Strategic Factors in Agile Software Development Method Adaptation: A Study of Market-Driven Organisations

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

DOCTOR OF PHILOSOPY
in Information Technology
at Massey University, Albany campus, New Zealand

Ramesh Lal
2011

Strategic Factors in Agile Software Development Method Adaptation: A Study of Market-Driven Organisations

Abstract

Agile methods now provide an alternative to the structured approach for software development. Since their early adoption in the mid-1990s, there has been a growing acceptance of agile methods as a legitimate development approach amongst the information systems development community. Amongst some of the talked about agile benefits in comparison to the structured approach are improvements in delivery and quality standards, productivity levels and knowledge transfer within the organisation, and customer satisfaction through collaboration and frequent delivery of implemented features. The agile philosophy of a team approach and ownership for product development creates a feel-good and motivational factor for engineers. Regardless, the inventors of the agile approach strongly emphasize adaptation to achieve continuous product development success. As a result, an understanding of the adoption of agile methods cannot be separated from an understanding of its adaptation by development teams.

Using a theoretical framework relating to software development process adaptation, developed by Fitzgerald (1998), and through two case studies (agile adopters) qualitative data in relation to agile adaptation was gathered and analysed to determine the nature of agile adaptation. This revealed that adaptation is a critical feature of agile approaches. These adaptations enhance the strategic nature of the organisational factors and the chosen agile approach to develop market differential products. The main contribution of the thesis is that it presents an adapted agile adaptation framework showing that at a conceptual level the main structures (the organisational and methodology factors) that influence software development remain consistent when moving to agile development. However, the underlying elements of these factors are comprehensively adapted, transforming an organisation into an agile organisation in a market-driven environment. Further, in this environment the organisational and development structures will keep evolving due to the emerging market factors and improvements in product development technologies.

Acknowledgment

This thesis would not have been possible without the help of many wonderful people who have in one way or another helped me for my Ph D study. Firstly, I would like to thank my main supervisor, Dr David Parsons, for all the guidance and support throughout this challenging journey. I am truly grateful for your commitment, especially your assistance in facilitating the case study organisations and for your instant feedback on the successive drafts of Chapters. I would also like to thank my second supervisor, Dr Rosemary Stockdale who has given tremendous input on my thesis despite moving overseas. I sincerely thank both of you for the time and help given to me.

Next, I would like to thank the two organisations featured in this study for their kindness in giving me their precious time. I truly would like to thank all the individuals whom I had interviewed to collect data. To protect their anonymity, the company and individual details cannot be disclosed.

Thanks to Paul Hughes and Manfred Lange for all the cooperation and support given to me, without you two this thesis would not have been possible. My appreciation is extended to Sam Alexandra, Lornie Enggong, and Guy Kloss for all the help and advice.

I would also like to acknowledge Massey University for the doctoral scholarship and for the part-time employment, the Agile Alliance for providing me with funding, and Software Education Associates Limited for allowing me to attend the Agile 2005 Conference.

Finally, my deep appreciation goes to my wife and daughter for their patience and never ending support. To my wife, Nancy, without your belief in me and encouragement this journey would not have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter one: Introduction

1.1 Agile methods for software development

Agile methods are widely being adopted because of expectations that these methods can bring development success (Esfahani, Yu, & Annosi, 2010). One of the main reasons for success with agile methods is that they are highly adaptive (Boehm & Turner, 2003). There is considerable research and publications on agile adoption and practices but work on agile adaptation issues is sparse, particularly research into agile adaptation factors (Abrahamsson, Warsta, Siponen, & Ronkainen, 2003; Fitzgerald, Russo, & O'Kane, 2003). This research investigates how agile adaptation factors drive adaptation of agile practices to maintain or enhance a strategic product development environment in rapidly shifting business conditions.

1.1.1 The evolution of software development

Software is an indispensable tool for businesses, which make significant investment to acquire new software products (Charette, 2005). Since the inception of software for business data processing in the early 1960s, the software development community has continuously experienced numerous software development project failures, costing substantial amounts of money (Keil & Robey, 2001; Mahaney & Lederer, 1999; Wallace & Kell, 2004).

In the 1960s, failures in software development were mostly attributed to using an ad hoc approach (Brooks, 1975). The introduction of the more formal System Development Life Cycle (SDLC) process in 1970 established the essential prescribed structure for software development (Fitzgerald, 1997). A decade later, the engineering approach focused on achieving a testable and repeatable development process for projects to achieve successful product developments (Wang & King, 2000). However, the software engineering approaches did not alleviate the development problems faced by the software development community (Satzinger, Jackson, & Burd, 2005).

Against this background, the mid-1990s saw the emergence of the agile approach that claims to overcome development problems. The agile approach emphasises development in short development cycles, micro-planning, cross-functional collaboration, face-to-face interactions, and frequent retrospectives (Beck & Boeham, 2003; Highsmith & Cockburn, 2001).

1.1.2 Agile practices

The agile practices are claimed to lead to quality gains (Coram & Bohner, 2005), improvement in project visibility (Smits, 2007), improvement in productivity, reduction in development costs, and reduction in time-to-market (Reifer, 2002). In addition, an agile approach has potential to make product development responsive to customer and market needs through product customisation while delivering general purpose products (Breu, Hemingway, Strathern, & Bridger, 2001).

Agile product development practices also introduce changes in team culture in an attempt to bringing reciprocal effects of loyalty and commitment to the team and projects (Sherehiy, Karwowski, & Layer, 2007). Agile practices are also regarded as relationship-oriented by inspiring, recognizing, supporting, team building, networking, mentoring, and rewarding (Kathuria & Partovi, 1999). Through agile practices, product development teams focus on developing highly committed, involved and motivated individuals (Vãzquez-Bustelo, Avella, & Fernãndez, 2007).

The software development community has shown considerable interest in this new approach (Barnett, 2007; Rosenberg, Stephens, & Collins-Cope, 2005). The results of surveys on agile method adoption by different sources have claimed that agile methods have been widely embraced for software development (Barnett, 2007; Ambler, 2007).

However, if agile methods are to deliver on any of these promises, then they must be well understood. Successful agile development requires a change in the style of product development and a transformation in organisational culture and structure (Berger, 2007).

Continuing success is dependent upon the ability of adopters to mould an agile method to meet development and business objectives in a market-driven environment (Highsmith, 2002). Market-driven environment have competitors and frequent changes in requirements, customer preferences and technologies. Therefore, adopters must also understand agile adaptation, which is the process of changing the agile practices to suit different development projects. Adaptation is the key driver for achieving continuous success with agile methods (Abrahamsson et al., 2003). This thesis investigates agile software development environments to provide an understanding of agile method adaptation.

1.2 Background to software development process and methodology

The previous section provided an overview of the evolution of software development and the emergence of agile methods. Next, a brief background to the software development process is provided together with the related research problem. Later, it is followed by a section on the theoretical foundation for the research, the research questions and research contributions. The final section provides a conceptual model for the study together with a brief outline of the subsequent chapters.

1.2.1 Definition of process and methodology

Regardless of the approach taken, all software development has some kind of process and methodology. A process is an approach or a system for achieving a result (Runnker & Brache, 1995). A process has input(s) and output(s) with a set of activities or tasks that are described to be performed on inputs to transform them into output(s). A formal software development process has a defined set of practices with a list of major tasks and their related activities (Wang & King, 2000). There are some key development tasks such as planning, analysis, design and development (Pressman, 2005). A software development process also has a prescribed order in what has to be done, how it has to be done, what has to be produced, and what level of details and information are required (Pressman, 2005; Sommerville, 2004; Wang & King, 2000).

A software development process incorporates an appropriate development methodology and the required infrastructure (Pressman, 2005). A software development methodology enables the execution of each of the tasks identified with a development process. A methodology provides a comprehensive strategy for software development, which consist of a collection of procedures and techniques (Avison & Fitzgerald, 2003; Britton & Doake, 1996), while also presenting the guidelines for completing the required activities (Satzinger, Jackson, & Burd, 2005).

The development infrastructure provides the automated and semi-automated tools and documentation support for undertaking the development tasks. Tool and documentation support is regarded as necessary and a means to enhance productivity of individuals in the team and to achieve a better quality product (Schach, 2004).

In the next section some key aspects of agile process and methods are introduced.

1.2.2 The agile approach to software development

The agile approach to software development is based on the understanding that software requirements are dynamic, where they are driven by market forces (Fowler, 2002; Cockburn & Highsmith, 2001; Beck, 1999a). This approach incorporates shared ideals of various stakeholders, and a philosophy of regularly providing the customers with product features in short time-frames (Southwell, 2002). This frequent and regular feature delivery is achieved through a team based approach (Coram & Bohner, 2005) and by having an automated development and testing environment (Holmes & Kellogg, 2006).

Agile teams consist of multi-skilled individuals (Fowler, 2002). The development teams also have on-site customers with substantial domain knowledge to help them better understand the requirements (Abrahamsson, Salo, Ronkainen, & Warsta, 2002). Multiple short development cycles also enable teams to accommodate requests for change and provide the opportunity to discover emerging requirements (Highsmith, 2002). The agile approach promotes micro-project plans to help determine more accurate scheduling and delivery commitments (Smits, 2006).

1.2.3 Agility

The agile approach to development is about agility of the development process, development teams and their environment (Boehm & Turner, 2004). Agility is the ability to sense and respond to business prospects in order to stay inventive and aggressive in an unstable and rapidly shifting business environment (Highsmith, 2002).

The agile approach for software development must employ a “barely sufficient” process (Highsmith, 2002 pg xxvii). Through agility, teams quickly adjust their process to deliver features and products (David, McCarthy, & Sommer, 2003). The agility characteristics of individuals in agile teams are most important for instantaneous responses to try out improvements to their development practices (Jin, Wang, & Palaniappan, 2005)

1.2.4 Undocumented agile approach

The agile discipline is driven by tacit knowledge of the approach and method practices (Kahkonen, 2002). Agile development is dependent upon the implicit knowledge of individuals. An excellent technical and domain knowledge and substantial local

development experiences are required for agile development. Disseminated through the practices and culture of agility are product knowledge, technical skills and development experiences amongst newcomers (Baker & Thomas, 2007).

The agile process endorses learning consistently through development experiences, face-to-face interaction, and frequent retrospectives of the development tasks (Beck, 1999). The agile acquaintance culture also ensures learning from one another by working together on development tasks or to learn by requesting help to acquire or to upgrade skills and knowledge (Faegri, 2009).

The agile process is driven by its four values and twelve principles, listed in Table 1 (Cockburn, 2001; Dennis, Wixom, & Tegarden, 2005). The agile values represent the core beliefs of its approach for successful software development. These values are about having instantaneous interaction and communication in a product development environment among the key stakeholders. The twelve principles are the guidelines to achieve the agile values to ensure successful development. These principles are about selecting agile method practices to have high performing teams for frequent and regular delivery of high quality software.

Table 1 Agile Values and Principles as listed in the Agile Manifesto

Agile values: 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 principles: 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 time scale. 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-organising teams. 12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Source: Agile manifesto (2001)

1.2.5 Agile methods

In its early days, these methods were known as light-weight methods and regarded as informal means for developing software. The agile approach has many software development methods under its umbrella. Table 2 provides a list of some of the agile methods. Each agile method has its own guiding principles with a number of practices.

Most of the methods under the agile umbrella were created before the agile manifesto was established in 2001. The increasing interest in this approach led to the establishment of a professional body named the Agile Alliance (Cunningham, 2001; Highsmith, 2002).

Table 2 Agile Methods

Method name	Year
Dynamic Systems Development Method (DSDM)	1995
Scrum	1995
Crystal	1998
Extreme Programming (XP)	1999
Internet-Speed Development (ISD)	1999
Adaptive Software Development (ASD)	2000
Feature-Driven Development (FDD)	2002
Agile Modeling (AM)	2002
Lean Development (LD)	2002

Source: Highsmith (2002), Abrahamsson et al. (2003)

The agile approach for software development does not strictly enforce adoption of practices only from a chosen agile method. A hybrid agile method could be assembled by selecting and adopting practices from different agile methods or prescriptive methods (Ambler, 2006). A hybrid method implies that agile adoption can also be gradual.

The agile approach has the potential to enable teams to achieve continuous development success (Thomas, 2008). As an adaptive approach, it is an attractive development process in a dynamic business environment. Many current software development projects are of short durations requiring an internet speed development and delivery. Short duration projects appear to make the agile approach with a goal of frequent and regular delivery of software highly suitable for this type of environment.

1.3 Method adaptation

In the IS development community, it is widely accepted that development projects are different from one another. Several researchers have identified that methodology for

projects is appropriately modified to mitigate their unique risk factors (Avison & Wood-Harper, 1991; Brinkkemper, Saeki, & Harmsen, 1998; Ørvik, Olsen, & Sein, 1999; Slooten & Hodes, 1996).

Method adaptation, method tailoring or method engineering is the change or modification of a software development method (Brownsword, Oberndorf, & Sledge, 2000). The adopted agile approach for software development requires on-going adaptation to mitigate emerging risk factors impacting projects such as their scope, quality, productivity, budget, and delivery schedules (Cockburn, 2000).

1.3.1 Adaptation definition

Method adaptation provides development competencies relevant to market needs to be able to deliver timely and specific features. In this environment, inappropriate competencies result in project failures with substantial financial implications. Agile teams adapt their development methods to enhance their product development strategies and remove barriers to product innovation.

A specific definition of adaptation given by Aydin, Harmsen, Slooten, & Stagewee, (2005, pg 128) in relation to agile software development methods is as follows;

> A process or capability in which human agents through responsive changes in and dynamic interplays between contexts, intentions, and method fragments determine a system development approach for a specific project situation.

The context (contextual factors) in which development takes place makes projects different from one another. The contextual factors are also the risk factors, which emerge from within the organisation (development practice, team, project management, cross-functional and management support) and also outside the organisation (suppliers and competition) impacting project implementation (Hofstede & Verhoef, 1997). The intentions are driven by potential benefits such as delivering innovative and high quality features ahead of competitors or to compete in a new market. The context and intention create the need to adapt the method for different projects. Method fragments are used to adapt an agile method (Adyin et al., 2004). The collections of method fragments are put together to create a development method or an approach (Harmsen, 1997).

1.3.2 Research problem

The agile approach for software development advocates a change-driven or a market-driven product development environment emphasising adaptation as critical for development success (Boehm & Turner, 2003). However, there is lack of information on agile adaptation (Aydin, Harmsen, Slooten, & Stagewee, 2005). In a market-driven environment, adaptation provides the capability to innovate better and faster, and to respond quickly to competitive initiatives, new technologies, and emerging customer requirements (Highsmith, 2002).

For this reason, knowledge on how adaptation works in practice is essential. An understanding of agile adaptation factors provides method users with information on product development strategies, method practices and development infrastructure that need to be improved, altered or replaced to overcome emerging problems or risks impacting the delivery of software. This research investigated two agile software development houses to identify and analyse the adaptation factors used in practical agile development. Adaptation factors are the elements which drive adaptation of the agile approach for software development.

