The Mythical Man-Month*. Reading, MA., Addison-Wesley.
- Charette, R. (2001a). "Fair Fight? Agile versus heavy methodologies." Cutter Consortium e-Project Management Advisory Service, 2001a, 2(13).
- Charette, R. (2001b). "The fight is on? Agile versus heavy methodologies." Cutter Consortium e-Project Management Advisory Service, 2001b, 2(14).
- Charette, R. (2001c). "Fists are flying? Agile versus heavy methodologies." Cutter Consortium e-Project Management Advisory Service, 2001c, 2(17).
- Cioch, F. A., Brabbs, J. and Kanter, S. (1994). Using the spiral model to assess, select and integrate software development tools. Proceedings of the Third Symposium on Assessment of Quality Software Development Tools, Washington, DC.
- Charette, R. (2001d). "The decision is in: Agile versus heavy methodologies." Cutter Consortium e-Project Management Advisory Service, 2001d, 2(19).
- Coad, P., LeFebvre, E. and De Luca, J. (2000). *Java Modeling in Colour With UML: Enterprise Components and Process*. NJ, Prentice Hall.
- Cockburn, A. (2002). *Agile Software Development*. Boston, MA., Addison-Wesley.
- Cockburn, A. and Highsmith, J. (2001). "Agile Software Development: The People Factor." *Computer*, November 2001, 34(11): 131-133.
- Cockburn, A. (2004). *Crystal Clear: A Human-Power Methodology for Small Teams*, Addison-Wesley.

Cohen, D., Lindvall, M. and Costa, P. (2003). *Agile Software Development - A DACS State-of-the-Art Report*, Technical Report. Maryland, The University of Maryland.

Cohn, M. and Ford, D. (2003). "Introducing an Agile Process to an Organization." Computer, June 2003, **36**(6): 74-78.

Creswell, J. W. (1994). *Research Design: Qualitative and Quantitative Approaches*. Thousand Oaks, CA, Sage Publications Inc.

Denzin, K. N. and Lincoln, S. Y. (2000). *Handbook of Qualitative Research*. London, SAGE Publications.

DeMarco, T. (1979). Structured Analysis and System Specification. New York, Yourdon Press.

DeMarco, T. and Boehm, B. (2002). "The Agile Methods Fray." Computer, June 2002, **35**(6): 90-93.

Deursen van, A. (2001). *Customer Involvement in Extreme Programming*. Proceedings of XP Agile Universe, North Carolina.

Dittrich, Y., Ronkon, K., Lindeberg, O., et al. (2005). *Co-operative Method Development Revisited*. Proceedings of the International Conference of Software Engineering, St Louis, Missouri, USA.

DSDM Consortium (2003). *Guidelines For Introducing DSDM Into An Organisation: Evolving to a DSDM Culture*. Kent, UK. http://www.dsdm.org/en/products/white_papers.asp (accessed 10/03/2005)

Dyba, T., Kitchenham, B. and Jorgensen, M. (2005). "Evidence-Based Software Engineering for Practitioners." IEEE Software, January/February 2005, **22**(1): 58-65.

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

El Eman, K. (2003). "Agile Project Management." Cutter Consortium Agile Software Development and Project Management Advisory Service, 2003, 4(11).

Elsamadisy, A. and Schalliol, G. (2002). Recognizing and Responding to 'Bad Smells' in Extreme Programming. ICSE, Orlando, Florida.

Extreme Programming. (2002), <http://www.xprogramming.com/xpmag/whatisxp.htm> (accessed 01/07/2003)

Fredrick, C. (2003). *Extreme Programming: Growing a Team*. Proceedings of XP Agile Universe, New Orleans.

Fowler, M. (2002). *The New Methodology*. <http://www.martinfowler.com/articles/newMethodology.html#id109533>. (accessed 04/05/2004)

Gilb, T. (1977). Software Metrics. Cambridge, Massachusetts, Winthrop Publishers, Inc.

Gilb, T. (1981). "Evolutionary Development." ACM Software Engineering Notes, April 1981, **6**(2): 17.

- Gilb, T. (1988). Principles of Software Engineering Management. Reading, Massachusetts, Addison Wesley.
- Glass, R. (2001). "Agile Versus Traditional: Make Love, Not War!" *Cutter IT Journal*, December 2001, 14(12): 12-18.
- Glass, R. L., Vessey, I. and Ramesh, V. (2002). "Research in Software Engineering: an analysis of the literature." Information and Software Technology, 2002, (44): 491-506.
- Glass, R. L. (2005). "A Sad SAC Story about the State of the Practice." *IEEE Software*, Jul-Aug 2005, 22(4): 119-120.
- Grinyer, A., Segal, J. and Sharp, H. (2006). Why do organisations adopt agile methods? An initial study. Submitted to XP2006, Oulu, Finland.
- Hauge, H. J. (2002). How to Combine Software Process Improvement and Quality Assurance with Extreme Programming, *Diploma Thesis*. Norway, Norwegian University of Science and Technology.
- Highsmith, J. (1999). Adaptive Software Development: A Collaborative Approach to Managing Complex Systems, Dorset House Publishing Company.
- Highsmith, J. and Cockburn, A. (2001). "Agile Software Development: The Business of Innovation." Computer, 2001, **34**(9): 120-122.
- Highsmith, J. (2001). "The great methodologies debate: Part 1." *Cutter IT Journal*, 2001, 14.
- Highsmith, J. (2002a). "The great methodologies debate: Part 2." *Cutter IT Journal*, 2002, 15.
- Highsmith, J. (2002b). Agile Software Development Ecosystems. Boston, MA., Addison-Wesley.
- Highsmith, J. (2002c). "What is Agile Development?" *Crosstalk - The Journal of Defense Software Engineering*, October 2002, 15(10): 4-9.
- Huang, J. (2001). "Pair Programming on the C3 project." *Computer*, 2001, 34(2): 118-119.
- Huo, M., Verner, J., Zhu, L., et al. (2004). Software quality and agile methods. Proceedings of the 28th Annual International Computer Software and Applications Conference.
- Jacobson, I., Booch, G. and Rumbaugh, J. (1999). *The Unified Software Development Process*, Addison-Wesley.
- Jeffries, R. (2005). Practice: That's What We Do. <http://www.xprogramming.com/xpmag/jatPractice.htm> (accessed 12/02/2005)
- Jenson, R. W. (2003). A Pair Programming Experience. <http://www.stsc.hill.af.mil/crosstalk/2003/03/jensen.html> (accessed 02/05/2003)
- John, M., Maurer, F. and Bjomar, T. (2005). Human and Social Factors of Software Engineering - Workshop Summary. Proceedings of the International Conference of Software Engineering, St Louis, Missouri, USA.

Kitchenham, B., Pfleeger, S. L., Hoaglin, D. C., et al. (2002). "Preliminary Guidelines for Empirical Research in Software Engineering." *IEEE Transactions on Engineering Management*, August 2002, 28(8): 721-734.

Kitchenham, B., Linkman, S. and Fry, J. (2003). Experimenter Induced Distortions in Empirical Software Engineering. Proceedings of the 2nd workshop in the Workshop Series on Empirical Studies in Empirical Software Engineering, Italy.

Kruchten, P. (1999). The Rational Unified Process - An Introduction. Boston, MA., Addison-Wesley.

Kruchten, P. (2001). "Agility with the RUP." *Cutter IT Journal*, 2001, 14(12): 27-33.

Lanubile, F. (1997). "Empirical Evaluation of Software Maintenance Technologies." *Empirical Software Engineering*, 1997, 2(2): 97-108.

Larman, C. and Basili, V. R. (2003). "Iterative and Incremental Development: A Brief History." Computer, June 2003, **36**(6): 47-56.

Lethbridge, T. C., Sim, E. S. and Singer, J. (2005). "Studying Software Engineers: Data Collection Techniques for Software Field Studies." *Empirical Software Engineering*, 2005, 10: 311-341.

Lincoln, S. Y. and Guba, E. G. (1985). *Naturalistic Inquiry*. Thousand Oaks, CA., Sage Publications.

Lindvall, M., Basili, V. R., Boehm, B., et al. (2002). Empirical Findings in Agile Methods. Proceedings of XP Agile Universe, Chicago.

Lindvall, M., Muthig, D., Dagino, A., et al. (2004). "Agile Software Development in Large Organisations." Computer, 2004, **37**(12): 58-65.

Lippert, M., Becker-Pechau, P., Breitling, H., et al. (2003). "Developing Complex Projects Using XP with Extensions." *Computer*, June 2003, 36(6): 67-73.

Lycett, M., Macredie, R. D., Patel, C., et al. (2003). "Migrating Agile Methods to Standardized Development Practice." *Computer*, June 2003, 36(6): 79-85.

McCauley, R. (2001). "Agile Development Methods Poised to Upset Status Quo." *SIGCSE Bulletin*, 2001, 33: 14-15.

Miles, M. and Huberman, M. (1994). *Qualitative Data Analysis*. London, SAGE Publications.

Mills, H. (1973). "On the development of Large, Reliable Programs." IEEE Symp. Computer Software Reliability, 1973: 155-159.

Murru, O., Deias, R. and Mugheddu, G. (2003). "Assessing XP at a European Internet Company." *IEEE Software*, May/June 2003, 20(3): 37-43.

Nawrocki, J. and Wojciechowski, A. (2002). Experimental Evaluation of Pair Programming. <http://www.xprogramming.com/xpmag/whatisxp.htm>

Orr, K. (2002). "CMM Versus Agile Development: Religious Wars and Software Development." Agile Project Management, 2002, **3**(7): 1-29.

Oxford English Dictionary, 5th Series 2001.. Oxford, UK., Oxford University Press.

- Palmer, R. S. and Felsing, M. J. (2002). A Practical Guide to Feature-Driven Development, Prentice Hall.
- Parish, A., Smith, R., Hale, D., et al. (2004). "A Field Study of Developer Pairs: Productivity Impacts and Implications." *IEEE Software*, 2004, 21(5): 76-79.
- Patton, M. Q. (1990). *Qualitative Evaluation and Research Methods*, SAGE Publications.
- Paulk, C. M., Curtis, B., Chrissis, B. M., et al. (1993). *Capability Maturity Model for Software Version 1.1*, Technical Report, CMU/SEI-93-TR-024 & ESC-TR-93-177.
- Paulk, C. M. (2001). "Extreme Programming from a CMM Perspective." *IEEE Software*, November/December 2001, **18**(6): 19-26.
- Paulk, C. M. (2002). "Agile Methods and Process Discipline." *Crosstalk - The Journal of Defense Software Engineering*, October 2002, 15(10): 15-18.
- Petre, M., Bugden, D. and Scholtz, J. (2004). "A Focus on the Human Side of Software Engineering." *Empirical Software Engineering*, 2004, 9: 271-274.
- Pfleeger, S. L. and Winifred, M. (2000). "Marketing Technology to Software Practitioners." *IEEE Software*, January/February 2000, **17**(1): 27-33.
- Pfleeger, S. L. (2005). "The Role of Evidential Force in Empirical Software Engineering." *IEEE Software*, January/February 2005, 21(1): 66-73.
- Poppendieck, M. and Poppendieck, T. (2003). Lean Software Development: An Agile Toolkit, Addison-Wesley.
- Potts, C. (1993). "Software Engineering Research Revisited." *IEEE Software*, September 1993, 10(5): 19-28.
- Pressman, S. R. (2005). Software Engineering: A Practitioner's Approach. Sixth Edition. New York, McGraw-Hill.
- Raghavan, S. and Chand, D. (1989). "Diffusing Software-Engineering Methods." *IEEE Software*, 1989, **6**(4): 81-90.
- Rainer, A., Hall, T. and Baddoo, N. (2003). Persuading developers to "buy into" software process improvement: a local opinion and empirical evidence. *Proceedings of the International Symposium on Empirical Software Engineering*, Rome, Italy.
- Rainer, A., Jagielska, D. and Hall, T. (2005). Software engineering practice versus evidence-based software engineering research. *Proceedings of the REBSE workshop, associated with ICSE 2005*, St. Louis, Missouri, USA.
- Rasmusson, J. (2003). "Introducing XP into Greenfield Projects: Lessons Learned." *IEEE Software*, May/June 2003, 20(3): 21-28.
- Rifkin, S. (2001). "Why Software Process Innovations Are Not Adopted." *IEEE Software*, July/August 2001, 18(4): 112-111.
- Robinson, H. (2001). "Reflecting on research and practice." *IEEE Software*, Jan/Feb 2001, 18(1): 112-111.
- Robson, C. (2002). Real World Research. Oxford, Blackwell Publishing Ltd.

- Rogers, M. E. (2003). Diffusion of Innovations, 5th Edition. New York, The Free Press.
- Royce, W. W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. Proceedings of WESCON, Los Angeles.
- Schwaber, K. and Beedle, M. (2001). Agile Software Development with Scrum, Prentice Hall.
- Schwaber, K. (2004). Agile Project Management with Scrum. Redmond, Washington, Microsoft Press.
- Scotland, K. (2003). Agile Planning with Multi-customer, Multi-project, Multi-discipline Team. Proceedings of XP Agile Universe, New Orleans.
- Seale, C. (1999). The Quality of Qualitative Research. London, UK., SAGE Publications.
- Seaman, C. B. (1999). "Qualitative Methods in Empirical Studies of Software Engineering." IEEE Transactions on Software Engineering, July/August 1999, 25(4): 557-572.
- Segal, J. (2003). The Nature of Evidence in Empirical Software Engineering. Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice, Amsterdam, The Netherlands, IEEE CS Press.
- Segal, J., Grinyer, A. and Sharp, H. (2005). The type of evidence produced by empirical software engineers. Proceedings of the REBSE workshop, associated with ICSE 2005, St. Louis, Missouri, USA.
- Sepulveda, C. (2003). Agile Development and Remote Teams: Learning to Love the Phone. Agile Development Conference, Salt Lake City, Utah.
- Sharp, H., Robinson, H. and Woodman, M. (2000). "Software Engineering: Community and Culture." IEEE Software, January/February 2000, 17(1): 40-47.
- Sharp, H. and Robinson, H. (2003). XP Culture: Why the twelve practices both are and are not the most significant thing. Agile Development, Salt Lake City, Utah.
- Sharp, H. and Robinson, H. (2004). "An Ethnographic Study of XP Practice." Empirical Software Engineering, 2004, 9(4): 353-375.
- Sim, E. S., Singer, J. and Storey, A. M. (2001). "Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research." Empirical Software Engineering, 2001, 6(1): 85-93.
- Stapleton, J. (1997). DSDM: Dynamic Systems Development Method, Addison-Wesley.
- Stephens, M. and Rosenberg, D. (2003). Extreme Programming Refactored: The Case Against XP. US, Apress.
- Stotts, D., Williams, L., Nagappan, N., et al. (2003). Virtual Teaming: Experiments and Experiences with Distributed Pair Programming. XP Agile Universe, New Orleans.