1.4 Research into agile adaptation

An extensive literature review was undertaken by the researcher to identify an appropriate information systems development method adaptation theory. A theory helps to understand what is being studied (Yin, 1994). No such adaptation theory appeared to be established for software development approaches, methodologies or process models.

The literature review identified Fitzgerald's (1998) adaptation framework, known as the "Framework for IS Development Process", as a relevant adaptation framework for this study. This is an empirically grounded framework developed over a period of time using quantitative and qualitative research methods. This framework identifies three major adaptation factors; the profile of the development environment, covert and overt factors. The constructs and relationships from Fitzgerald's framework served as the theoretical basis for this study.

According to Fitzgerald's framework, the methodology-in-action is adapted from the originally adopted formalised methodology. A methodology-in-action for a project consists of a collection of new and modified practices together with the other method

practices. Fitzgerald's adaptation framework has several components: (a) development context: business opportunity and problem situation, (b) developer/methodology user and developer-embodied factors, (c) profile of development environment, and (d) roles of methodology: overt intellectual vs. covert political factors. This study investigates the three adaptation factors identified in the framework (the profile of development environment, overt and covert factors) in real agile product development environments in order to identify those relevant to agile adaptation.

1.4.1 Research questions

The goals of a qualitative inquiry are to help identify, describe and explain a phenomenon (Miller & Crabtree, 1999). These goals must help to achieve the practical aim of a qualitative study (Maxwell, 2005). This research aimed to identify, describe, and explain the adaptation factors of an agile software development approach. The focus of this research was to identify and provide practical information on adaptation factors understandable and applicable for adapting an agile approach. To ensure credible research findings, the investigation involved a rigorous effort with agile practitioners.

The main research question for this study is as follows: how does adaptation work in an agile approach? The following sub-questions will be addressed in order to answer the main research question:

a. How and why do organisational factors influence the adaptation of an agile approach?
b. How and why do political and intellectual factors influence the adaptation of an agile approach?

These constructs to identify adaptation factors were established from Fitzgerald's adaptation framework. These constructs guided data collection and analysis. A case study approach was chosen to investigate the *how* and *why* research questions to determine the adaptation factors that are relevant to an agile approach and how they are applied in practice. Two highly successful international software development houses, one from New Zealand and the other from Australia, were part of this case study research. Both had successfully adopted agile approaches for their product development.

1.5. Research contributions

The thesis makes the following contributions to the adaptation knowledge of agile development process and methods:

The thesis identifies key agile adaptation factors. The study examines, identifies and provides deep understanding of several strategic organisational factors as key influences on adaptation decisions for agile methods. The thesis also examines, identifies and provides deep understanding of the overt and covert roles of agile methods as crucial influences on the agile method adaptation decisions. The thesis provides an adapted agile adaptation framework.

The study provides a deep understanding of the need for stakeholder collaboration for agile adaptation decisions. Agile software development activities are not exclusive to software development teams but are a cross-departmental function. The practices that enable them to work together are mutually achieved and are accordingly adapted rather than done solely to meet the needs of the development function and resulting in negative consequences for others.

Dynamic and static adaptation approaches are both suggested in the literature for agile adaptation and identified as being driven by coaches, project managers and method engineers (Aydin, Harmsen, Slooten, & Stegwee, 2005). However, this study identifies that agile adaptation is mostly dynamic and is driven by the actual method users at the ground-level. This study discovered that in a market-driven product development environment engineers are fully empowered for decision-making because they are being made accountable and responsible for all product deliverables. Engineers as method users adapt to carry-out additional tasks such as coaching, project management and method engineering activities enabling more accurate and quicker decision making.

This thesis documents the agile adaptation experiences of two software development houses who are world leaders with their products, adopting agile philosophy and hybrid agile methods. This thesis provides understanding on agile adaptation and helps practitioners to determine if their product development environment is capable of such adaptation or how they might achieve it. This thesis also provides understanding that in a market-driven environment an agile approach adapts the entire organisation into a single product development unit. This implies that changing the practice of a

development team is just the beginning of the adaptation process and that the other functional teams must also adapt to the agile philosophy.

1.6 The study roadmap

Figure 1 shows the roadmap to this study. The theoretical and methodological frameworks are accordingly provided in chapter two and chapter three.

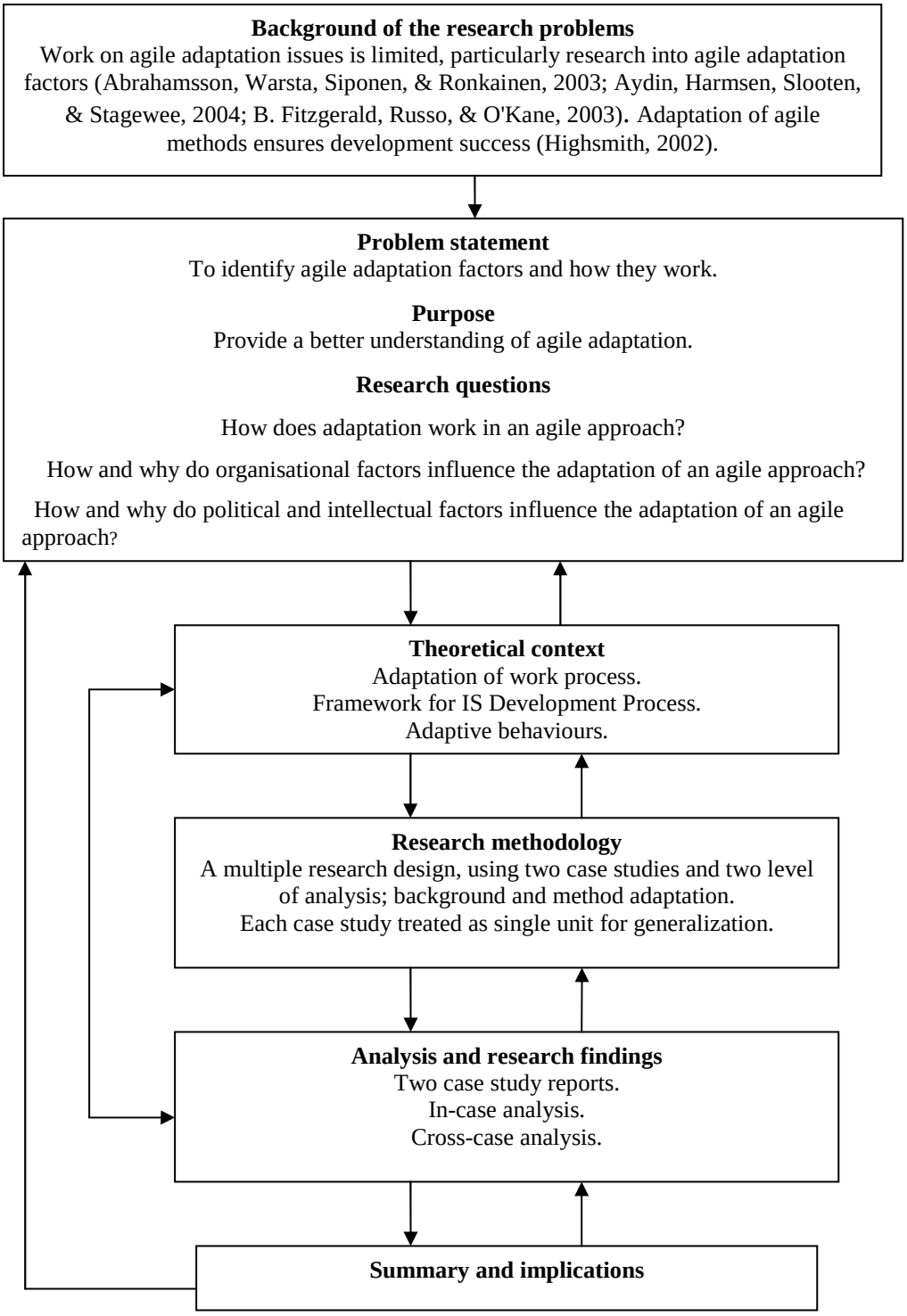


Figure 1 the study roadmap

1.7 Study Overview

The outline of this study is as follows:

Chapter two provides key adaptation literature on agile development methods and organisations (workplace and behavioural adaptations).

Chapter three provides a description of the application of case study research methodology to answer the research questions. It also includes a methodological framework for the research.

Chapters four and five present the two case studies of agile method adaptation (New Zealand and Australian case studies). The chapters include a description of case study backgrounds and agile method adaptations based on Fitzgerald's adaptation framework.

Chapter Six provides the in-depth cross-case analysis using key adaptation literature to identify and explain each case study's agile adaptation factors.

Chapter seven provides conclusion on research findings, discusses the practical and theoretical implications, states the limitations for the study and provides recommendation for future research.

Chapter two: Literature review

This chapter introduces a framework to identify and explain adaptation factors in agile software development including providing information on other alternative adaptation frameworks. This chapter also provides an examination of the existing literature reviewing the main behavioural and work practices of agile organisations, the common practices of agile development methods and theoretical perspectives on agile software development, integrating them with four main components of the adaptation framework; the profile of the development environment, overt factors, covert factors and methodology-in-action.

2.1 Theoretical basis for the research

A research project requires some related theory or a conceptual framework to guide the research design decisions; these include selection of case study sites, data collection methods, and data analysis (Maxwell, 2005). The researcher must have a holistic view and be careful that the theory does not impose the research, whereby the researcher is not able to see other categories of data and relationships emerging (Lincoln, 1999). New information and relationships that emerge should be identified and used to update the theory or framework to make it relevant and current (Miles & Huberman, 1984) .

An initial literature review identified Fitzgerald's (1998) adaptation framework, known as the "Framework for IS Development Process" (Figure 2) as appropriate for this study into the adaptation factors of agile methods. The constructs and relationships from Fitzgerald's framework therefore serve as the theoretical basis for this study. The framework lists factors that can be used to modify an adopted software development method in a dynamic business environment.

According to this framework, in any software development project the methodology-in-action is adapted from the original adopted formalised methodology. This framework describes several components, such as: (a) development context: business opportunity and problem situation, (b) developer/methodology user and developer-embodied factors, (c) profile of development environment, and (d) roles of methodology: overt intellectual vs. covert political factors. Some of these components as adaptation factors (profile of development environment, covert and overt factors) influence the adaptation of a software development methodology. The focus of this study was to investigate

these factors in real agile software development environments in order to identify those relevant to agile adaptation by applying this method adaptation framework.

2.1.1 The components of the Fitzgerald's adaptation framework

Briefly described below are the various components and related factors that influence method adaptation in Fitzgerald's adaptation framework.

1. **Original formalised methodology**

This is the adopted method(s), which the team thinks is relevant for their software development work for the time being. However, it may not be applied exactly by the adopters as prescribed or intended by the methodology developer for software analysis, design, and development tasks.

2. **Methodology-in-action**

This is the adapted or tailored version of the adopted method that is made relevant to the current software development project.

3. **Development context**

The context in which the software development takes place must be a good business reason. This context could be taking advantage of opportunities that exist in the market place or trying to quickly solve business problems by developing or improving a software product that would take care of such issues.

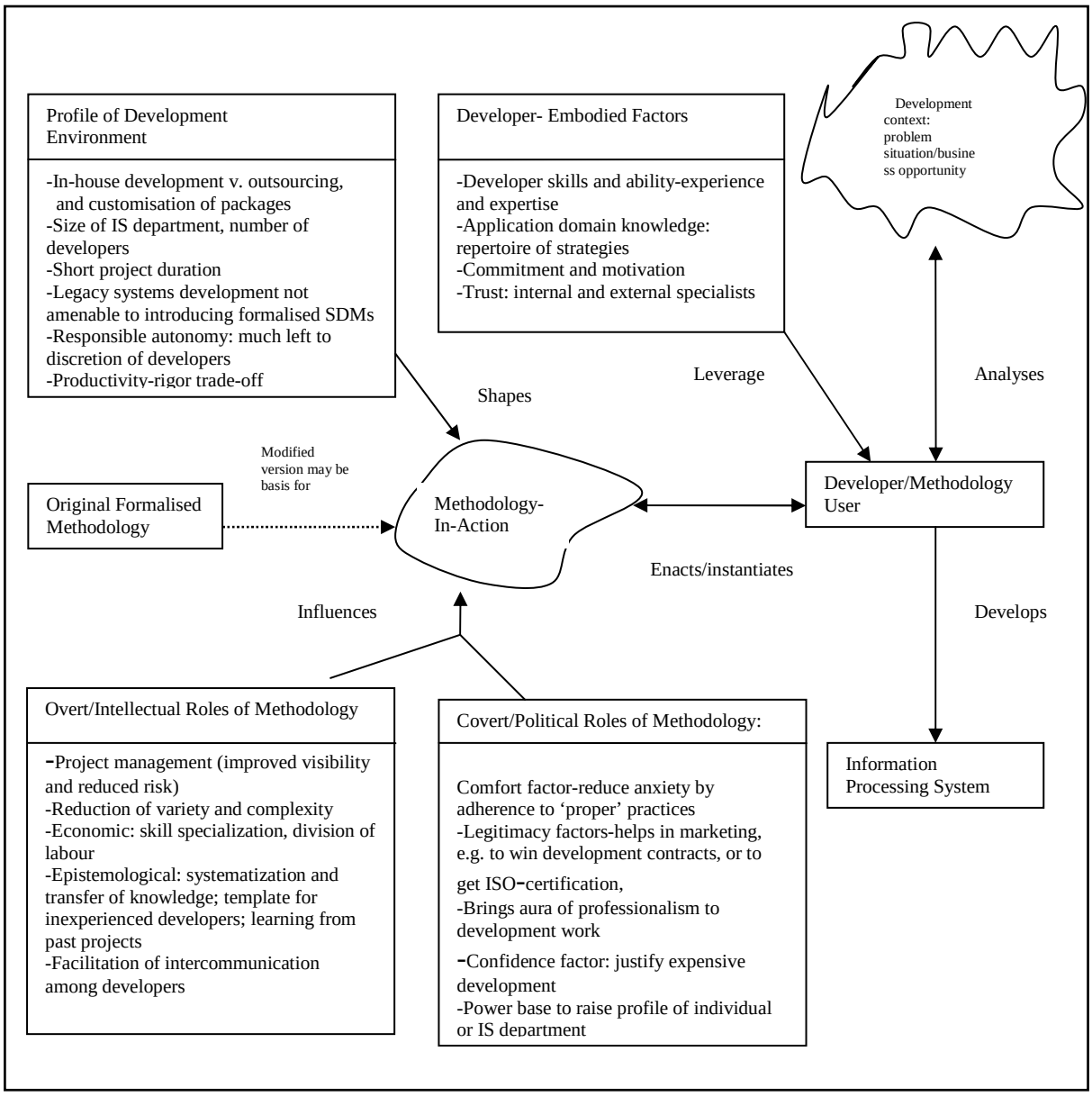


Figure 2 Framework for IS Development Process: adopted from (Fitzgerald, 1998)

This framework emphasises the need for the software engineering department to closely align its tasks with the organisation's business goals and objectives. The software engineers' involvement at the strategic business decision making level is critical; they must play a major role in identifying the software needs of the business organisation. The adapted method must help to deliver business value for the organisation.