- Sultan, F. and Chan, L. (2000). "The Adoption of New Technology: The Case of Object-Oriented Computing in Software Companies." *IEEE Transactions on Engineering Management*, 2000, 47(1): 106-126.
- Turner, R. and Jain, A. (2002). Agile Meets CMMI: Culture Clash or Common Cause? *Proceedings of XP Agile Universe*, Chicago.
- Tichy, F. W. (2000). "Hints for Reviewing Empirical Work in Software Engineering." *Empirical Software Engineering*, 2000, 5(4): 309-312.
- Valett, J. (1997). "The Practical Use of Empirical Studies for Maintenance Process Improvement." *Empirical Software Engineering*, 1997, 2(2): 133-142.
- Vriens, C. (2003). Certifying for CMM Level 2 and ISO9001 with XP@Scrum. *Agile Development Conference*, Salt Lake City, Utah.
- West, M. D. (2001). Enculturating Extreme Programmers. *Proceedings of XP Agile Universe*, North Carolina.
- Williams, L., Kessler, R. R., Cunningham, W., et al. (2000). "Strengthening the case for pair programming." *IEEE Software*, 2000, 17(4): 19-25.
- Williams, L. and Kessler, R. R. (2002). *Pair Programming Illuminated*, Addison Wesley.
- Williams, L. (2003). "The XP Programmer: The Few-Minutes Programmer." *IEEE Software*, May/June 2003, 20(3): 16-20.
- Williams, L. and Cockburn, A. (2003). "Agile Software Development: It's about Feedback and Change." *Computer*, June 2003, 36(6): 39-43.
- Wood, W. A. and Kleb, W. L. (2003). "Exploring XP for Scientific Research." *IEEE Software*, May/June 2003, 20(3): 30-36.
- Wright, G. (2003). Achieving ISO 9001 Certification for an XP Company. *Proceedings of XP Agile Universe*, New Orleans.
- Yin, R. K. (1989). *Case Study Research: Design and Methods*. CA., Sage Publications.
- Yourdon, E. (2002). "Light methodologies." *Cutter IT Journal*, 2002, 13.
- Zelkowitz, M. V., Wallace, D. R. and Binkley, D. W. (2003). "Experimental Validation of New Software Technology". *Lecture Notes on Empirical Software Engineering*, World Scientific, 2003, 12: 229-263.

APPENDICES

APPENDIX A – Experience Reports of Agile Methodology Adoption

- Harrison, B. N. (2003). A Study of Extreme Programming in a Large Company, Avaya Research Labs. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Griffin, A. L. A Customer Experience: Implementing XP, Escro.com. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Schuh, P. "Code Redemption - XP as a Means of Project Recovery." <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Lindvall, M., Muthig, D., Dagino, A., et al. (2004). "Agile Software Development in Large Organisations." *Computer*, 2004, 37(12): 58-65.
- Karlstrom, D. Introducing Extreme Programming - An Experience Report. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Murru, O., Deias, R. and Mugheddu, G. (2003). "Assessing XP at a European Internet Company." *IEEE Software*, May/June 2003, 20(3): 37-43.
- Smigelschi, D. "Combining Predictability with Extreme Programming in Embedded Multimedia Project." <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Cohn, M. and Ford, D. (2003). "Introducing an Agile Process to an Organization." *Computer*, June 2003, 36(6): 74-78.
- Srinivasa, P., Neveu, J.-N. and Lynch, F. Extreme Programming in a Customer Services Organization. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Kane, D. Introducing Agile Development into Bioinformatics: An Experience Report. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Pedroso, M., Visoli, M. and Antunes, J. "Extreme Programming by Example." <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Benedicenti, L. and Paranjape, R. Using Extreme Programming for Knowledge Transfer. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Wright, G. Extreme Programming In A Hostile Environment. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Rasmusson, J. (2003). "Introduction XP into Greenfield Projects: Lessons Learned." *IEEE Software*, May/June 2003, 20(3): 21-28.
- Boelsterli, B. S. "Iteration Advocate/Iteration Transition Meeting: Small Sampling of New Agile Techniques Used at a Major Telecommunications Firm." <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Bedoll, R. A tale of two projects: How 'agile' methods succeeded after 'traditional' methods had failed in a critical system-development project.

Washington, The Boeing Company.
<http://www.agilealliance.org/articles/index> [accessed 2004/2005]

- Grenning, J. (2001). "Launching Extreme Programming at a Process-Intensive Company." *IEEE Software*, November/December 2001, 18(6): 27-33.
- Johansen, K., Stauffer, R. and Turner, D. Learning by Doing: Why XP Doesn't Sell. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Ambler, S. (2002). "Lessons in Agility from Internet-Based Development." *IEEE Software*, March/April 2002, 19(2): 66-73.
- Lundh, E. "Promoting change among your peers." <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Bossi, P. and Francesco, C. "Repo Margining System: Applying XP in the Financial Industry." <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Shuch, P. (2001). "Recovery, Redemption, and Extreme Programming." *IEEE Software*, 2001, **18**(6): 34-41.
- Kini, N. and Collins, S. Steering the Car: Lessons Learned from an Outsourced XP Project. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Cohn, M. (2003). The Upside of Downsizing. *Software Testing and Quality Engineering*: 18-21.
- Bossavit, L. (2002). "The Unbearable Lightness of Programming: A Tale of Two Cultures." *Cutter IT Journal*, September 2002, **15**(9): 5-11.
- DeGregorio, G. Use of Agile to Successfully Deploy Major Web-enabled Systems at Dana Commercial Credit and Caterpillar Financial Services Corporation Illustrate Why it's Catching On Fast. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Poole, C. and Huisman, J. W. (2001). "Using Extreme Programming in a Maintenance Environment." *IEEE Software*, 2001, **18**(6): 42-50.
- Elsamadisy, A. "XP On A Large Project – A Developer's View." <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Pine, S. M. An Application of XP in a multiple team/multi-process environment, Advanced Technologies Integration, Inc. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Bryant, R. and Richardson, B. (2001). Can You Teach a Dinosaur New Tricks? Introducing XP in a Large Corporate Development Structure. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Mitchell, S., Auernheimer, B. and Noble, D. Scenarios, Tall Tales, and Stories: Extreme Programming the Oak Grove Way, Oak Grove Systems. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]
- Nies, Z. (2003). Six Degrees - An experience report of adopting XP, Creo, Inc. <http://www.agilealliance.org/articles/index> [accessed 2004/2005]

- Drobka, J., Noftz, D. and Raghu, R. (2004). "Piloting XP on four mission-critical projects." IEEE Software, November/December 2004, **21**(6): 70-75.
- Lippert, M., Becker-Pechau, P., Breitling, H., et al. (2003). "Developing Complex Projects Using XP with Extensions." Computer, June 2003, **36**(6): 67-73.

APPENDIX B – The Twelve Principles of the Agile Manifesto

- 1 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- 2 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- 3 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- 4 Business people and developers must work together daily throughout the project.
- 5 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- 6 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- 7 Working software is the primary measure of progress.
- 8 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- 9 Continuous attention to technical excellence and good design enhances agility.
- 10 Simplicity, the art of maximizing the amount of work not done, is essential.
- 11 The best architectures, requirements, and designs emerge from self-organizing teams.
- 12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Appendix C The Twelve Practices of Extreme Programming

- **The Planning Game:** Business and development cooperate to produce the maximum business value as rapidly as possible. The planning game happens at various scales, but the basic rules are always the same:

1. Business comes up with a list of desired features for the system. Each feature is written out as a **User Story**, which gives the feature a name, and describes in broad strokes what is required. User stories are typically written on 4x6 cards.
2. Development estimates how much effort each story will take, and how much effort the team can produce in a given time interval (the iteration).
3. Business then decides which stories to implement in what order, as well as when and how often to produce a production releases of the system.

- **Small Releases:** Start with the smallest useful feature set. Release early and often, adding a few features each time.

- **System Metaphor:** Each project has an organizing metaphor, which provides an easy to remember naming convention.

- **Simple Design:** Always use the simplest possible design that gets the job done. The requirements will change tomorrow, so only do what's needed to meet today's requirements.

- **Continuous Testing:** Before programmers add a feature, they write a test for it. When the suite runs, the job is done. Tests in XP come in two basic flavours.

1. **Unit Tests** are automated tests written by the developers to test functionality as they write it. Each unit test typically tests only a single class, or a small cluster of classes. Unit tests are typically written using a unit testing framework, such as [JUnit](#).
2. **Acceptance Tests** (also known as **Functional Tests**) are specified by the customer to test that the overall system is functioning as specified. Acceptance tests typically test the entire system, or some large chunk of it. When all the acceptance tests pass for a given user story, that story is considered complete. At the very least, an acceptance test could consist of a script of user interface actions and expected results that a human can run. Ideally acceptance tests should be automated, either using the unit testing framework, or a separate acceptance testing framework.

- **Refactoring:** Refactor out any duplicate code generated in a coding session. You can do this with confidence that you didn't break anything because you have the tests.

- **Pair Programming:** All production code is written by two programmers sitting at one machine. Essentially, all code is reviewed as it is written.

- **Collective Code Ownership:** No single person "owns" a module. Any developer is expect to be able to work on any part of the codebase at any time.

- **Continuous Integration:** All changes are integrated into the codebase at least daily. The tests have to run 100% both before and after integration.

- **40-Hour Work Week:** Programmers go home on time. In crunch mode, up to one week of overtime is allowed. But multiple consecutive weeks of overtime are treated as a sign that something is very wrong with the process.

- **On-site Customer:** Development team has continuous access to a real live customer, that is, someone who will actually be using the system. For commercial software with lots of customers, a customer proxy (usually the product manager) is used instead.
- **Coding Standards:** Everyone codes to the same standards. Ideally, you shouldn't be able to tell by looking at it who on the team has touched a specific piece of code.

(Brewer, 2001)

Appendix D The type of evidence produced by empirical software engineers

Judith Segal

Department of Computing
Faculty of Maths and
Computing
The Open University
Walton Hall
Milton Keynes
MK7 6AA
UK
+44 (0)1908 659793
j.a.segal@open.ac.uk

Antony Grinyer

CSW Group Ltd
4240 Nash Court
Oxford Business Park South
Oxford
OX4 2RU
ENGLAND
+44 (0) 1865 337400
antony.grinyer@csw.co.uk

Helen Sharp

Department of Computing
Faculty of Maths and
Computing
The Open University
Walton Hall
Milton Keynes
MK7 6AA
UK
+44 (0)1908 653638
h.c.sharp@open.ac.uk

ABSTRACT

This paper reports on the research published between the years 1997 and 2003 inclusive in the journal of Empirical Software Engineering, drawing on the taxonomy developed by Glass et al. in [2]. We found that the research was somewhat narrow in topic with about half the papers focusing on measurement/metrics, review and inspection; that researchers were almost as interested in formulating as in evaluating; that hypothesis testing and laboratory experiments dominated evaluations; that research was not very likely to focus on people and extremely unlikely to refer to other disciplines. We discuss our findings in the context of making empirical software engineering more relevant to practitioners.

General Terms

Experimentation

Keywords

Empirical software engineering; research taxonomy; evidence; field studies.

INTRODUCTION

In [1], we argued that the gap between empirical software engineering and software engineering practice might be lessened if more recognition were afforded to the following two points:

- Evidence from case or field studies of actual software engineering practice is essential in order to understand and inform that practice.
- The nature of evidence should fit the purpose to which the evidence is going to be put. For example, quantitative evidence might be necessary to convince a manager to introduce some change in working practices; a rich case study might persuade developers to accept such a change.

In this paper, we investigate the nature of the evidence published over a period of 7 years in the academic journal, Empirical Software Engineering (see <http://journals.kluweronline.com/>). Our investigation was inspired in part by questions arising from the argument above – what is the prevalence of case and field studies of software engineering practice? Is there a wide variety in the types of evidence reported in the field of empirical software engineering? – and in part by the work of Glass, Vessey and Ramesh, as reported in [2]. These latter sought to describe the current state of software engineering research by scrutinizing 369 papers representing a sample of those papers published in 6 top software engineering journals over a period of 5 years. The papers were classified along the following dimensions:

- The topic covered (for example, algorithms, data structures; organisational issues);
- The research approach. This was divided into the following categories: descriptive; formulative (for example, of guidelines, methods, algorithms,

models, etcetera) and evaluative. This latter included the subcategories of positivist and interpretivist. Positivist research assumes the existence of an objective measurable reality and the independence of the researcher and object of research. It frequently takes the form of hypothesis testing, referred to as 'Evaluative deductive' in this paper. Interpretivist research, on the other hand, argues that our understanding of reality depends on how we interpret our perceptions in the light of our experience: that is, the object of research and the researcher are not separate. A field study in which our understanding of reality emerges through social interactions is a typical interpretivist study.

- The research methods. These included positivist methods such as laboratory experiments and interpretivist methods such as field studies.
- The reference discipline. Just as civil engineering (say) is an applied science informed by the pure sciences (reference disciplines) of mathematics, geology, chemistry, physics etcetera, so software engineering is an applied science potentially informed by pure sciences such as mathematics, psychology, sociology, etcetera.
- Units of analysis. These included categories such as the profession; the group; individuals; computer system, computer element such as program.

Glass et al. found that

- The spread of topics was broad (though a closer look at their results shows that less than 3% of the papers were on organisational and societal topics. It appears that the term 'broad' refers only to technical topics).
- As to research approach, over half the papers were formulative; a further 28% were descriptive and only 14% evaluative (4% evaluative deductive; less than 1% interpretative).
- Research methods were dominated by conceptual analysis, proof-of-concept and mathematical analysis, which together accounted for nearly three quarters of all the papers.
- In 98% of the papers surveyed, reference disciplines were not mentioned.
- Most of the units of analysis were on abstract concepts: only 11% of the papers surveyed focused on people.

Glass et al's study did not include the journal *Empirical Software Engineering*, which we assumed had published a good portion of empirical software engineering research since its inception in the middle 1990s. We thus determined to carry out a classification on papers published by this journal, similar to the one carried out by Glass et al.

Given our argument in [1] and described above, we hoped to find many field studies of software engineering practice, and a variety of different types of evidence to fit the variety of purposes to which such evidence could be put. As we shall see, our hopes were not realized.

METHODOLOGY

The material

We classified all those 119 papers which appeared in *Empirical Software Engineering* between 1997 and 2003 inclusive, excluding only those that were pure polemic. Our classification scheme was based on that of Glass et al. with some amendments to fit our own purposes. These amendments are as follows:

- In 'units of analysis', where people were involved, we determined whether they were practitioners or students; what the nature of their activity was, and whether they were interacting with real-world or artificial systems. It was important for our purposes to differentiate between papers where the focus was on students and 'toy' systems, and those where the focus was

on practitioners developing, testing or maintaining systems in the real world.