4. Methodology users

The framework clearly identifies the software developers as the main method users. The method users help to identify, capture, design, develop, test, and implement the features and functionalities of a software product as dictated by the adapted method.

5. Developer-embodied factors

The skill sets and capabilities of developers are regarded as critical for the application and method tailoring of an adopted method; these two characteristics are identified as major influences impacting development productivity.

The other critical factors are expertise and knowledge gained by developers from different projects providing them with the necessary ideas, tips, and information for deciding how to adapt the method for the next project. The framework identifies the specialist skills and talents of individual developers as an important factor in deciding task allocation. The developer's commitment, motivation, and trust are identified as critical for project success. In addition, trust in the contribution of external specialists is also cited as critical for project success.

6. Profile of development environment

The framework defines the development environment as a significant factor for the method adaptation. Profile of development environment consists of organisational factors, influencing product development. The adaptation factors of the development environment are explained below.

Development methods are used if there is a high-level of in-house development. However, they may not be used as much if there is a high level of software package acquisition and/or outsourcing of the development work. They are mostly used by large organisations having a large IS department (more than 20 individuals). In-house development and the size of IS department are two method adaptation factors identified in the framework.

Other reasons for using software development methods are the long term durations of projects (more than 9 months); however, these projects are sub-divided into short-term projects and delivered incrementally. The short term project is identified as an adaptation factor.

According to Fitzgerald, the organisations that did not invest in software development methods acknowledged that it could add rigor to software development, however productivity issues are their major concern and they believe that good enough systems could be built without using them. However, outsourcing and subcontracting are

prevalent due to perceived low quality of in-house development. As such, productivity-rigor trade off is identified as an adaptation factor.

The formalised or structured methods are more suited to cumbersome and monolithic third generation software development environments. As such, legacy systems development is also identified as an adaptation factor.

Reasonable developer autonomy leads to developer motivation revealing that management style in projects and organisations is an important factor for successful software development. Therefore, responsible autonomy is also identified as an adaptation factor.

7. Roles of methodology: overt/Intellectual Vs. covert/political

The framework plainly distinguishes two different roles that a development method plays and which are used to adapt a method. These are overt and covert roles.

a. Overt intellectual roles

The overt or intellectual roles of development methods are defined in the adaptation framework as the obvious reasons why the methods are used or adopted for development work.

The framework states that the use of a development method makes the development work more open and clear for project management and control activities. Therefore, the development process becomes more transparent to all the stakeholders. According to the adaptation framework, it enables minimization of risks by having milestones and deliverables. The method also enables reviewing the progress of development work. Therefore, the project management, and the reduction in variety and complexity are identified as two key adaptation factors.

Software development methods must provide a framework for setting up the communication mechanism among the developers. Hence, the facilitation of intercommunication among developers is identified as an adaptation factor for development methods.

Another intellectual role stated in the framework is the economic rationale. The framework recognises skill specialisations, allowing payment of different rates to developers as way to reduce the cost of the development. Different rates are the

outcome of having several phases in the development, which require having different individuals with specific skill sets. Therefore, the economic factor for skill specialisation and division labour is identified as a development method adaptation factor.

The other intellectual reason identified in the framework for adapting a method is for the epistemological benefits i.e. organising in a systematic manner and providing for the development information and knowledge, especially for inexperienced developers. The epistemological factor is also identified as adaptation factor for development methods.

b. Covert/political roles

The covert or political roles of development methods enable a software product development environment which has a good feeling, thinking and functioning well in the quest of delivering software products consistently, regularly and when they are needed.

There are a number of political roles of development methods which are identified in the framework as method adaptation factors. The comfort factor helps the developers to cope with the stressful complexity of software development. The legitimacy factor helps to win development contracts or in marketing products. The confidence factor allows the justification of expensive development investment decisions. The aura of professionalism enables developers to negotiate realistic requirements and work demands. The audit trail records decisions made and the reasons for making them. Finally, the power base factor helps to raise the profile of the individual or the IS department.

2.1.2 Other adaptation frameworks

This section provides information on other adaptation frameworks which were considered for this study and provides reasons why they were not adopted.

SDM (systems development method) deployment model is another adaptation framework, which is proposed by Ørvik, Olsen & Sein (1999). SDM Deployment Model defines the method deployment process, which involves SDM interpretation process, SDM adaptation process (resulting in two SDM visions- one as the organisational standard and the other adapted for each individual project), SDM uses

process (use of adapted method) and epilogue (the final software product as a result of using the adapted method).

This adaptation framework identifies factors such as the structure, control systems, leadership style, values (risk taking or risk adverse, stability or innovation), and cultural factors of an organisation influencing the adaptation of a development method. These factors are captured in Fitzgerald's adaptation framework with the profile of development environment factor, which identifies a number of organisational factors (organisational strategies) as the method adaptation factors.

One of the major drawbacks is that this framework (SDM deployment model) does not have the same empirical foundation as Fitzgerald's framework. However, its major contribution is that it focuses on how a method is deployed inside the organisation identifying factors such as premise mismatch (the paradigmatic assumption in the different versions of methods) and contextual mismatch (the mismatch between paradigmatic assumption inherent in a method with beliefs and assumptions of the adapter and organisation), which may affect the adaptation process of a method. These two (premise and context) mismatches are captured through the original formalised methodology and methodology-in-action components in Fitzgerald's adaptation framework. In addition, the SDM deployment model is not as comprehensive. Fitzgerald's framework also provides a list of overt and covert factors making it more suitable theoretical framework to identify a wide range of agile method adaptation factors.

Another adaptation framework is proposed by Baskerville & Stage (2001), which is known as the framework for accommodating work practices. This framework has three main components, which are method fragments, work practice, and social process providing a basis for on-going method adaptation. The method fragments relate to the overt factors and the work practice component relates to the profile of development environment factors in Fitzgerald's adaptation framework. Similar to the SDM deployment model, this framework too is not as comprehensive as Fitzgerald's framework.

Fitzgerald's adaptation framework provides a broad list of organisational, overt and covert factors for method adaptation. This framework also has original formalised methodology and methodology-in-action components to capture the paradigmatic

assumptions underlying a method and of the adapters. Hence, Fitzgerald's framework includes adaptation concepts highlighted with Baskerville & Stage's (framework for accommodating work practices) and Ørvik, Olsen & Sein's (SDM deployment model) adaptation frameworks. With an empirical foundation, Fitzgerald's framework is more appropriate to guide data collection in relation to the adaptation of agile methods for this study.

In the remainder of this chapter, agile literature is evaluated and integrated within the four main components of the framework; profile of the development environment, overt factors, covert factors and methodology-in-action. Cases where agile literature implies changes to the framework are also highlighted.

2.2 Profile of the development environment

2.2.1 Agile organisation

Agile organisations have the ability to create and respond swiftly to both anticipated and unanticipated changes in their business environments, regardless of being in manufacturing or software development industry (Highsmith, 2002). This section provides insights into some key literature on agile organisations and agile software development.

2.2.1.1 Organisational culture

A study by Christopher & Towill (2000) highlights the organisation's agile culture, people and communication as the main success factors for product development in a volatile (market-driven) environment. According to Christopher & Towill, agile culture provides the ability for quick responses for making available high quality product features through functional, structural, process, employee and management dynamisms. In-house development requires acceptance of agile culture to have on-going flexibility through adaptation of its functional setups, roles, tasks, responsibilities and management style, not just their agile method practices.

A key factor in a market-driven environment is that the development environment requires an organisation-wide shift towards the agile culture, not restricting agility to their development process (Aitken, Christopher & Towill, 2002). In-house software development as a strategic factor also requires an organisation-wide adoption of the

agile culture. An insight into achieving a supportive agile software development environment at ThoughtWorks (an international IT company) is provided by an experience report by Doshi & Doshi (2009). According to Doshi & Doshi, through the adoption of agile culture organisation-wide the management, administration, recruitment, and infrastructure support groups at ThoughtWorks now collaborate closely with each other and with developers on their tasks, swiftly serving one another's needs.

However, a study by Fogelstrom, Gorschek, Svahnberg, & Olsson (2009) shows that agile software development methods do not provide any support for long-term product development in a market-driven environment. According to Fogelstrom et al., the agile methods place limitations on product management activities, especially on pre-project (vision and road map planning) activities. Successful adoption of the agile culture organisation-wide will require in-house development to also adopt agile method practices supporting product management functions.

Bespoke development (custom-made product development) relies on the customer to decide what functionalities should be developed (Fogelstrom et al., 2006). However, the market-driven product is developed through an iterative approach and delivered in multiple versions through release cycles (Gorschek & Wohlin, 2006). According to Gorschek & Wohlin in a market-driven environment there is a continuous incoming stream of requirements from internal (engineers managers, marketing department, market surveys, product support team) and external (potential and existing customers and business partners) sources. A shared agile culture is critical for organising high level requirements enabling short duration projects and delivering implemented features in multiple release cycles. For agile in-house development, multiple release cycles enable strategic feature releases.

Success in adopting the agile culture organisation-wide is also dependant upon the business function's ability to adapt to support in-house development (Cho, 2009). Cho's experience report provides insight into an agile software development environment at Liquidnet (a financial company with 400 employees) which required the user experience team (a business unit) to adapt their roles and processes for the pre-project activities and tasks (high level planning). Initially, they had to hire an agile coach to facilitate the process change to integrate and adapt the high level planning activities

with their in-house development environment. High level planning is a critical agile development activity identifying high level requirements and their estimates, which then are allocated into short duration projects for implementation.

Similarly, an experience report by Maples (2009) highlights that the business functions must also adapt to support agile development. Maples provides insight into Borland's (a software development organisation with 300 team members in five different locations) agile transformation of their in-house software development environment causing friction with their marketing, product management and sales units. According to Maples, at Borland the concept of continuous collaboration throughout their Scrum cycles was very disruptive for their business functions and they had to hire an agile consulting firm to help implement product planning practices based on the agile culture for collaboration between development and business units. To gain business function support for agile in-house development, it is critical that appropriate direction and learning are provided to them.

The agile culture helps to maintain product leadership in a dynamic and constantly changing environment by delivering customer focused products, cooperation, learning and a culture for change (Sherehiy, Karwowski & Layer, 2007). For in-house software development in agile environments, the cross-functional and client collaboration (cooperation) is critical enabling development teams to learn and understand the business goals and objectives, extract valuable information on the high-level features, and to gather the tacit knowledge with regard to customer preferences.

According to Lindvall et al., (2002) to be agile for software development is a cultural thing and if the culture is not right then organisations cannot be agile. Lindvall et al., identify rapid communication, dynamicity in requirements change, trusting people and fast feedback from customers as the key agile culture for in-house software development. While all of these are important, rapid communication is critical for in-house development agility. This is achieved through co-location of developers and business support roles for in-house development, facilitating instantaneous cross-functional collaboration for quick decision making.

Using iterations, user stories and velocity as in-house development practices does not mean that software development environments have successfully adopted the agile culture (Cockburn, 2005). According to Cockburn, agile culture is about delivering and

collecting feedback from the users on the implemented and fully tested features on a regular basis and in shorter time scales. Hence, productivity and rigor are strategic factors in agile market-driven development environments rather than a trade-off between the two as stated in the adaptation framework.

When shifting towards an agile culture organisation-wide in a bureaucratic environment, it has to evolve rather than be imported or imposed (Berger, 2007). Berger provides an experienced report on agile development in a bureaucratic arena (an innovative, real-world government information systems development project in the UK). This experience report highlights the key stakeholders' behaviour and attitudes influenced by their bureaucratic and hierarchical society as problematic for the agile project, incurring implementation delays. Agile culture is critical in the development environment allowing responsible autonomy to empower developers for quick decision making to avoid implementation delays. According to Berger, despite the high levels of commitment by senior management to remove the blame culture, the inherent working culture had a major influence on the agile project.

The organisational culture is an outcome of its mechanistic or organic (agile) structure (Burns & Stalker, 1994). Mechanistic organisations develop standardised products, emphasise development efficiency and compete based on product cost, quality and delivery (House, 1991). However, standardised software products have inferior product quality. Mechanistic organisations are hierarchical, structured, and procedural while being driven by order and control (Wallach, 1983). In these organisations individual behaviour is predetermined (Burns & Stalker, 1994). Such behaviour and control is counter-productive for in-house agile development requiring responsible autonomy for developers encouraging creativity to implement innovative product features.

Organic organisations are constantly changing and adapting organisations (Burns & Stalker, 1994). Organic organisations are agile organisations. They are innovative and flexible adapting to environmental demands to operate in an unpredictable and changing environment (House, 1991). Similarly, in-house development must become agile in the current turbulent software market for continual innovation and flexibility with processes and features (Highsmith, 2002). These capabilities are achieved through an evolution of agile culture in software development environments. Agile organisations compete based on new features and products, product customisation, and process and technology

innovations (House, 1991). To achieve these, the agile in-house development environment must provide individual empowerment through responsible autonomy.

2.1.1.2 Strategic product planning

Agile organisations have a strategic product (vision and roadmap) planning process (Feldman & Albert, 1984). A strategic agile product planning process requires business value features to be swiftly investigated, analysed and prioritised based on information gathered from a wide range of sources on a continuous basis. Product planning in agile environments also determines business value for legacy system development, helping to prioritise all development according to the market value. Hence, legacy system development is also subjected to agile practices rather than no formal practice as stated in the adaptation framework.

The product planning process requires stakeholders from different functions of an organisation to collaborate and jointly make decisions to develop a roadmap (Lehtola, Kauppinen, & Kujala, 2005). This cross-functional approach improves communication and ownership of a product plan (Albright & Kappel, 2003). Developer input contributes towards a better understanding of high-level requirements of legacy and new developments in agile development environments.

Product planning is a challenging process to distil high value features and to develop a roadmap plan for market releases (Phaal, Farrukh, & Probert, 2005). A survey by Farrukh, Phaal, & Probert (2001) highlights some key product planning challenges including starting up the process, keeping the process 'alive', and developing a robust process. A well created product plan prevents developers from procrastinating implementation in agile development environments.

There are several product planning approaches such as business-led, method-driven, architectural, administrative, and organisational (Earl, 1996). Agile organisations adopt the organisational approach based on informal, business themes, evolutionary change, teamwork, education, and decentralised decision-making (Gunasekaran & Yusuf, 2002). For high level planning, it is critical that developers jointly provide estimates learning in advance to prepare for feature implementation.

According to Nejmeh & Thomas (2002), agile methods provide the flexibility with planning product functionalities and releases but these methods do not address: (a) how

the development team should interact with key stakeholders and product managers, (b) how a product manager and development team can produce consistent feature requirements and priorities that will satisfy multiple customers, (c) techniques for product managers to balance the conflicting demands of multiple customers, and (d) techniques for product managers to determine the value of proposed features. The first two are critical agile issues where in-house development requires permanent stakeholder support and must also provide technical perspectives on high-level requirements allowing product managers to accurately determine the business value of high level requirements.

Agile organisations' planning approach significantly influences the organisational performance, business strategy, and resource commitments by having a steering committee for decision making on new developments (Jacques, 2007). Development function representation in a steering committee is critical to ensure that their concerns for proposed features are considered before they are allocated in short duration projects for agile in-house development.

2.1.1.3 Responsive behaviour

Agile organisations have flexibility to adapt their processes, behaviours and structures in a turbulent environment (Reed & Blunsdon, 1998). In this environment, there is a rapid change in customer tastes and preferences, numerous product innovations and frequent innovations in the development process (Vãzquez-Bustelo, Avella, & Fernãndez, 2007). Agile organisations have responsive behaviour to swiftly determine and deliver market-driven products before competitors (Kusiak & He, 1997). Responsive behaviour is the individual and organisational ability to swiftly learn, adapt and respond to deliver the market needs. A critical part of this behaviour for agile in-house software development is learning practices to swiftly acquire new knowledge, skills and support to deliver market-driven products.

An understanding can be gained into what contributes to successful responsive behaviour through the analysis of findings of a longitudinal case study by Salo & Abrahamsson (2005). Their study involved five successive agile software development projects, which shows that successful developments were achieved through a constant collaboration on process improvements between the project teams and an organisational level group. According to Salo & Abrahamsson, a majority of improvements agreed

within the project team required organisational level implementation support and also, the support for required learning activities. For agile in-house development, this organisational learning is a decisive developer-embodied factor for method adaptation to support their responsive behaviour.

A study by Takats & Brewer (2005) shows building relationships with customers and developers for regular communication is very important to determine vital product functionalities. Support for swift learning of new requirements is critical for responsive behaviour to deliver appropriate features.

Boehm (2002) identifies pitfalls in agile methods such as the requirement for highly skilled developers, customer involvement, architecture, refactoring and size of a development team. Beohm highlights that not all developers are highly skilled and not all teams will have significant levels of tacit knowledge. High technical skills and tacit knowledge are critical developer-embodied factors for agile in-house development. According to Boehm refactoring effort increases with less-than-great developers and the coordination of teamwork becomes increasingly difficult beyond 15 to 20 individuals. The size of IS department in agile environments is a critical factor requiring responsive behaviour for its adaptation to organise development efforts in small teams and also to encourage learning from one another.

A report by Heimgartner & Locke (2006) states that executive support and investment in tools for technical writers are important for them to meet challenges and assist development teams in becoming more proficient and efficient in agile development. According to Heimgartner & Locke the success of the writers depends heavily on the involvement and commitment of development and product managers for documentation activities. For agile in-house development, the responsive behaviour is critical to deliver documentation in parallel with iterations. Heimgartner & Locke provide a report on the documentation teams of two different agile software development organisations. At one the writers suffer due to looming agile iteration deadlines while at the other the writers feel the agile pace for writing documentation is not sustainable. According to Heimgartner & Locke these problems can be overcome by employing more writers, implementing technology to help writers and having fewer documentation deliverables. For agile in-house development, most important is the tool support for high productivity gains without a trade-off in rigor.

Responsive behaviour also provides the capability to quickly recover from the actions taken (Sharifi & Zhang, 1999). Law & Learn (2005) provide an experience report of the agile in-house development at TransCanda, a leading North American energy company highlighting their responsive behaviour to unexpected changes during a key project. This project went through changes in manager, shared programmers with other projects, had changes in requirement half way through the project, and shortened the testing period. In addition, two programmers were taken off the project for a higher priority project, a new programmer was hired but resigned a month before project completion, and a new person was hired as replacement. Their responsive behaviour for changes in resources included firstly, adopting pair programming to up-skill newcomers followed by adopting solo programming as well (because of sharing programmers with other projects jeopardizing the benefits of pair programming) which was adapted with code inspection. They also adopted a Wiki Web site to centralize and facilitate understanding in systems requirements. Their responsive behaviour for change in requirements was through adoption of a master backlog and sprint backlog to track the current requirements including having a business champion to prioritize the requirements. They also adapted the daily status meeting to have a representative of business and project groups to meet on a regular basis to have more efficient use of time. Their responsive behaviour to deal with a shortened warranty period was to adapt their 40 hour working week to working overtime to cover key features and business users worked overtime to perform user acceptance tests. For agile in-house development, teams must develop the ability to continue functioning to meet the development commitments and business objectives as changes are being made.

To be agile in a market-driven environment requires being customer responsive (anticipate customer requirements), human resource responsive (empower employees, have highly capable and motivated employees and provide autonomous decision making capabilities), global market responsive (but have operations and infrastructure tailored to the local requirements), team work responsive (this is a key part of the core competency), and responsive to continuous enhancement (includes core competency, work practice and organisational structure) (Vernadat, 1999). These are some of the major challenges for agile in-house development requiring an on-going organisational reconfiguration in structure, strategies, roles and management style.

Responsive behaviour is critical to be able to swiftly adjust business objectives and goals and deliver products according to market requirements (Breu, Hemingway, Strathern, & Bridger, 2001). For agile in-house development, responsive behaviour is also dependent upon having a collective attitude to quickly learn from one another to carry out changes and effective cross-functional effort.

2.1.1.4 Speed for learning and development

Individuals in agile organisations swiftly develop new skills and abilities including quickly adapting to a new development approach and innovative management skills (Breu et al., 2001). For agile in-house development, the collective abilities to quickly learn, adapt and respond with innovations ahead of competition are key developer-embodied factors.

According to Cockburn & Highsmith (2001) agile software development environments focus on the high talents and skills of individuals and adapt processes to specific people and teams. Cockburn & Highsmith state that individual competency is the most critical factor in project success stating that good enough individuals can use almost any process to accomplish their assignments. However, to have such competency for agile in-house development not only requires time and a supportive environment but also the ability to identify individuals who are capable of swiftly developing high expertise.

According to Shinkle (2009) individuals go through various skill levels to become an expert in agile environments; novice, advanced beginner, competent, proficient and expert levels. Novices focus on accomplishing immediate goals but require mentoring and coaching. Advanced beginners have experience and begin to develop understanding of relevant context while breaking rules but fail to experience holistic understanding. Competent individuals have considerable experience coping with real situations to solve problems. Proficient individuals begin to seek out and want to understand the big picture to correct poor task performance, reflect on what they have done and revise their approach to perform better next time. Expert individuals decide how to do things and have enough experience to dictate an intuitively appropriate response. Shinkle identified these levels in 2007 through adoption of Kanban (a concept related to lean and just-in-time production) at SEP, a software engineering company. Kanban enabled Shinkle to successfully provide learning practices at SEP to grow expertise levels and rekindle organisational interest in an agile approach where it failed to have impact when it was

adopted in 2004. For agile in-house development, high expertise is a critical developer-embodied factor providing leadership behaviours.

Most important is the ability to learn and carry out new tasks efficiently in an adapted environment delivering new products in the shortest possible time (Sherehiy et al., 2007). Individual flexibility, improvement in task speed, reduction in setup time and cross-training are the key factors facilitating shorter average development cycle times (Hopp & Van Oyen, 2004). The goal for shorter development cycles can only be achieved through expertise in agile software development (Shinkle, 2009). Achieving the shortest development cycle in agile short duration projects is essential to gain competitive advantage in the marketplace.

Automation is a critical factor for short development cycles reducing development time, achieving development flexibility, and delivering innovations to achieve high performance (Gunasekaran, 1998). Continuous testing in agile short duration projects would not be possible without automation because of the size and number of platforms of market-driven software products (Shaye, 2008). Shaye provides an experienced report describing challenges in moving a large development team from waterfall test phases to agile test methods. According to Shaye, selecting regression test coverage, funding automation initiatives and providing training for automation skills are very important. For agile in-house development, the ability to continuously adapt automated regression testing to get feedback on broken builds in an acceptable time frame is critical to maintain high development productivity.

Agile organisations also have intelligent hardware and tools such as internet and multimedia to eliminate non-value adding activities (Gunasekaran, 1999). Internet access enables short development cycles though instant communication with industry experts, product managers and customers for rapid feedback (Mathieu, 1997). Such tool support is critical in distributed agile software development environments to mitigate the inefficiency of non-face-to-face communication (Phalnikar, Deshpande, & Joshi, 2008). Distributing a development project in an agile way is challenging for example in documenting requirements to an understandable level of detail and getting questions answered quickly. According to Phalnikar, Deshpande, & Joshi message boards or Wikis instead of email, collaboration systems, IM tools, net meeting and video conferencing considerably reduce the response time. These tools are also very useful for

co-located agile in-house development teams to promptly get contextual information on features from the field.

2.1.1.5 Management practices

In an agile organisation, the management culture is to inform, listen, negotiate, motivate, and teach the workforce (Owusu, 1999). According to Owusu, agile management culture is one of trust and respect for employees where they are recognized to be showing interest in their work, capable of and willing to improve and their interests are clearly supported. For agile in-house development, management trust is a key factor to provide responsible autonomy for day-to-day decision making by developers at project level. This paradigm shift in software development environments requires very special leadership (Bouldin, 1990).

Seger, Hazzan, & Bar-Nahor (2008) suggest that in software development environments managers and co-workers must make efforts to design supportive environments to enhance team members' agile orientation. Their study investigated individual characteristics such as psychological needs, self-efficacy and perceived support from peers and managers from 376 software developers employed in two divisions of an international Israeli company. According to Seger et al., managers need to develop the leadership and interpersonal skills of developers, and to mentor the junior developers for them to cope with the demands of agile development. For agile in-house development, the leadership behaviours, interpersonal skills and ability to handle stress are critical developer-embodied factors.

Agile organisations have decentralized knowledge and control where there are fewer adherences to authority, rules and practices allowing for a higher degree of flexibility and freedom of choice (Sherehiy et al., 2007). However, agile software development organisations must learn to accommodate human-resources issues where development team members are often required to cross their job boundaries requiring significant skills and experience to adequately perform in development environments (Boehm and Turner, 2005). Hence, multi-skills are a critical developer-embodied factor for agile in-house development.

In agile management, responsibilities associated with tasks and projects are delegated to individuals and teams at the development level (Kathuria & Partovi, 1999). For agile in-

house development, self-organising teams is a critical practice requiring responsible autonomy. However, agile development teams require several informal leadership roles such as mentor, co-ordinator, translator, champion, promoter and terminator to self-organise (Hoda, Noble, & Marshall, 2010). These roles were discovered through grounded theory research involving 14 software organisations in New Zealand and India. According to this research, most of these roles are played by an agile coach while a business analyst plays co-ordinator and translator roles. However, for agile in-house development a specialist coaching role is not permanently required and a business analyst is not always permanently co-located with teams.

The management in agile organisations also adapts to macro-management practices where they manage by coaching, supporting, and leading; challenging the work-force with bold goals, providing the required resources and empowering the workforce to accomplish goals (Sharp, Irani, & Desai, 1999). According to Nerur, Mahapatra, & Mangalaraj (2005), the biggest challenge in agile software development environments is to get the project managers to relinquish their authority to become facilitators directing and coordinating the collaborative efforts of those involved in development, thus ensuring that the creative ideas of all participants are reflected in the final decision. For agile in-house development, responsible autonomy is a key factor for developers and on-site customers to jointly manage projects.

Management individuals lead innovations, provide information to become the single source of truth on innovations, reduce structural complexity and standardize business processes to remove barriers to cross-functional collaboration, to successfully implement new innovations (Lukens, 2007). These are critical activities which development, product and project managers must carry out in helping to create flexible agile software development environments.

2.1.1.6 Workforce agility

In agile organisations, development agility is achieved through a workforce that is capable of dealing with unexpected tasks (Youndt, Snell, Dean, & Lepak, 1996). Workforce agility is having highly skilled, competent and adaptable individuals capable of dealing special situations. To achieve this agility for in-house development, organisations must place a high value on individuals and their skills rather than on job roles they have.

Agile organisations continuously perform and are successful in turbulent business environments through the knowledge and skills adaptations of their workforce (Forsythe, 1997). For this high performance, building agility is critical for in-house development due to change, uncertainty and unpredictability of software requirements from sources such new technologies, changes in consumer demand patterns, change in leadership with different priorities and change in user interfaces that emerge with use (Boehm, 2008). According to Boehm, along with agility for in-house development, dependability of software produced is equally critical to differentiate an organisation's products. Agility to produce dependable software is based on identifying, managing, and negotiating mutually satisfactory (success-critical) stakeholders' requirements. Achieving mutually accepted requirements requires developers to be knowledgeable in concepts and techniques of the user environment, understanding the tradeoffs between software feasibility and application domain feasibility for deciding appropriate architectures (Boehm, 2008). Business knowledge is a critical developer-embodied factor for agile in-house development to implement flexible architectures for swift extensions and enhancements to products.

Workforce agility enables organisations to swiftly respond to unforeseen events, in a market-driven environment (Plonka, 1997). Agility facilitates individuals to work effectively in a cross-functional project team to deliver swift responses to unexpected situations (van Oyen, Gel, & Hopp, 2001). However, for agile in-house development to undertake large-scale cross-functional projects requires adapting the co-location and face-to-face collaboration practices involving product managers to gain a better understanding and visibility of success-critical stakeholders' needs, and for effective backlog prioritizing (Ktata & Levesque, 2009). According to Ktata & Levesque, agile development environments require new ways to incorporate product manager support while a steering committee (not a single individual) must identify success-critical stakeholders and prioritise their needs. These adaptations are critical for in-house development agility by enhancing the first time right decisions being made through a collective effort.

The agile workforce has an excellent attitude towards new ideas, new technologies, and self-development; learning and problem-solving abilities; consensus behaviour to accept changes; the ability to generate new ideas; and the capability to accept new

responsibilities (Plonka, 1997). However, these agility behaviours are difficult to achieve when scaling to large scale agile development (Moore & Spens, 2008). Moore & Spens provide an experience report from a large scale agile development, involving over 300 people across 3 global sites, at Siemens Solution USA. Over one year of the project they grew to have 25 scrum teams with average of 12 per team (Scrum master, 4-5 developers pairs, 1-2 analysts, and 1-2 testers). While they adopted a service oriented architecture, it became counterproductive because developers could not find an effective way to manage dependence and continuous integration due to a larger number of teams on a single project. The individual team practices reduced their effectiveness in the large-scale project environment. Moore & Spens provide critical developer-embodied factors for a large scale in-house development environment, such as: the flexibility to operate alongside other teams and individuals; the ability to take on different responsibilities, raise issues and fight for priorities; and to accept changes and decisions being made.

Organisation agility is achieved through the workforce having three behavioural adaptations; these are proactive, adaptive and generative behaviours (Dyer & Shafer, 2003). Behavioural adaptation is the change in the work behaviour of individuals. According to Dyer& Shafer proactive behaviour is the individual ability to search for new opportunities to contribute to the team's success. Adaptive behaviour is the individual ability to perform in multiple roles while generative behaviour is the individual ability to learn and educate on tasks and responsibilities of other roles while working with them (Dyer & Shafer, 2003). The proactive behaviour is a critical developer-embodied factor for agile in-house development to have individuals with an attitude to share and work together on development tasks and to take full responsibility for quality of features.