- We included another category to record whether the authors of a particular paper were academics, practitioners or both. We based our decision on the email addresses provided. We shall comment in section 3 on the inherent ambiguity in this.

The method

Two coders classified all the 1997 papers and then came together to discuss their individual classifications, thus coming to a shared understanding of the classification scheme. Papers from the years 1998 – 2003 were then coded independently, though the two coders continued to discuss how the coding scheme might be interpreted and extended as issues arose. For each paper, we hoped to gain sufficient information to complete the coding by reading its abstract and conclusions; if this did not suffice, we speed read the paper to determine (for example) whether there was mention of a hypothesis or of statistical testing (both indicative of an evaluative deductive research method), or whether the participants (if any) were students or practitioners. Only if the required information was still not forthcoming did we carefully peruse the paper.

We accept that the resulting coding cannot be completely objective, not least because of the inherent ambiguity of the classification scheme. For example, if a paper describes the use of some metric in order to distinguish between two testing schemes, it is difficult to know if the authors' intention was to focus on the metric or on testing. Nevertheless, the two independently coded sets of papers showed a high measure of agreement, averaging 83% across the classification categories. The papers were then handed to a third coder (who had not been party to the earlier discussions), who recoded those papers on which the first two coders disagreed, according to the agreed classification scheme.

RESULTS

Here, we report our findings.

Research topic

5 topics (at the level of granularity provided in [2]) covered over three quarters of the papers, as shown below.

Table 1. Topic

Software life-cycle/engineering (incl. requirements, design, coding, testing, maintenance)	33%
Measurement/metrics (development and use)	19%
Process management	10%
Tools (incl. compilers, debuggers)	8%
Computing Research (that is, meta-level issues, such as discussions about methodologies)	8%

No other topic was covered in more than 4% of the papers. Analysing the most common topic (software life-cycle/engineering) in more detail, almost half of the papers covering this topic were concerned with maintenance and a further third with review and inspection. Perhaps this bias is a reflection of the workshops held and journal special issues published during the period under consideration.

Research approach

The research approaches seen in the papers we scrutinised are recorded in Table 2 below.

Table 2. Research Approach

Evaluative deductive	46%
Descriptive	13%
Formulative (process, method, algorithm)	10%
Formulative (model)	8%
Formulative (guidelines/standards)	5%
Evaluative – other	5%
Review of literature	5%
Evaluative – interpretive	2%

No other research approach was seen in more than a single paper. Given that we are considering empirical software engineering, we were surprised that only just over a half of the papers featured evaluation (though mindful that this was about four times the number noted in the general software engineering literature by Glass et al.). Of these, evaluative deductive – testing hypotheses in a very positivist tradition – dominated.

Research method

Table 3 records the research methods used.

Table 3. Research Method

Laboratory experiment (human subjects)	29%
Data analysis	15%
Case study	13%
Descriptive/exploratory survey	12%
Laboratory experiment (software)	7%
Concept implementation (proof of concept)	7%
Meta-analysis	6%
Literature review/analysis	6%

No other research method was seen in more than 2 of the papers. The dominance of laboratory experiments (in 36% of the papers) reflects the dominance of the evaluative deductive, hypothesis testing, research approach.

Reference discipline

The next table indicates that empirical software engineers, like software engineers as noted in [2], tend to be insular and take little cognisance of research in other disciplines.

Table 4. Reference Discipline

None	85%
Psychology	6%
Statistics/mathematics/computational science	5%
Social Science	4%

Units of analysis

In table 5, we consider units of analysis, that is, the entities which were the focus of the papers.

Table 5. Units of Analysis

Real-life computer system/data/project	36%
Profession	24%
Individual students	18%
Individual practitioners	7%
Teams of students	4%
Artificial laboratory computer system/data/project	3%
Team of practitioners	2%
Other	6%

Here, we see that only 31% of the papers had people at their focus (and given the results of Table 3, presumably the vast majority of these concerned laboratory experiments, that is, people in an artificial setting). On the other hand, over a third of the papers focussed on real-world systems.

Who wrote the papers?

Finally, we consider authorship of papers in Table 6.

Table 6. Paper author(s)

Academic authors alone	73%
Mixture of academic and practitioner authors	16%
Practitioners alone	11%

We noted in 2.1, that we determined whether an author was an academic or practitioner simply on the basis of his/her email address. We recognise that this does not take account of practitioners being seconded to academic institutions and vice versa, nor the fact that the distinction between an academic department and a research department in an industrial setting may not be clear-cut. Nonetheless, academic authors clearly predominate.

DISCUSSION

Our results reveal that, based on an analysis of 119 papers, comprising nearly all the output of the journal Empirical Software Engineering between the years 1997 and 2003 inclusive, empirical software engineers are

1. Somewhat narrow in topic, with measurement/metrics, maintenance, and review and inspection accounting for about half the papers;
2. Almost as interested in formulating (processes; models; guidelines), describing and reviewing, as in evaluation;
3. Far more likely to evaluate using hypothesis testing than any other method;
4. Likely to do laboratory experiments (described in over a third of the papers);
5. Very unlikely to refer to any other scientific discipline;
6. Not very likely to focus on people.

This last point is, we think, unfortunate, in that software engineers are in general agreement as to the importance of people factors in the successful practice of software engineering ([3], [4], [5]). Given this importance, we feel that software engineers might take more cognizance of reference disciplines in the social sciences such as psychology and sociology (cf. point 5 above).

With respect to point 4 above: cognizant of the fact that software engineering takes place within a context – a particular team of people are involved; a particular application environment; a particular development environment; a particular organisational, social, market environment – we would encourage empirical software engineers to consider using research methods which take account of the complexity of context, such as field studies, rather than methods which factor out the effect of context, such as laboratory studies. Such studies can make use of natural controls to confirm/disconfirm a hypothesis ([6]), cf. point 3.

An argument often made against field studies is that they cannot be replicated – but neither can a software engineering activity in the real world (one cannot dip one's toes into the same river twice!). Validation of such studies can be based not on replication of the study but on replication of the interpretation: the question to ask is, would other researchers from the same scientific cultural tradition as the original researcher(s) and given the same data, come to the same conclusions?

We recognize the practical difficulties of involving practitioners in research and performing field studies ([7]). It is clearly easier to involve students in a laboratory experiment, and as Tichy says in [8], graduate students in computer science can be more technically adept and up to date than practitioners. However, graduate students are *not* practitioners: they do not work in the same organisational and professional context; they are not subject to the same pressures. It is plausible that there are circumstances where laboratory experiments with students might yield results which can inform practice (for example, experiments concerned with individual cognition, for example, one designed to test whether *this* representation is easier to comprehend (in some sense) than *that*). Nevertheless, we urge researchers to scrutinize the external validity of their laboratory experiments – do the results of their research really have the potential to inform the richly contextualized practice of software engineering?

In the above, we have argued for the importance of field studies in empirical software engineering within the positivist tradition of hypothesis deduction and testing. In [1], we argued for their importance in constructing within the interpretivist tradition an understanding of the actual practice of software engineering. We argued that such an understanding on the part of empirical software engineers may have an important role to play in bridging the gap between them and practitioners.