An important aspect of an agile workforce is their ability for adaptive performance behaviour (Allworth & Hesketh, 1999). Adaptive performance behaviour is the ability to deal with change and aptitude to apply learning from one task to another. According to Allworth & Hesketh, this behaviour has two components which agile work forces must adapt with; the cognitive and emotional components. Cognitive adaptation is the individual ability to learn new things, devise problem-focused coping strategies, access information on change and solve problems associated with change. Emotional adaptation is the individual ability to cope with change, not resist but allow the change

to happen, and have a positive reaction to change and to the new opportunities made available (Allworth & Hesketh, 1999). For in-house development, learning is a key factor to enhance agility and equally important is emotional adaptation to maintain consensus and collective attitude for high team performance.

The importance of emotional adaptation can be seen in an experience report by Little (2005) on the complexity and uncertainty of agile projects at Landmark Graphics, a supplier of software products for oil and gas production companies in USA. Agile projects at Landmark Graphics are classified as colts (simple projects for new products but with high market and technical uncertainty), bulls (complex projects for emerging products with high uncertainty), cows (complex projects for cash cow products with low uncertainty) or dogs (simple projects for mature products with low uncertainty). According to Little, colt and bull projects require significant agility. For complex projects, emotional adaptation is more critical for agile in-house development since these projects create significant pressure to implement next generation products with high investment.

In agile organisations, adaptive performance behaviour enables: solving problems; dealing with unpredictable work situations; learning development tasks, technologies and procedures; adapting with interpersonal and cultural behaviours; and dealing with work stress and crisis (Pulakos, Arad, Donovan, & Plamondon, 2000). The adaptive performance behaviour is grouped into three major behavioural categories: proactive, reactive and tolerant behaviours (Sherehiy, Karwowski, & Layer, 2007). According to Sherehiy et al., proactive behaviour allows solving problems and dealing with crisis; reactive behaviour allows learning new things and enabling interpersonal, cultural and physical adaptations; and tolerant behaviour allows coping with stress and uncertainty. For agile in-house development, interpersonal adaptation is extremely important to effectively interact and collaborate on tasks to meet team delivery commitments.

2.1.1.7 Development cooperation

Agile organisations achieve development agility through internal and external cooperation (Gunasekaran, 1999). Internal and external cooperation is critical for enhancing competitiveness in a turbulent environment (Sherehiy et al., 2007). Internal cooperation is having teamwork and cross-functional collaboration (Vokurka & Fliedner, 1998; Hormozi, 2001; Sohal, 1999). External cooperation is having strategic

alliances, customer and supplier collaborations, and outsourcing (Meredith & Francis, 2000; Duguay, Landry, & Pasin, 1997). For agile in-house development, external cooperation is critical to enhance development capabilities rather than choosing in-house development or outsourcing, as stated in the framework.

However, internal cooperation is extremely difficult to achieve if functional teams are not able to adapt even when they have the desire and organisational support to be agile (Markham, 2009). Markham provides an insight into organisational software development process improvement through an agile approach where management themselves did not agree to be the product owner to provide leadership at development level (to prioritise backlogs and stories) despite allocating budget to implement changes. Their service oriented teams struggled without real product manager support and were working on deliverables without any clues of the audiences. The design team had no immediate feedback when sponsors for key meetings were not available. The database team made their own list to work without product manager inputs. The matrix team resisted accepting agile practices introduced by the project manager. With the quality assurance team, individuals integrated with project teams but required quality matrices to be completely re-engineered. According to Markham, without the ability to communicate and adapt, agile in-house development will not provide real benefits. To implement changes for agile in-house development, it is very important to gain management's commitment to change the behaviours of the individuals.

Developers must learn to understand and work with each other prior to agile adoption to attain effective internal cooperation (Ruhnow, 2007). Ruhnow provides an experience report on agile adoption at Weyerhaeuser. Prior to agile adoption they characterized the personas of their programmers into cowboy, pessimist, meeting master and private. At Weyerhaeuser, a cowboy programmer has an engineering background and can add features without customer requests but often avoids code quality practices. The pessimist is a developer in QA role and others fear the pessimist looking at their work. The meeting master attempts to gain business and technical knowledge by holding long meetings at the expense of others. A private developer is not used to elevating thinking beyond what the project manager requests. To break the barriers between the developers they encouraged developers to work with each other prior to agile adoption. They had incremental adoption of agile practices to have short iteration cycles, developers involved in all phases of development, constructive conflict on process, shared

workspace, and team responsibility for quality. Their internal cooperation was further enhanced with product and project managers adapting their practices to help meet team quality goals. Key to achieving effective internal cooperation for agile in-house development is measured and conscious changes through collaborative discussions among stakeholders.

Having an effective external support through distributed and offshore software development teams does not improve productivity, quality and performance in agile software development environments (Cohen & Thias, 2009). Cohen and Thias provide an experience report on the effectiveness of offshore and distributed software development teams. They provide a case of a large financial institution which co-located the development team and completed the work on time using only 10% of the original estimated resource for offshore team and improved the time to market by 50%. The second case involved developing an embedded system for a medical device by 50 developers of two companies distributed in three locations (co-located team with 7 developers, on-shore team with 13 developers, and offshore team with 30 developers). According to Cohen and Thias, the off-shore team delivered only 27% of work assigned to them where as the co-located team delivered 78% and the on-shore team 58% of work assigned to them. On investigation of the offshore development it was discovered that they did not have any estimation of their work, no project tracking, and no metrics to measure productivity or speed despite claims by the project manager that work was on schedule. They also lacked appropriate programming skills, and hardware to integrate and test. Hence, knowledge of skill sets, practices and resources of distributed and offshore teams are extremely important when deciding for external cooperation to boost agile in-house development efforts. However, Cohen and Thias advocate creating powerful co-located teams with a focus on customer/developer relationship, cooperation, short iterations, quality and learning. This report highlights risks associated with outsourcing, suggesting in-house development capability as most critical for successful product development.

In agile organisations, cross-functional collaboration increases communication frequency and creates communication pathways improving relationships between different functional teams to ensure resource availability for projects (Randolph & Posner, 1992; Larson & Gobeli, 1988). Cross-functional collaboration is having functional teams working together on projects. Cross-functional collaboration also

enables quick flow of information within an organisation allowing effective and quick decision making (Ford & Randolph, 1992; Larson & Gobeli, 1987). For agile in-house software development, cross-functional collaboration is vital for teams to deliver their short cycle commitments.

To have quick approvals for project funding and support for resources to enhance effective cross-functional collaboration, agile techniques must be applied successfully at project, project portfolio and enterprise levels (Steindl, 2005). Agile benefits at project levels are reduced risk (user and customer involvement), reduced time-to-market (early and regular delivery of features), and increased productivity (focus on value adding features and value creating activities). Steindl provides agile practice and benefits for project portfolio and enterprise levels at IBM. According to Steindl, agility at enterprise level provides benefits such as becoming more responsive, focused and resilient while having flexibility in considering various sourcing models for development. However, most critical is to show benefits at project level to gain organisational support for agile in-house development.

The next section provides literature on some critical software development issues requiring adaptation of agile methods to achieve organisational agility in a market-driven environment.

2.2.2 General software development issues

2.2.2.1 Team effort

Software development requires group or team efforts which contribute to increases in productivity and quality of features (Katzenbach & Douglas, 1993). Software development teams consist of at least two people working towards a common goal requiring some form of dependency among group members (Dyer, 1984). However, in agile in-house development there is a total dependency on one another requiring whole team awareness and acceptance of tasks, involvement, collective effort, and information radiators and noticeable progress (Whitworth & Biddle, 2007). Whitworth & Biddle's grounded theory research based on interviews of 22 individuals working in different agile teams discovered pitfalls with the agile philosophy of team effort, such as individuals feeling stressed or socially inactive after a day's work, feeling burnout from increased daily contact with the same team members and from increased immersion in

the same project activities, inability for certain individuals to properly integrate into agile teams, stress faced by individuals when transitioning into or out of the unique culture of an agile team and becoming isolated in an organisation, and practices revolving solely around developer activities.
Most important is a solid agile culture and adopting an appropriate hiring method to select a highly flexible and capable workforce.

Effective team work is dependent upon the team's communication, coordination, member contribution, mutual support, effort and cohesion (Hoegl & Gemuenden, 2001). These team features are also critical for agile in-house development. However, another critical factor for agile in-house development is reflective practice helping to improve team productivity, quality, and collaboration (Lamoreux, 2005). Hence, for agile in-house development productivity and rigor (quality) are both important. Lamoreux provides an experience report of an agile development team at Medtronic where reflection was one of most challenging practices to adopt. Initially, they had mixed results where some reflections turned into gripe sessions and some were short but a 'little too sweet'. Lamoreux identifies some key roadblocks for effective reflection sessions such as inviting managers and other 'chickens' (non-team members as defined in the Scrum method), doing reflections with the entire team rather than in sub-teams, repeating meeting formats, expecting a list of action items, allowing team members to choose not to attend, giving up after one or two tries and holding the meeting without a leader. In agile in-house development, reflection meetings are critical and through adaptation provide opportunities for all to learn and become better teams.

To function effectively, software development teams must create a sense of belonging and friendship and have team-building obligations such as meeting commitments, participation in team activities, accepting and performing a team role, establishing and striving to meet team goals and building and maintaining the team (Humphrey, 2000). These are also critical for agile software development teams. For agile in-house development, self-organising teams are formed for them to function effectively (Karhatsu et al., 2010). However, important team practices for self-organising agile teams are having a collective autonomy to negotiate for mutually acceptable agreements and sharing of tasks to swiftly deliver short development cycle commitments.

Heil (1999) describes a model that integrates team effort, communication and the software development process. It is a cyclic model, where the process and communication affect each other, while teamwork improves progress and strengthens communication. In agile development, teamwork is also critical to foster knowledge creation to improve the development process (Kahkonen, 2004). According to Kahkonen, large organisations' agile in-house development consisting of multiple teams requires cross-team workshops to solve problems in communication and coordination. Kahkonen's study is based on three agile software development methods developed at Nokia (RaPiD7, Integration Camp and SEED) which are successfully being deployed in multi-team organisations.

Sawyer (2004) describes three generic types of software development teams; sequential, group, and network. The sequential model is driven by a process where social interactions are controlled and driven by the routines. The group model is also process driven but development is iterative, based on collective skills and driven by social interactions with group rules and behaviours to help resolve conflicts. The network model emphasises the product where the process used is secondary and is based on the belief that a good product comes from having appropriate individuals. For agile in-house development, teams need to be based on the network model facilitating developers working effectively with non-developers to swiftly deliver market-driven products.

2.2.2.2 Skill and task coordination

Software development teams possibly will not have the same members for different development projects. This is due to different skills that are required at different times in projects and people are put in different projects for different durations as need arises for a particular skill (McGuire, 1986). While skills coordination facilitates a productive effort, in agile development environments moving individuals from one team to another also leads to knowledge sharing (Qureshi & Kashif, 2009).

Team expertise is the aggregation of individual skill, knowledge, and experience, which must be coordinated and managed properly within the team (Faraj & Sproull, 2000). Faraj & Sproull's study describes two different coordination processes; administrative coordination and expertise coordination. Administrative coordination is required to assign tasks, allocate resources, and to integrate output whereas for the complex non-

routine intellectual tasks expertise coordination is required enabling teams to know where the expertise is located. For agile in-house development, both administrative and expertise coordination are expected to be collectively coordinated by self-organising agile teams but large agile development environments require leadership roles (Moore, 2009). Hence, the size of IS department is an important factor for adapting sub-teams, and administrative and expertise coordination on agile projects. Moore provides an experience report of a large-scale global development effort with 300 people. According to Moore, leadership in a large agile in-house development must maximise team velocity through creating flexible sub-teams, drive team and project success, and balance team and individual needs. Balancing team and individual needs is a critical leadership behaviour showing management interest in supporting individual career goals and aspirations.

2.2.2.3 Performance factors

The participatory approach to involving developers in the decision making process has a positive influence on their work performance. Miller & Monge (1986) used the cognitive, affective, and contingency models to examine the effects of participation in decision making. The cognitive model suggests a positive impact on performance where workers make effective decisions when they are part of the process and have access to appropriate information. However, the affective model suggests that the satisfaction of being part of the decision making process alone improves the performance. While the contingency model suggests that no single method of participation is suitable for all individuals, it has more impact on satisfaction than productivity. In agile development environments, the cognitive model is important for high team performance in productivity and quality.

For agile in-house development, decision making process is adapted so that the development teams are responsible for making operational level decisions (Moe & Aurum, 2008). Moe & Aurum's case study research highlights that in agile development role specialisation is a barrier for decision making on the operation level and to aligning decisions with management levels. Daily meetings are important for preventing decision-hijacking and removing hold-ups to make it easier for developers to participate in the decision making process. While Moe & Aurum's issues are all critical, having development practices to create multi-skilled developers must be a key target in

any agile development environment developing universal behaviour for collective decision making.

A study done by Chung & Guinan (1994) showed that team size and team member experience affect participation. Their study reveals that the experience of members in small teams impacts performance and participation but has no major impact on larger teams. For agile in-house development, small team size and team member experience are critical for effective interaction and collaborations for high productivity and quality gains.

According to Melnik & Maurer (2004), verbal face-to-face interaction facilitates a higher team velocity. Highly experienced developers provide learning and implementation speed in agile development environments (Savolainen, Kuusela, & Vilavaara, 2010). Savolainen et al., provide an experience report based on two agile transitions highlighting that different teams with different competencies tend to decrease speed while increasing design efforts and management involvement. Hence, experience is critical developer-embodied factor for agile in-house development to achieve high productivity.

Hiring new developers is another important issue that has to be appropriately dealt with. The newcomers must be quick learners adapting to the team culture and contributing to the team's performance expectations. Chen's (2005) study reveals that to ensure early performance, newcomers should be given challenging goals, provided with adequate social support, and empowered with appropriate and required resources. If it takes time to bring newcomers up to speed, business opportunities will be missed if the team cannot increase the rate at which they complete functionalities (Opelt, 2008). Opelt provides an experience report where an XP team successfully doubled with 16 developers in three weeks to deliver additional functionality requested by customers. According to Opelt, co-location, pair programming, test driven development, system metaphor and simple design are key practices to support newcomers. While not all agile in-house development teams adopt these XP practices, task sharing is critical for swift newcomer integration maximising team productivity and enforcing rigor to achieve quality gains.

A study done by Hoegl & Proserpio (2004) provides useful insights on team members' proximity and its effects on collaboration for achieving a higher team performance.

Close proximity enables a higher number of spur-of-the-moment meetings, a flow of rich information, and better coordination of tasks and contributions, while providing mutual support and being the catalyst for high individual effort. For agile development, close proximity is achieved through co-location of developers and on-site customers (Law & Ho, 2004). For agile in-house development, co-location is important to enhance productivity and quality.

Next, literature on agile methods is evaluated and integrated with the overt factor component of Fitzgerald's adaptation framework.

2. 3 Overt Factors

2.3.1 Agile methods

This section provides insights into some of the key agile software development practices regardless of the adopted agile method. Their adaptation is crucial in achieving development agility.

2.3.1.1 Multiple short cycles

Structured methods are mostly stage-based with sequential or linear phases (Schach, 2007). Such development approaches have a single development cycle with a single product release while most development teams expect stable requirements (Boehm & Turner, 2004). Accommodating late requests for changes requires a revision of budget and delivery schedules and almost doubles the cost of projects and the time spent on them (Baird, 2003). However, agile approaches to software development have multiple short development cycles and multiple product releases (Highsmith, 2004). Short development cycles are multiple small phases for implementation in a project, leading to a release.

Short development cycles are a key overt factor for project management using agile methods allowing for clear visibility of a project plan and its implementation schedule. Short development cycles (iterations/sprints) not only enable delivery of features quickly and regularly (Ding & Ravichandran, 2000) but also enable requests for changes during projects (Cao & Ramesh, 2007). Emerging requirements are easily accommodated in the next iteration (Kalliney, 2009). Iterations enable a quick start and stop for projects (Subramanian, Klein, Jiang, & Chan, 2009). Iterations also enable

meeting the market's current needs (Khetan & Fowler, 1995). Acceptance tests done during iterations confirm the business value of features (Cohn, 2004). Agile development makes project status visible upfront (Morien, 2005). The teams are aware of where they are in a project by tracking the iterations (Zhi-gen, Quan, & Xi, 2009).

However, for agile teams to move smoothly from one cycle to another domain support is critical to planning for each short cycle, decomposing features into tasks or stories with estimates (Smits, 2007). According to Smits, agile teams must adapt to prepare for the next cycle so that short cycles overlap and are continuous without breaks, features in the product backlog are pre-staged and the top of the product backlog is ready to be included into next cycle. Another challenge highlighted by Smits is that the team must set aside time to prototype at least one cycle ahead. Hence, short development cycles, as an overt factor for project management, also enable management of risks associated with product requirements.

Overcoming architecture challenges is also critical if short development cycles are to provide strategic products (Madison, 2010). Madison provides information on challenges for agile architecture based on his experience at a large insurance company. According to Madison, while agile delivers the short term values, high value architecture is only achieved through long-term objectives. Hence, an effective architecture evolves over a long period of time requiring an early focus on combination of short development cycles. Madison proposes architectural interaction points with agile development which requires up-front architectural planning, storyboarding and backlog planning where the architect is a key stakeholder and contributes with architectural user stories, participates in short sprint cycles and working software must also include architectural documentation. Most critical is the architect's participation in short sprint cycles to achieve high quality architecture. However, agile teams will need to adapt team structures and skills to be able to share this specialist support in sub-teams.

The XP method has two week iteration cycles (Rosenberg, Stephens, & Collins-Cope, 2005). Scrum iterations are known as sprint cycles, which are one month long (Schwaber and Beedle, 2002). Crystal Clear (one of several types of Crystal methods) requires releases every three to four months which have smaller durations for their iteration cycles (Cockburn, 2002). Feature driven development has its iteration cycles

between two days to two weeks. DSDM has a time-box-approach for its iteration cycles usually from a few days to a few weeks (Abrahamsson, Warsta, Siponen, & Ronkainen, 2003). However, most important is the team ability to adapt to the shortest possible development cycles for more frequent delivery of features providing strategic product benefits through this overt factor.

2.3.1.2 Product backlogs

The product backlog is an important overt factor for reduction of variety and complexity. The agile analysis phase establishes low level requirements (stories or tasks) to create a product backlog (Lehto & Rautiainen, 2009). Product backlogs have prioritised requirements ready for implementation in short development cycles (Schwaber & Beedle, 2002). The activity of creating a product backlog is usually time-boxed to ensure implementation starts in the first short development cycle (Rosenberg, Stephens, & Collins-Cope, 2005). The product backlogs enable product goals to be met (Raatikainen, Rautiainen, Myllärniemi, & Männistö, 2008). Hence, the product backlog as an overt factor is critical to splitting low level requirements to be implemented in short development cycles.

The activity to create the product backlog is held prior to the start of the first iteration and during projects providing the opportunity to discover emerging requirements (Highsmith, 2004). Therefore, a product backlog is not a static list of prioritized features (Rising & Janoff, 2000). However, swift access to product managers, field staff or clients is extremely important to validate the market value of any emerging requirements.

Features can be removed or the priority of features to be implemented can be changed by the owner or by the product managers (Sutherland, 2005). The change management of product backlogs does not require any formal process, expecting that a product manager has good business reasons for making such a decision (Engum, Racheva, & Daneva, 2009).

Product backlogs are based on business value where the requirements are elicited and analysed on a “user-oriented basis” and efficiently prioritized (Keil & Kuhrmann, 2006). Requirements in product backlogs are in their smallest form (Kalliney, 2009). These are usually referred to as user-stories with XP, Scrum and Crystal Clear methods;

functional and non-functional requirements with DSDM; and features with adaptive software development and feature driven development (Highsmith, 2004). Most important are the developer abilities, experiences and support to break requirements into their smallest form, delivering them in a single iteration cycle.

The software engineers' involvement in product backlog creation enables them to gain a better understanding of features while the quality assurance engineers get to know which features to exhaustively test (Dinakar, 2009). With agile methods, priority setting is done by the product managers or the backlog owners but scheduling for implementation with the feature driven development method is considered to be a technical decision and done collectively with engineers (Highsmith 2002). With this product backlog overt factor, backlog reviews on a regular basis with the entire team are essential to provide them with insights into upcoming implementation challenges.

An experience report by Kalliney (2009) highlights the challenges of moving from agile development with a team backlog to enterprise product management with a company-wide backlog. They had to deal with issues of implementing the product vision by product owners because of poor communication by business owners who are responsible for setting product strategy and vision. Due to the small nature of the user stories in team backlogs, the business owners and portfolio managers had difficulty validating the implemented product vision. According to Kalliney, they had to adapt into two separate departments, product strategy and development, to deal with cross-team risks and dependencies. The product strategy group now consist of business owners, portfolio managers, product managers, and product owners working together to develop product visions and to prioritise high level requirements. Most important is a single high-level product backlog to provide a clear company-wide visibility of releases.

Lehto & Rautiainen (2009) also provide an experience report of an organisation (with 300 employees of which half are software engineers) in agile transition highlighting major challenges in implementing product backlogs. According to Lehto & Rautiainen, development teams could not get swift clarification on product backlogs due to product owners' other commitments and multiple owners requesting work from a single team. Lehto & Rautiainen also discovered that development teams abandoned task-planning making it difficult to link the backlog items with high-level plans and to measure implementation progress. According to Lehto & Rautiainen feedback loop and

prioritisation practices are vital to product backlog practice. Most essential with this product backlog overt factor is adapting team structures to have cross-functional effort and support for achieving effective coordination and implementation flexibility.

2.3.1.3 Frequent integration and builds

Frequent integration and builds are important practices supporting the agile overt factor for short development cycles. Agile success is significantly dependant on automatic development and testing environments for continuous and regular delivery of working software (Ferreira & Cohen, 2008). The two important tools are the integration and build systems (Bowyer & Hughes, 2006). Integration systems enable software engineers to submit their latest changes to their source code into a central source control system (Beck, 1999a). A build system on a dedicated server with all the latest code runs all the unit tests (Germain & Robillard, 2005). These two systems enable errors to be detected at a very early stage of development (Rosenberg et al., 2005). Effectiveness of frequent integration and builds is dependant upon engineer support for implementing appropriate unit tests.

Continuous integration and regular builds for iterations enable implemented features to be tested and packaged with others (Fruhling & Vreede, 2006). The agile teams run their automatic build systems as frequently as possible (Beck, 1999b). This continuous integration practice enables agile development to achieve its value of measuring progress based on frequently delivering working software in short development cycles (Hansson, Dittrich, Gustafsson, & Zarnak, 2006). However, the technical ability and support to adapt the continuous integration and build systems are crucial to shorten the time it takes to build as the automated test cases grow.

Some of the popular commercial licensed integration and build systems tools are BuildForge, TeamCity and AnthhillPro (Kim, Park, Yun, & Lee, 2008). However, agile teams tend to use more of the open source tools such as CruiseControl, Tinderbox, and Buildbot (Brooks, 2008). The important thing with integration and build systems tools is that they must provide flexibility for agile teams to adapt their features to meet the local conditions and support the dynamic products.

Downs, Hosking, & Plimmer's (2010) study discovered that some developers felt frustrated with the agile practice of 'assigning responsibility' to an individual for fixing

a broken build rather than to the individual breaking the build. According to Downs et al., others felt good natured teasing when they broke builds while one felt that team pride and professionalism was adversely affected by broken builds. However, regular meetings are important to decide how best to adapt to fix broken builds to maintain good team sprit and dynamics.

2.3.1.4 Task estimation

Task estimation is an important activity supporting product backlog overt factor for reduction of variety and complexity. Task estimations are done collectively by engineers, product managers and project managers contributing to provide estimates for requirements when creating product backlogs (Smits, 2006). During the meetings they analyse and split requirements into smaller estimated tasks (Dinakar, 2009). However, for getting reliable estimates, engineer input is critical since they have the necessary technical expertise to identify and understand the complexities associated with each requirement.

When a requirement is not well understood by engineers, a fixed period of time (known as a spike) is allocated to further investigate the requirement during an iteration (Cohn, 2004). It may also involve prototyping the requirements before an estimate can be provided (Cao & Ramesh, 2008). Once sufficient information is compiled, it is presented so that everyone can understand the requirements better. Supportive practices for task estimation are vital for making reliable estimates so that teams can efficiently transit to the next iteration and provide accurate implementation schedules organisation-wide, making product backlog a strategic overt factor.

Trust and a common sense approach enable teams and management to establish and agree upon reliable estimates, which the engineering teams commit to and deliver by the end of the iteration (Highsmith, 2002). However, effective individual contributions on proposed estimates are extremely important for getting reliable estimates.

According to Haugen (2006), the unstructured agile estimation practice is affected by company politics, group pressure, anchoring, and dominant personalities. His study investigated the semi-structured Planning Poker and compared it with unstructured group estimation for four releases (2 non-structured and 2 poker planning) by an agile software development team. Haugen’s result indicates that planning poker practice

provides more reliable estimates if the team had experienced similar tasks but increases extremes in estimates because this practice generates more group discussion. Unreliable planning poker estimates for unfamiliar tasks indicate that agile teams must continuously adapt to become better at producing reliable estimates to allow for effective backlog planning.

2.3.1.5 Micro planning

Micro-plans are important activities supporting the agile overt factors for project management, and reduction of variety and complexity. Agile methods do not encourage a detailed macro-plan for an entire project due to the understanding that the business environment is volatile and stable requirements can never be guaranteed (Highsmith & Cockburn, 2001). However, it is critical to minimise the impact of micro-planning meetings on iterations by having focused meetings or adapting team participation.

Agile planning involves meetings attended by customers, engineers and other stakeholders (Liu, Erdogmus, & Maurer, 2005). Agile planning invites important stakeholders and is based on negotiations and commitments (Schwaber & Beedle, 2002). Effective participation in agile planning is dependant upon showing the mutual benefits for all the stakeholders and by having a shared vision for products organisation-wide.

The agile approach promotes several micro-project plans for software product development. There are five different levels of agile product planning; product vision, roadmap, release, iteration, and daily plans (Smits, 2006). The levels of detail and time spent planning would vary for each of the five different plans. For example, the duration for the daily planning meeting is only15 minutes (Dinakar, 2009). To empower whole team participation in iteration planning is critical.

The product vision plan is an important overt factor for improving visibility organisation-wide of new features to be implemented. The product vision plan is compiled by the owner of a proposed product, usually by the product manager (Hitt, Ireland, & Hoskisson, 1997). The product plan is formulated and presented to create awareness and interest in the organisation for new features (Smits, 2006). However, it is important that the product manager consults widely to collect, compile and test information to identify and determine new features. A vision plan has information on

target users or customers, a few key features or high-level requirements, product benefits, reasons for clients to buy it, how it differs from other competing products, and how it will bring competitive advantage (Vähäniitty & Rautiainen, 2008). Most important with vision plans is that the high-level requirements give visibility to each project's scope.

Product roadmap planning is an important overt factor for reduction of variety and complexity of feature releases using agile methods. Product roadmap planning follows product vision planning. Product roadmap plan shows features which will be in different releases of a software product (Bagnall, Rayward-Smith, & Whittley, 2001). The product roadmap plan identifies the number and dates of each release of a product with a list of requirements with rough estimates to be implemented as features for a release (Saliu & Ruhe, 2005). Critical for developing an effective roadmap plan is input by the development team on their capacity and capability to deliver on those dates.

Release planning is knowledge intensive, and requires stakeholders to participate (Lindgren, Wall, Land, & Norstrom, 2008). Without a good release plan, crucial features will not be made available at the right time resulting in unfulfilled customers, time and budget overruns, and decreasing competitiveness in the marketplace (Ngo-The & Ruhe, 2009). Vital for developing an effective release plan is creating a well prioritised product backlog maximising its implementation in each iteration cycle.

With agile development, the release plan provides benefits such as helping to build customer confidence, maintaining the existing customer base, driving new sales, building sales department confidence, telling the development team exactly what they have to build and why, and it is a single source of truth showing how the current effort moves the product in alignment with the business strategy (Wilby, 2009). Release plan describes which features will be delivered in upcoming release.

With non-agile development, the exact content of a release is determined by the release plan whereas with agile development the exact content for a release is determined during the projects allowing changes by customers (Lindgren, Wall, Land, & Norstrom, 2008). Implementation flexibility is extremely important for agile teams knowing that requirements cannot be frozen. A formal process is used with non-agile development to determine the content of a release based on parameters such as customer value and cost of development (Saliu & Ruhe, 2005).

The iteration plan is an important overt factor for improving visibility organisation-wide of new features to be implemented during projects. The iteration plan identifies the specific requirements that will be implemented in iterations (Liu, Erdogmus, & Maurer, 2005). Iteration or sprint plan provides a subset of estimated and analysed features that are to be implemented Klein & Canditt (2008). Iteration planning happens a few days before the iteration begins (Kinoshita, 2008). Iteration plans enable the team to determine more accurately the number of hours an individual will be available for the next iteration (Engum, Racheva, & Daneva, 2009). Iteration plans account for who would be available and the hours that individuals will be away from the team work area attending meetings, training, or carrying out other team commitments, including personal appointments and annual leave (Rautiainen, Vuornos, & Lassenius, 2003).

Iteration planning meetings are joint meetings between the product backlog owner and the agile development team (Abrahamsson, Salo, Ronkainen, & Warsta, 2002). During iteration planning meetings complex requirements are split into smaller tasks for implementation and estimates are re-negotiated (Frank & Hartel, 2009). The agile team has a better knowledge and information to make commitments to deliver a certain number of features by the end of the iteration (Cohen & Thias, 2009). Re-negotiation in iteration planning is crucial for getting mutually acceptable estimates.