Although we are disappointed by the lack of field studies and especially interpretive field studies, we are pleased by two facets of current empirical software engineering research. Firstly, we applaud the efforts of many of the researchers surveyed to consider real-life systems, data and projects (see Table 5). Secondly, our analysis presents evidence that empirical software engineers are 'reflective practitioners'; they reflect upon their discipline and how it might be improved (witness the fact that a quarter of the papers surveyed focused on the profession, as recorded in Table 5). It remains to be seen whether this reflection will influence empirical software engineering practice.

REFERENCES

1. Segal, J. The nature of evidence in empirical software engineering. *Proc. Intl. Workshop on Software Technology and Engineering Practice (STEP 2003)*, IEEE Computer Society Press, 40-47, 2003.
2. Glass, R.L., Vessey, I., Ramesh, V. Research in software engineering: an analysis of the literature. *Information and Software Technology*, 44, 491-506, 2002.
3. Seaman, C. Methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25(4), 557-572, 1999.

4. Boehm, B., Turner, R.. *Balancing Agility and Discipline*. Addison-Wesley, 2004.
5. Cockburn, A. *Agile Software Development*. Addison Wesley, 2002.
6. Lee, A.S. A scientific methodology for MIS case studies. *MISQ*, 33-50, 1989.
7. Sim, S.E., Singer, J. and Storey, M-A. Beg, Borrow or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research. *Empirical Software Engineering*, 6, 85-93, 2001.
8. Tichy, W. Hints for reviewing empirical work in software engineering. *Empirical Software Engineering*, 5, 309-312, 2000

Appendix E Why do organisations adopt agile methods? An initial study

Antony Grinyer*

Judith Segal**

Helen Sharp**

*CSW Group Ltd, 4240 Nash Court, Oxford Business Park South, Oxford, UK, OX4 2RU
antony.grinyer@csw.co.uk

**Department of Computing, Faculty of Maths and Computing, The Open University,
Walton Hall, Milton Keynes, MK7 6AA, UK
<j.a.segal; h.c.sharp@open.ac.uk>

Abstract. In this paper, we examine experience reports and responses to a question posed to three popular online agile mailing lists, with the aim of identifying the factors that persuade organisations to adopt agile approaches to software development. Our findings reveal that most organisations adopt agile approaches to software development as a response to recurring project failure; that organisations can be greatly influenced by internal and external parties to adopt agile approaches; that some organisations adopt agile approaches believing that software time-to-market will be reduced, and that a small number of organisations implement agile approaches in response to their changing business requirements or organisational downsizing. In addition, our data reveal many unanswered questions. We discuss our findings in the light of a general model of diffusion of innovations, described by Rogers [1], and in the wider context of empirical software engineering.

1. INTRODUCTION

The escalating pace of the computer industry and the ever-growing demands from customers to deliver software faster, and to the highest quality, has placed software practitioners under increasing pressure to seek ways of rapidly addressing consumer expectations. Prior to the turn of the century, long-standing approaches to software development ([2], [3], [4]) provided the models and process heuristics for developing software in industry, but since the late 1990s, these traditional approaches have been challenged by agile methods. But why do practitioners choose to adopt agile approaches, especially given the dearth of any hard empirical evidence as to its efficacy in specific contexts? [5]

The rapid uptake of agile methods seems somewhat akin to the spread of Software Process Improvement (SPI) through the software engineering industry in the 1990s, in that the latter was rapid, attracted a great deal of attention and took place within a context in which little hard empirical evidence had been amassed, published, aggregated or analyzed. In the absence of this evidence, organisations adopted an SPI programme for a variety of reasons, for example, because their customer mandated it or because they wanted to 'jump on the bandwagon' (see, for example, [6]). Informal influences through peer networks and colleagues appear to have been more influential than any other persuasion mechanisms (see, for example, [7] and [8]).

This paper explores the following questions: Are factors similar to these influencing organisations to adopt agile methods? What other factors affect adoption? We report on an initial study investigating the factors which persuade software engineers to adopt agile methods. Our methodology for doing this was based on experience reports - mostly from literature but also some email communications. Evidence from such an analysis has limitations as to what it does, and does not, reveal, as we shall discuss in Section 4 below. However, it does provide a starting point on which future studies can build.

In what follows, we shall firstly discuss our methodology and then present our results. We discuss how our findings fit with a general model of diffusion of

innovations, and finally, consider the role of empirical evidence in informing practitioners' decisions.

2. METHODOLOGY

2.1 Data sources

Our study was based upon literature analysis and an analysis of responses to an informal question posed to three popular agile related mailing lists. To identify the literature for our study we investigated sources of publications that might provide experience reports relating to adoption of agile approaches. Of these sources, we found the Agile Alliance Article Library [9] to contain a vast collection of approximately 1165 articles relating to agile development practices. These articles ranged from informal experience reports to papers published in well-known peer reviewed journals, such as IEEE Software. This library formed the literature base for our investigation.

It would be an extremely time consuming process to read and scrutinize every article of the one thousand-plus online library papers in the time available. We therefore selected articles from those categories of the Article Library which we believed might contain experience reports of adopting agile methods, for example, those from the category *Introducing Agile Processes*. Within each such category, we selected articles on the basis of their titles and then read each abstract to determine the relevance of the article to our investigation. We examined approximately 335 paper titles from 7 main categories of the Article Library. Of these 335 paper titles, we found 34 articles with a particular focus on experiences with adoption of agile methods. These 34 articles included publications from academic journals and less formal experience reports from software development practitioners.

In order to obtain further experience reports with respect to the factors that influence organisations to implement agile approaches, we posted a simple question to three popular agile methods online discussion mailing lists - the DSDM consortium mailing list; the Extreme Programming mailing list, and the Selling Agile mailing list. We asked '*What primary factor(s) persuaded your organisation to adopt an agile software development approach?*'. These three mailing lists were selected on the prevalence and frequency of agile related online discussions posted on a daily basis.

From the mailing lists, we received 20 responses to the question we posed; however, only 7 responses across all three target mailing lists provided personal experiences and factors governing the adoption of agile approaches. The remaining 13 mailing list respondents provided speculative ideas and personal perceptions concerning the factors influencing organisations to adopt agile approaches. Because these 13 responses were purely speculative, we have not included them in our results.

Once the articles and mailing list responses were collected our efforts turned to identifying the factors persuading organisations to adopt agile approaches.

2.2 Adoption Factor Taxonomy

We closely scrutinized each of the articles and mailing list responses collected and identified the adoption factors cited. These adoption factors were used to build up a classification system we called the 'Adoption Factor Taxonomy'. The decision to develop the taxonomy using a bottom-up approach, that is, creating the classifications as the literature was examined, was taken so as to lessen the biasing effect of any a priori assumptions or conjectures regarding factors that might persuade organisations to adopt agile approaches. As each article and mailing response was classified and more adoption factors were identified, so the taxonomy developed.