The daily stand-up or scrum meeting is an important overt factor for reduction of variety and complexity with daily feature implementations. The daily stand-up or scrum meetings attended by all the team members are extremely beneficial for collaboration, keeping everyone focused, solving problems and delivering results (Shaye, 2008). The stand-up meeting is an important planning practice, lasting no more than fifteen minutes, where each team member answers three questions; what they did since the last meeting, what obstacles they had, and what they will do before the next meeting (Schwaber & Beedle, 2002). Daily plan shows the story or task an individual will implement for the day. As a co-located agile team matures, it is vital that stand-up meetings are adapted to keep providing good value for the team.

Daily scrum meetings support the team in organising itself for the day where the members update their task status and take up new tasks (Rubart & Freykamp, 2009). Daily scrum meetings also encourage team members to communicate more outside the

meetings and provide all stakeholders with a good overview of the project situation (Paasivaara, Durasiewicz, & Lassenius, 2009).

An experience report provided by Greening (2010) shows how the ideas from daily scrum meetings were adapted to have an enterprise weekly scrum meeting for managing 25 scrum teams in a software engineering organisation. Adapting to a weekly stand up, lasting for an hour, provides a forum for product managers, scrum masters, user experience manager and technical leads to discuss progress and short-term plans, identify difficulties, and find collaborative solutions with the direction of product management, VP of engineering and enterprise scrum master. According to Greening enterprise scrum enabled them to effectively start, postpone or cancel whole projects. However, enterprise scrum created release issues for the operation group due to a large number of releases at once and fluctuating work demand requiring user-experience testing, branding, customer support and user interface groups to prioritise their work (Greening, 2010). This example suggests that adaptability and flexibility of functional units in agile development environment is extremely critical to achieve enterprise product agility.

Results of an investigation of a software product organisation by Lehtola et al., (2005) reveal some key findings in regard to product planning such as: it strengthens the link between the business decisions and requirement engineering, it is a tool for communicating ideas to various stakeholders by product managers, and accurate release decisions are made when product plans are made for shorter time periods. However, Lehtola et al. (2005) also discovered that while developers provide estimates, their viewpoints are less emphasised, product plans are not tied with development resources, and these plans immediately get outdated. For agile development, it is critical that product plans are prioritised, where each high-level plan represents a particular functionality which can easily be implemented.

Software product plans are usually based on a single market release making it difficult to reduce the scope, to respond to changes in market condition or to launch a new product (Nejmeh & Thomas, 2002). According to Nejmeh & Thomas, agile methods provide the flexibility with planning product functionalities and releases but these methods do not address: (a) how the development team should interact with key stakeholders and product managers, (b) how a product manager and development team

can produce consistent feature requirements and priorities that will satisfy multiple customers, (c) techniques for product managers to balance the conflicting demands of multiple customers, and (d) techniques for product managers to determine the value of proposed features. The first two are critical agile issues where it requires permanent stakeholder support and must also provide technical perspectives on high-level requirements allowing product managers to accurately determine the business value of high-level requirements

2.3.1.6 On-site customer role

The on-site customer is an important role supporting the agile overt factor for project management to reduce risks. Agile development requires the business function to work with the development function on a daily basis where the business function provides individuals for on-site customer roles (Highsmith, 2002). On-site customer is a business individual providing support at development level. The on-site customer is responsible for all the business decisions at development level (Martin, Biddle, & Noble, 2003). The on-site customer provides a clear understanding of what exactly is to be developed through being co-located with the engineers and as part of the development teams (Beck, 1999a). The on-site customer provides instant face-to-face communication with the engineers to discuss issues during projects (Williams, 2003). The on-site customer participates in all the phases of agile development (Wang, Wu, & Zhao, 2008). Most important is permanent co-location of the on-site customer with the development team to provide continuous support during projects.

An ideal on-site customer will have excellent understanding of the product domain, how the software provides business value, regular delivery of software, and prioritising the functionality including accepting the responsibility for success or failure of projects while representing the diverse business stakeholders (Martin, Biddle, & Noble, 2003). Extensive domain knowledge of products is most critical for swiftly solving developer issues including up-to-date market knowledge to prioritise the backlog to provide a clear implementation guide. This co-located on-site customers' extensive domain and market knowledge ensure project management a strategic overt factor with agile methods.

According to Martin, Biddle, & Noble (2004), on-site customers have a pressured and stressful role. Martin et al's., interpretative case study research involving three XP

projects from different companies discovered that while on-site customer practices achieved excellent results, the role appeared to be unsustainable for long and high pressure XP projects. According to Martin et al., on-site customers as an interface between business and development team perform multiple business, administrative and development tasks, often working twice as long as any other individuals in a development team. However, learning practices are vital to acquire cross-functional knowledge and skills in agile team setups ensuring effective sharing of tasks and responsibilities for sustainable agile development.

There are different types of on-site customer roles such as such as geek interpreter, technical liaison, political advisor, acceptance tester, negotiator, diplomat, super-secretary, customer coach, user interface designer, technical writer and diplomat (Martin, Noble, & Biddle, 2009). Martin et al's., grounded study collected data through interviews and observation from eleven XP projects in New Zealand, USA and Europe. They discovered that XP projects require several on-site customer roles where one person could have multiple on-site customer roles and multiple individuals could also combine to carry out a single on-site customer role. The most critical for a market-driven product development environment are negotiator, diplomat, super-secretary, and customer coach roles requiring a business individual adapting with their skills to work as on-site customer. The coaching skills are as critical as others since task-sharing is the means for imparting knowledge and skills for engineers to step into on-site customer roles.

2.3.1.7 Generalist skills

The agile approach requires more general skill sets whereas the more formal approaches firmly favour specialist skill sets (McConnell, 2004). Generalist skills are an important overt factor with agile methods. Generalists are flexible individuals having multiple development and business skills in agile software development teams. As such, individuals are required to continuously train and develop new skills (Bottani, 2009). The generalist roles are more productive, achieve more frequent delivery commitments, reduce delivery time, produce better quality products and deliver broader product features (Hopp & Van Oyen, 2004). Frequent delivery and quality are the two most critical measurements of agile team productivity.

While agile teams have generalist roles, individuals in these roles are expected to be experienced (Cockburn, 2002). The XP method has the following team roles; programmer, customer, tester, tracker, coach, consultant, manager (Beck, 1999a). The programmer, customer and tester roles can be made more generalist by incorporating tracker and coaching responsibilities. These roles can be further enhanced into more generalist roles by assigning project management responsibilities making the manager role more proactive with actual development work.

The scrum method has scrum master, product owner, scrum team (developers), customer, and management in team roles (Schwaber & Beedle, 2002). The product owner and developer roles can also become more generalist by incorporating scrum master responsibilities making it a strategic overt factor. This method has more business role representation (product owner, customer, and management) at development level than the XP method enhancing the chance for project success.

Crystal clear methods have several roles such as sponsor, senior designer-programmer, designer-programmer (designer-programmer has sub roles such as business class designer, programmer, software documenter and unit tester), UI designer, database designer, architect, usage expert, business analyst/designer, technical facilitator, writer and tester (Cockburn, 2002; Cockburn, 1998). This group of methods have more technical roles requiring specialist skills. However, the designer-programmer role can be made into a more generalist role and a strategic overt factor by incorporating UI design and database design responsibilities. The senior designer-programmer roles can also undertake chief architect and technical facilitator roles making them more generalist.

The feature driven development method also has several roles such as project manager, chief architect, development manager, chief programmer, class owner (programmers), domain expert, domain manager, release manager, programming language lawyer/guru, build engineer, toolsmith, system administrator, tester and technical writers (Palmer & Felsing, 2002). With this method, the programming role can also be made into more generalist role by incorporating project management, build engineer, toolsmith, and system administrator responsibilities. The chief programmer role can also undertake chief architect responsibilities. This method, like scrum, has business representation (domain expert, domain manager, and release manager) at development level.

The DSDM method has developer, senior developer, technical coordinator, ambassador user, adviser user, product visionary and executive sponsor roles in the development team (Stapleton, 1997). Individuals are required to perform in more than one team role (Palmer & Felsing, 2002; Cockburn, 1998). This method like the scrum and feature driven development methods have business representation at development level (ambassador user, adviser user, product visionary and executive sponsor).

Despite their differences, agile methods have business roles as part of the product development teams. The developer or programmer roles can be enhanced into more generalist roles by adapting with the responsibilities of the other technical roles and project management tasks. Business role support for learning domain knowledge and skills adaptation are two key agile practices which are critical to provide developers with general skill sets.

With agile methods cross-training and job sharing enable individuals to develop multi-skills to take another role in their teams (Hopp & Van Oyen, 2004). Pair programming is a key practice for knowledge transfer to develop multi-skilled engineers but pairs must be rotated to ensure that everyone in a team pairs with everyone else (Goebel, 2009). Task-sharing, mentoring and coaching is critical in agile development environments to develop a general skill base if teams do not adopt pair programming.

2.3.1.8 Test driven development (TDD)

Test driven development (TDD) supports the agile methods' overt factor for short development cycles ensuring their quality requirements are being meet. TDD as an XP method practice has gained wide acceptance with other agile methods (Beck, 2001). TDD was first used in the 1960s by NASA for a software development project (Larman & Basili, 2003). However, with structured methods, a single development cycle meant that this practice was infrequently used (Larman & Basili, 2003). TDD is as an evolutionary approach to development, involving implementing a test before implementing and refactoring the code.

TTD is an important practice supporting short development cycles and a zero-defect delivery policy for each iteration commitment (Williams, Kudrjavets, & Nagappan, 2009; Rendell, 2008). TTD requires writing a few lines of automated unit tests in parallel with a few lines of code (George & Williams, 2003). First, running the test

cases to fail and then implementing the code to allow the unit test to pass (Bhat & Nagappan, 2006). TDD also involves refactoring the tested code and regularly re-running all the test cases to ensure the new code did not break when integrated with the code base (Muller & Hagner, 2002).

The main benefit of TDD is improvement in product quality when compared to not using TDD (Bhat & Nagappan, 2006). However, this practice impacts developers implementation velocity and developers also require training and support to write useful unit tests. The fine granularity of writing tests before coding provides continuous feedback for the developer (Kaufmann & Janzen, 2003). The collection of automated tests is a valuable asset for regression testing (Williams, Maximilien, & Vouk, 2003). To enhance TDD support for short development cycles as an overt factor, frequent adaptation of the build system is critical to maintain an acceptable build time since the ongoing accumulation of automated tests will impact the frequency and time for automated builds.

The low-level designs are test-driven with classes, methods and interfaces requiring a minimal upfront design effort (Maximilien & Williams, 2003). The test suite also describes the dynamic aspects of the system by mapping the expected values returned by each method (Erdogmus, Morisio, & Torchiano, 2005). However, creation of an effective automated test suite is dependent upon the product related information being readily made available in the development environment.

The TDD practice provides a greater predictability of development performance helping to estimate project cost (Canfora, Cimitile, Garcia, Piattini, & Visaggio, 2006). However, significant reduction in defect rates with TDD is critical if short development cycles are to provide strategic products. The automated tests run at least daily enabling daily integration which serves as the heartbeat of a project and minimises the risks, reduces manual testing (the defects are identified quickly) and the source of the defect is more easily determined (Maximilien & Williams, 2003). The TDD practice also suits easy changes to the product driven by business needs (Sukkarieh & Kamal, 2009). TDD allows development driven by the requirements rather than losing the requirements perspective as the development progresses (Park & Maurer, 2008).

Customer support is critical for successful use of TDD practice to determine a relevant set of use cases and user stories without explosive growth of them (Fraser & Mattu,

2007). Fraser & Mattu (2007) provide an experience report on using TDD practice. According to Fraser & Mattu, the involvement of customers and domain experts is critical for an overall architecture design before code development, to avoid costly upfront-design omissions, and for planning to determine the ordering of implementation of user stories minimising problems related to concurrency and real time requirements. To enhance TDD support for short development cycles as an overt factor, user stories with no business value must not be implemented avoiding a negative impact on the build speed due to increase in automated test cases.

2.3.1.9 Tool support

Automation in agile development environment supports the overt factor of short development cycles. Agility is achieved through intelligent technologies automatically performing many manual tasks to have short development cycles (Baldwin & Chung, 1995). Automation is vital to support mundane and repetitive tasks to deliver features frequently and regularly through short development cycles.

Tool support is regarded by the software engineering discipline as necessary to enhance the productivity of individuals in the development team and to achieve a better quality product (Schach, 2004). Other reasons are that it helps faster implementation, improves the skill sets of individuals, and enhances portability and compatibility of new systems with others through standard and improved management of the systems development process (Hoffer, George, & Valacich, 2004).

Integrated Development Environments (IDE) with version control systems, integration tools, build systems, coding tools, testing tools, communication tools, modelling tools, and reporting tools are required for fast paced product development environments (Leithiser & Hamilton, 2008; Avision & Fitzerald, 1995). IDEs are extremely important in agile software development for automation of development tasks giving development speed to support short development cycles as a strategic overt factor.

Agile teams require an automated unit testing environment for test driven development (Williams, Kudrjavets, & Nagappan, 2009; Tan & Edwards, 2008). Automating acceptance testing has also become a common practice with agile development teams (Martin, 2005). Agile development has tool support for these two types of automated testing for product development such as XUnit and FIT (Hanssen & Haugset, 2009;

Alwardt, Mikeska, Pandorf, & Tarpley, 2009; Do, Rothermel, & Kinneer, 2004; Holmes & Kellogg, 2006). Another tool support for agile development is for product backlog planning such as XPlanner (Engum, Racheva, & Daneva, 2009). Tool support for unit and acceptance tests is critical in agile development to instantly determine if the individual unit of source code is fit to use as it is being implemented and to quickly determine the completeness of a user story.

2.3.1.10 Pair programming

Pair programming is a key overt factor for transfer of knowledge. Pair programming is an XP method practice but it is widely adopted with other agile methods to replace solo code development. Pair programming is an agile practice in which two engineers work together at one work station. While pair programming facilitates teamwork through collaboration for code development, it also provides benefits such as job rotation and succession against personnel turnover and skills transfer for knowledge management (Lui & Chan, 2008).

Pair programming balances the increase in personnel cost with implementation speed and reduction of defects (Padberg & Muller, 2003). Padberg & Muller's study is based on a model derived by integrating the process metrics, product metrics and process metrics. However, pair effort reduces the number of features which can be simultaneously implemented.

Pair programming has a positive impact on learning, scheduling of tasks, acquaintance culture and team cohesion (Vanhanen & Lassenius, 2007). Pair programming improves the discipline of the individual engineers in the team, leads to better code, helps to create better solutions, improves team morale, and raises working knowledge of the code base by the engineers (Williams, McDowell, Nagappan, Fernald, & Werner, 2003; Nosek, 1998). These benefits of pair programming lead to improvement in the quality of the software (DeClue, 2003; Hanks, McDowell, Draper, & Krnjajic, 2004; McDowell, Werner, Bullock, & Fernald, 2002). However, effective collaboration is a key factor for gaining these development benefits from pair programming practice enhancing transfer of knowledge as a critical overt factor.