3. RESULTS

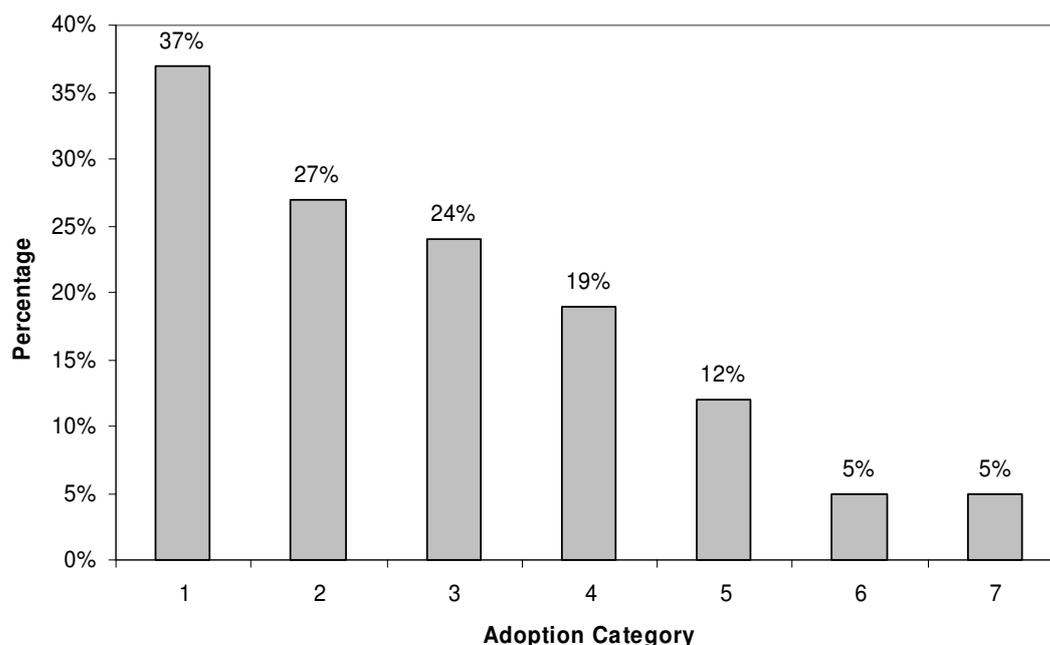
From the combined total of 41 experience reports drawn from 34 articles from the Agile Alliance Article Library and 7 mailing list responses, we found 53 adoption factors. These 53 factors fitted into 7 categories, as in Table 1.

Table 1. Adoption Factor Taxonomy: Number of factors in each category.

Adoption Category	Number of Factors
1 Software development process improvement in response to recurring project failure.	15
2 Internal influences in organisation – group/individual (e.g. management, lead developer, or system architect).	11
3 Competitive edge (e.g. time-to-market).	10
4 External influences in organisation – outside coaching/training/consulting.	8
5 Response to changing business requirements.	5
6 Following industry leaders, industry gurus, technology trends and fashions.	2
7 Downsizing/scaling down development team and process.	2
Total number of factors	53

In Figure 1, we show the percentages of the 41 reports citing factors in each of the categories. These percentages add up to more than 100% as some papers cited factors in more than one category.

Figure 1. The percentage of reports citing factors in each category.



Software development process improvement in response to recurring project failure was cited as a factor influencing the decision to adopt agile approaches in over a third of the reports.

About a quarter of the reports cited the influence of internal individuals or groups such as management, lead developers, and system architects. Again, about a quarter of the reports stated that the presumed gain in competitive edge influenced the decision. These organisations were motivated to move to agile approaches on the premise that developing software over short iterations would improve productivity, decrease time to market, provide greater business value, and thus provide the edge over competitors. In about a fifth of reports, the adoption of agile approaches was said to be influenced by external parties such as coaches, trainers, and consultants who were employed by organisations to provide expertise and coaching during software development projects.

Other adoption factors were identified in just a small number of experience reports. These included: the adoption of agile approaches in response to changing business requirements, following other leading organisations in industry, and adopting agile approaches when downsizing their development team and software development process.

Our data reveal three more findings that we feel are of interest though not immediately related to adoption factors. The first is that none of the experience reports reported adopting an agile method as described in a textbook: all reported customising or adapting the method to fit the local context. The second is that only four of the 41 reports recorded that they attempted pilots or trials of agile approaches before fully adopting the selected agile practices. Thus, our findings suggest that organisations adopt agile approaches with little attempt to perform risk analysis of the impact of these approaches within their organisation. The third is that reports did not provide any reasons why adopters believed that adopting an agile method would overcome the problems they had encountered, or improve the existing situation. For example, where agile development was adopted in response to repeated project failure, no justification was presented to indicate why the adopters believed that agile methods might avoid project failure in the future; if it was claimed that an agile method was adopted in response to changing business requirements, no reasons were given as to why the adopter believed that an agile approach was now suitable.

4. LIMITATIONS OF OUR STUDY

We consider two types of limitations here: the limitations of our methodology and the limitations of our findings.

4.1 Methodological limitations

Firstly, we relied on our judgment to select appropriate articles from the Agile Alliance Article Library based on their category, title and abstract. It is thus likely that several agile adoption experience reports were overlooked.

Secondly, our decision to send the question '*What primary factor(s) persuaded your organisation to adopt an agile software development approach?*' to three popular agile related mailing lists, limited responses to self-selected participants, that is, we did not identify a target sample of participants, and participants who responded did so by choice. We do not know, therefore, how representative the participants are of agile practitioners.

4.2 The limitations of our findings

Our decision to pursue a research methodology based purely upon written experience reports constrained our findings to the evidence we found in the literature we examined. Many questions arose which could not be answered using the available evidence.

For example, we found there was a strong link between the adoption of agile methods in organisations, and the influences of internal and external parties (people) – 46% of reports cited this as an adoption factor. However, we found little evidence in our study to suggest reasons why these influential people chose agile approaches in the first place. This raises the questions - to what extent do organisations risk moving to new agile approaches to software development based upon the opinions of internal and external parties? Why do these influential parties advocate and believe agile approaches will work? On what information sources or evidence do they base their judgments? In reports that described an adoption factor as being an external influence (such as an external consultant employed by an organisation), it was not clear whether the consultant, or the person(s) who was responsible for employing the consultant in the organisation, initiated the adoption of agile approaches. That is, did the organisation already know they wanted to adopt an agile approach and hence employed a knowledgeable consultant to help their transition? Or did the consultant persuade the organisation that an agile approach was what was needed?

Questions remain as to the deep reasons why an organisation chose a particular methodology. Most papers simply stated that methodology X was adopted due to its 'popularity'. We were left wondering whether such an adoption was simply an example of following a trend, or whether the real influencing factor was that the popularity of X meant that there was widespread knowledge and support available to implement X.

The reports did not provide any reasons why adopters believed that adopting an agile method would overcome the problems they had encountered, or improve the existing situation. We did not follow up this question and so we have no insight into why they thought this was the case.

We are cognizant that investigations of agile adoption factors through more rigorous and diverse methodology may provide richer data and more informed findings, therefore one of our future aims is to conduct such studies, using a mixture of qualitative and quantitative techniques, as discussed in [10].

5. DISCUSSION

In this section, we discuss our findings and the questions they raise in the light of the general model of the diffusion of innovations as reported in Rogers [1], and the role of empirical evidence in the practice of software engineering.

5.1 A general model of the diffusion of innovations

In [1], Rogers describes a model of the innovation process in organisations. This model is constructed from data from a large number of studies of diffusion of different types of innovation. It consists of five stages:

- Agenda setting, where needs and problems are identified and prioritized, and the environment is searched for innovations which might meet these needs and problems;
- Matching, where the fit between the problem and the innovation is investigated. This might involve a consideration of risks and benefits and a feasibility test;
- Redefining/restructuring, where 'the innovation is re-invented so as to accommodate the organization's needs and structure more closely, and when the organization's structure is modified to fit with the innovation. Both the innovation and the organization are expected to change, at least to some degree' [1:424];
- Clarifying, where the innovation is put into more widespread use in an organisation and its importance becomes clearer to the members of the organisation, and

- Routinizing, where the innovation becomes totally integrated within the organisation.

We now discuss how our findings fit this model. Since we are only concerned with *adoption* of an innovation (agile approaches to software development), we are only interested in the first three stages (agenda setting, matching and redefining/restructuring).

In discussing agenda setting, Rogers says of the studies he scrutinised:

'.. in most cases, a shock to the organization reached a threshold of attention and led to action by the organisation's participants' [1:422]

This may have been the case in those organisations (over a third of our sample) which adopted agile methods in response to recurring project failure.

As to 'matching', we have already commented that trialling was reported in less than 10% of the experience reports. We should note, however, that Rogers comments that the experiences of one's near-peers with the innovation might be equivalent to trialling an innovation personally.

With respect to redefining/restructuring, we reported in section 3 above that all the experience reports scrutinised had customised agile methods to fit their particular context. This is consistent both with the received wisdom in the software engineering literature that software methodologies are rarely adopted 'out of the box', but rather that they are tailored to fit the project in hand ([11], [12]) and with the results reported by El Eman [13] where, in his study of 21 small projects utilizing agile methods, he found none adopted agile methods in pure form.

5.2 The role of empirical evidence in the practice of software engineering

Empirical software engineers often voice disquiet that practitioners' decisions on adopting new tools/techniques/approaches to support software development, appear to be based on purely capricious reasons rather than on hard empirical evidence that *this* tool/technique/approach is most efficacious in *these* circumstances. ([14], [15], [16]) As we mentioned in our introduction, such hard empirical evidence concerning agile approaches is difficult to find. There are several experience reports available, as noted in Section 3 above, but it is difficult to aggregate experience reports into a coherent body of evidence, given the context-rich nature of such reports and their inherent subjectivity. Some work has been reported on developing a framework to provide the basis for such aggregation (see, for example, [17]), though the issue of how to aggregate qualitative data has not been adequately addressed.

One question which remains is: if there were hard empirical evidence as to the efficacy of agile methods in a particular context, would consideration of such evidence play a significant part in influencing practitioners' decision whether or not to adopt?

Some people in the literature think the answer to this question is 'no'. Rogers [1] deduced from his survey of individuals making adoption decisions that

'Most individuals evaluate an innovation [in deciding whether or not to adopt] not on the basis of scientific research by experts but through the subjective evaluations of near peers who have adopted the innovation' [1:36]

This quote begs the following question: what persuades early adopters to adopt an innovation, since for such adopters there is no existing community of peers who have already tried to adopt?

While this quote is a statement about individuals and not organisations, it does align with the findings of [8] considering the adoption of SPI:

'The evidence that practitioners value appears to be very different to the evidence that researchers value. In particular, practitioners seem to value local expertise in contrast to independent empirical evidence...' [8:3]

And with the statement of Zelkowitz [18] that

'...the industrial community is generally wary of laboratory research results...'. [18:255]

An alternative point of view (see [19]) is that the scientific evidence emanating from the empirical software engineering community does not adequately address practice. From this point of view, the questions which need addressing include: Is the empirical evidence emanating from the academic community addressing issues pertinent to practitioners? Is the evidence of the right type to influence practitioners? Is the evidence presented in a form which appeals to practitioners? Evidence of the factors which influence practitioners' decision making processes, can help the empirical software engineer address such questions. For example, given the influence of the reported experience of near peers, it is plausible that context rich case studies (reported by an outsider and having more objectivity than insider experience reports) might be more appealing to practitioners than controlled laboratory experiments.

6. CONCLUSIONS

The aim of this investigation was to begin to identify the factors influencing organisations to adopt agile approaches. Our findings reveal that responding to recurrent project failure, and the influence of internal and external individuals, are the major factors in persuading organisations to adopt agile methodologies. Given the gaps we have identified in the experience reports and the known limitations and constraints of this study, our findings have strongly spurred our interests to pursue further investigations into the adoption of agile approaches in organisations. Using data triangulation [10] to draw upon multiple research methods for data collection, our future research plans include investigations using survey questionnaires and interviews, together with more longitudinal studies involving observations of organisations, in order to further investigate the questions we have raised in this paper.

We hope that the results of these further studies will serve to articulate practitioners' decision-making processes, will enhance a general model of diffusion of innovations, and will inform empirical software engineers as to the type of evidence which might best appeal to practitioners.

References

- [1] Rogers E.M., (2003), Diffusion of Innovations, Fifth Edition, Free Press.
- [2] Royce, W. W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. Proceedings of WESCON, Los Angeles.
- [3] DeMarco, T. (1979). Structured Analysis and System Specification. New York, Yourdon Press.
- [4] Boehm, B. (1988). "A spiral model of software development and enhancement." Computer, May 1988, **21**(6): 61-72.
- [5] Lindvall, M., Basili, V. R., Boehm, B., et al. (2002). Empirical Findings in Agile Methods. Proceedings of XP Agile Universe, Chicago.

- [6] Segal J., 2001, 'Organisational Learning and Software Process Improvement: A Case Study', in *Advances in Learning Software Organisations*, K-D Althoff, R.L. Feldmann, W. Muller (Eds.), Lecture Notes in Computer Science, Vol. 2176, Springer, 68-82.
- [7] Sharp, H., Hovenden, F. and Woodman, M. (2003) Tensions in the Adoption and Evolution of Software Quality Management Systems, in Proceedings of PPIG/EASE 2003, pp297-312, Keele University.
- [8] Rainer, A., Jagielska, D. and Hall, T. (2005). Software engineering practice versus evidence-based software engineering research. Proceedings of the REBSE workshop, associated with ICSE 2005, St. Louis, Missouri, USA.
- [9] Agile Alliance Article Library. (2001). <http://www.agilealliance.org/articles> (accessed 06/02/2005)
- [10] Robson, C. (2002). Real World Research. Oxford, Blackwell Publishing Ltd.
- [11] Boehm and Turner, 2004, *Balancing agility and discipline: a guide for the perplexed*, Addison-Wesley
- [12] Glass, R. 2002. Searching for the Holy Grail of Software Engineering. *Comm ACM* 45(5): 15-16
- [13] El Eman, K. (2003). "Agile Project Management." *Cutter Consortium Agile Software Development and Project Management Advisory Service*, 2003, **4**(11).
- [14] Shepperd M., "Empirically-based Software Engineering", *UPGRADE*, IV (4), Aug. 2003, 37-41.
- [15] Pfleeger, S.L., 2005, 'Soup or Art? The role of evidential force in empirical software engineering' *IEEE Software*, Jan-Feb, 2005, pp 66-73
- [16] Budgen, D. and Kitchenham, B., 2005, 'Realising evidence-based software engineering: a report from the workshop held at ICSE 2005', Proceedings of ICSE-2005, <http://portal.acm.org/dl.cfm>
- [17] Williams, L., Layman, I., Abrahamsson P, 2005, 'On establishing the essential components of a technology dependent framework for industrial case-study based research', Proceedings of the Workshop on Realising Evidence-Based Software Engineering, ICSE-2005, <http://portal.acm.org/dl.cfm>
- [18] M.V.Zelkowitz, D.R. Wallace, D.W. Binkley, 'Experimental validation of new software technology', in *Lecture Notes on Empirical Software Engineering*, N.Juristo, A.M. Moreno (eds.), World Scientific Publishing Co., 2002, pp. 229-263.
- [19] Segal J. 2004, 'The nature of evidence in empirical software engineering', Proceedings of the International Workshop on Software Technology and Engineering Practice (STEP) 2003, IEEE Computer Society Press: 40-47.