Pair programming also leads to an improvement in the satisfaction and morale of software team members (Hanks et al., 2004; McDowell, L, Bullock, & Fernald, 2003;

Mendes, AL-Fakhri, & Luxton-Reilly, 2005) since individuals working in pairs find it more enjoyable compared to going solo. The feel good factor helps to increase the productivity of the entire team working on large projects (Sison, 2009; Lui, Chan, & Nosek, 2008). However, a multiple case study research by Hulkko & Abrahamsson (2005) discovered that pair programming does not necessarily provide extensive quality benefits and does not result in consistent superior productivity when compared with solo programming. They also discovered that pair programming was suitable for learning in the beginning of a project, solving problems and ways of doing complex tasks and finding mistakes from simple code. However, effective pair discussion and support with on-site customers and agile testers are vital factors contributing to productivity and quality rather than just pair programming.

The basic concept which drives pair programming is that two software engineers take part in a joint development effort at one terminal through driver and navigator roles (Williams & Kessler, 2002). This joint effort raises the level of cooperation and accountability for code design and implementation. However, ethnographic observations of two software development teams by Chong & Hurlbutt (2007) highlight the lack of division of labour where both developers took driver and navigator role simultaneously. According to Chong & Hurlbutt, pair programmers do not think at different levels of abstraction instead they move together considering and discussing issues at the same strategic level requiring dual keyboards to facilitate rapid switching of the keyboard. They also discovered that individuals disliked pairing with less expert individuals. Having individuals capable of quick learning is critical to the success of agile teams.

2.3.1.11 Open workspace

Openwork space is an important agile practice supporting facilitation of intercommunication among developers as a key overt factor. Agile teams require a very large open space to have the entire team to be co-located with other important individuals whom they frequently interact with on a daily basis (Law & Ho, 2004). Law & Ho provide an experience report of the agile software development environment at TransCanda (a company that transports natural gas in Canada and America), with evolution of co-location practice from a group of “honey comb” cubicles in 2000 to a conference room in 2001 and finally in open environment with a co-located on-site

customer in 2004. According to Law & Ho, co-location enables communication but the essence of communication is dependent upon the team and also, the open environment may not be favourable to everyone because of lack of privacy and distractions. Hence, adaptation of open workspace is critical to have separate personal workspace, team workspace, and meeting room minimising team distraction. Openwork space is an important agile practice supporting facilitation of intercommunication among developers as a key overt factor. Agile teams require a very large open space to have the entire team to be co-located with other important individuals whom they frequently interact with on a daily basis (Law & Ho, 2004). Law & Ho provide an experience report of the agile software development environment at TransCanda (a company that transports natural gas in Canada and America), with evolution of co-location practice from a group of "honey comb" cubicles in 2000 to a conference room in 2001 and finally in open environment with a co-located on-site customer in 2004. According to Law & Ho, co-location enables communication but the essence of communication is dependent upon the team and also, the open environment may not be favourable to everyone because of lack of privacy and distractions. Hence, adaptation of open workspace is critical to have separate personal workspace, team workspace, and meeting room minimising team distraction.

The soft skills are a very important developer embodied-factor for agile development to facilitate face-to-face interaction in teams (Owusu, 1999). Without widespread soft skills, forming effective sub-teams on projects for agile development would be difficult. Communication skills are extremely important soft skills in agile product development environments driven by team work, empowerment, cross-functional collaborations and external support requiring spontaneous interaction (Dyer and Shafer, 2003). Communication skills facilitate agility capabilities such as responsive behaviour, development competencies, flexibility and speed (Sharifi and Zhang, 1999). These agility capabilities are critical developer-embodied factors.

In agile software development effective communication skills are not only critical to quickly learn what is to be developed but also for getting rapid feedback on what is being developed through face-to-face interaction and co-location (Paasivaara & Lassenius, 2006). Global development supporting agile in-house development also requires effective communication and cooperation throughout projects. Global development involves having outsourcing, subcontracting, in-house development and

partnership with others. In this environment, daily communication requires supporting different agile practices through suitable communication and media practices, encouraging informal interaction, and building trust (Paasivaara & Lassenius, 2006). Informal interaction and trust are critical factors in agile software development environments to make delivery commitments.

In addition to communication skills, on-site customer is a key communication practice for agile development. However, not all agile development environments have a permanent on-site customer. Korkala, Abrahamsson, & Kyllonen's (2006) research using 4 case studies showed that with partial on-site customers in agile development environments, the use of face-to-face communication is extended along with email and telephone to manage customer-developer communication during iterations. However, their study result suggests that increased reliance on less informative communication channels result in higher defect rates. Videoconferencing is second in richness to face-to-face communication while telephone conferencing does not transmit visual clues (but provides instant feedback) and emails are more suitable for communicating well-understood issues (Korkala et al., 2006). For agile development, most critical is face-to-face interaction through co-location, providing instant access to the on-site customer.

Most importantly, effective communication in software development environment leads to job satisfaction and better performance by the employees (Javed, Maqsood, & Durrani, 2004). Orsted, (2000) identifies communication skills as an important soft skill for software engineers at Microsoft in Ireland. Orsted also identifies other soft skills such as change management, self development, stress management problem solving, drive for results, and interpersonal skills. These soft skills are also very important in agile software development environments but interpersonal and communication skills are the most critical to regularly meet team delivery targets.

2.4 Covert factors

Next, literature on agile methods is evaluated and integrated with the covert factor component of Fitzgerald's adaptation framework.

2.4.1 Agile methods

2.4.1.1 Mutual adaptation

Mutual adaptation of cross-functional team effort is most important to ensure legitimacy of agile approach as an important covert factor for software development. In cross-functional project teams, individuals require creating, understanding, and conveying knowledge to have successful mutual adaptation (Bygstad, 2004). In market-driven product development environments, collective effort, discussions and consensus on adaptations are vital to ensure agile as a legitimate development approach for organisations. Mutual adaptation requires collective decision making, involving all the stakeholders.

Mutual adaptation is a dynamic and emergent activity, which cannot be planned in advance and successful outcome is based on negotiation (Bygstad, 2004; Leonard-Barton, 1988). Empowered cross-functional teams in agile development environments are important to make collective decisions as project issues are encountered. Mutual adaptation requires any changes to work practices to be carefully analysed to establish the differences between what was previously done and improvements it will bring (Bynjolfson, van Alstyne, Bernstein, & Renshaw, 1997). However, mutually deciding on improvements of agile practices is critical so that any adaptation does not reduce the benefits or impacts the activities of product management, marketing and sales units.

A web-based survey by Begel & Nagappan (2007) at Microsoft with 500 responses, showed that less than 40% of the respondents' identified that agile methods did not worked well in their large teams, impacting collective effort and individual morale despite achieving improvements in communication, releases, and design flexibility in their small development teams. However, strategic market releases is vital for product leadership achieved through large team efforts on projects. Other key findings by Begel & Nagappan were that the daily scrum meetings were seen by developers as a micro-managing practice and a high level management buy-in was regarded as difficult by development managers since the progress reports and productivity metrics were hard to gather in their agile environment. Hence, agile benefits must be shown in practice to the stakeholders organisation-wide to be regarded as a legitimate development approach.

2.4.1.2 Empowerment

Developer empowerment is an important practice supporting aura of professionalism to development using agile methods. This developer empowerment is critical to provide speed for developing new products in uncertain conditions, enhancing competitive advantage at the marketplace (Reilly, Chen, & Lynn, 2003). An experienced report provided by Gat (2006) on a large-scale agile development at BMC Software highlights empowerment and risk taking practices as key for delivering a major product to the market in less time and with higher quality. According to Gat, empowerment must also be supported with appropriate team structure and roles (ScrumMaster, developers, product manager, tester and technical writer) to assign team responsibility and accountability for delivering features. Empowerment to make product related decisions, and for planning and managing the day-to-day operations at project level is a vital to enhance aura of professionalism to development work.

A seven month ethnographic study by Moe, Dingsoyr, & Dyba (2008) highlights three different types of autonomy (external, internal and individual) impacting self-organising teams when transiting to agile development. External autonomy is the outside influence on team activities, internal autonomy is the degree to which all team members jointly share decision-making responsibilities and individual autonomy is amount of freedom an individual has in carrying out assigned task. According to Moe et al., reduced external autonomy of team over project decisions and high individual autonomy are barriers to self-organising teams. External autonomy is vital to facilitate development level input for making first time right project decisions by senior management while consensus and task-sharing are two critical factors for deciding levels of individual autonomy in agile teams.

2.4.1.3 High Performance

Management confidence to successfully undertake development projects is a critical covert factor with agile methods. High agile team performance on a continuous basis is critical to provide management confidence for successful project implementations. Acceptance of agile culture organisation-wide is most important to achieve significant development performance benefits such as leaps in productivity and effective cross-functional effort (Ingalls & Frever, 2009). However, this agile culture is not always suited to all individuals or regarded as appropriate by some in organisations.

The organisational culture is widely acknowledged as critical for business success (Arogyaswamy & Byles, 1987). Through an accepted culture, employees feel that they are working for something meaningful while management get a full employee commitment for tasks (Jiwen Song, Tsui, & Law, 2009). With agile methods, a shared ideal of the agile culture is critical in a development environment.

Agile culture and practice strengthens the presences, value and significance of a shared project team identity instead of individual identity (Whitworth, 2008). This shared identity organisation-wide in a market-driven environment for the product is even more critical. According to Whitworth, the whole team involvement, values, culture of action and change, and collective thinking are agile practices which create a cohesive team while negotiation, iterative delivery, team meetings and feedback are associated with high team performances. Her study involved interviewing 22 individuals who were members of agile teams also state that agile practices supporting individual satisfaction and cohesive teamwork was highly dependent on how the practices were implemented and context in which they were implemented. Management support for agile adoption organisation-wide is most critical for gaining cross-functional support at development level for high development performances.

2.4.1.4 Work practice

Providing developer comfort through work practices is also critical to sustain high performance levels for long periods of time in agile product development environments. For this reason, comfort factor is an important agile overt factor. According to Segar, Hazzan & Bar-Nahor (2008), supervisory and co-worker support are crucial factors mediating relations between the individual psychological needs and agile orientation. Their study involved collecting data using survey method from 376 developers employed in an international Israeli company. However, agile team practices to deal with personal (non-work related) issues are also important for individual motivation to work in high performing environments.

According to Sfetsos, Angelis, & Stamelos's (2006) study, a 40 hour working week is one that is designed to address a sustainable level of productivity in agile development. They investigated XP method practices with 15 Greek software companies and discovered that 40 hour working week practice was not applied by large organisations. However, strictly adhering to rules for working hours is critical to sustain long term

high-level developer productivity in agile development environments. Such work related practice to minimise individual burnout is important overt factor facilitating to have a full development capacity on regular basis.

Next, agile methods literature is evaluated and integrated with the methodology-in-action component of Fitzgerald's adaptation framework.

2.5 Methodology-in-action

This section provides insights into some key aspects on development method adaptation to achieve agility with agile software development methods in a market-driven environment.

2.5.1 Method adaptation

Organisations are frequently adapting software methods as a result of different situations under which a particular software development project is undertaken (Kumar & Welke, 1992). Reflecting and incorporating knowledge, skills, and experiences gained from agile projects are critical for having an appropriate method-in-action so that mistakes are not repeated and good experiences are further explored.

According to Bajec, Vavpotic, & Krisper (2007), software development methods are never applied exactly as originally intended. Developers have different interpretations of how a method should be applied and never apply the same method in a similar manner to different projects (Fitzgerald, 1998). According to Fitzgerald the development environment now is dynamic, rather than static, as it was 25 to 30 years ago. Most important with agile development is that the method-in-action must reflect the dynamics of a project and then only appropriate solutions will be delivered.

Different versions of software development methods may exist, such as the one described by the creator, the one understood by the method user, and the one deployed for actual development projects (Ørvik, Olsen, & Sein, 1999). In agile development environments, the most important version is the one deployed on a project through adaptation. Adaptation is necessary if the interpreted method is not adequate to meet the development requirements of projects (Bajec & Vavpotic, 2008).

Organisations constantly change, making their new structures and processes incompatible with the existing method structures and without method adaptation they

clash impacting the project outcome (Baskerville, Travis, & Truex, 1992). For agile development success, it is critical that organisational structures, processes and method-in-action including the development team structures are created based on agile philosophy.

The context mismatch due to the bias and barriers in the interpretation and adaptation processes affects the adaptation of software development methods (Ørvik et al., 1999). According to Ørvik et al., these are results of cultural differences which influence judgment and the mental models of the appropriateness and applicability of software development methods. With agile development, a shared ideal of agile culture is extremely important to achieve appropriate method-in-action.

A study that investigated adaptation of software development methods in a real life setting was done by Fitzgerald, Russo, & O'Kane (2003). They investigated method tailoring practices at Motorola and revealed that method adaptation happens at different levels; at the industry, organisation and project levels. According to Fitzgerald et al., the software development teams must recognize the benefits of having a standard development method and the need for tailoring to match the specific requirements of development projects. The lengthy tailoring process was avoided at Motorola by adapting at macro level in advance while method tailoring at project level was done during the projects. Regardless of these levels, any formalised adaptation process in agile environments significantly impact development agility.

Work practice has a major influence on method adaptation for selecting or constructing of method fragments (Baskerville & Stage, 2001). According to Baskerville & Stage, organisational boundaries and structures, management, collaboration, techniques, tools and evaluation methods define work practice and are key drivers for adapting development methods. In agile product development environments fluid boundaries and structures, and participation are critical features allowing swift adaptations to have an appropriate method-in-action.

2.5.1.1 Method fragments

Method fragments are described by (Harmsen, 1997) in his doctoral theses as “a description of an Information Systems engineering method, or any coherent part thereof”. Software development methods are made of several method fragments or

building blocks (Aydin, Harmsen, Slooten, & Stegwee, 2005). A method fragment is a guideline, goal, value, principle, belief, practice, fundamental concept, tool, technique, activity, task, hint, and tip including the product to be delivered (Harmsen, 1997; Iivari, Hirschheim, & Klein, 2001). A collection of method fragments are used to express a software development method and they also play an important role in its adaptation to achieve an appropriate method-in-action for a project (Harmsen, 1997). However, highly experienced individuals in local product development are critical for selecting, creating and adapting method fragments to have an appropriate agile method-in-action.

2.5.1.2 Contextual factors

Contextual factors are the risk factors turning into problems and issues during project implementation. Problems have lower severity ratings, while issues have higher severity ratings implying that they can have a major impact on project outcome (Chin, 2004). In a market-driven agile product development environment, internal and external factors are both critical influencing adaptation to achieve an appropriate method-in-action for projects.

There are many risk factors which make each project unique and a common method unsuitable for all software development projects (Benyon & Skidmore, 1987; Brooks Jr, 1987; Sol, 1983). One solution is to adapt a method based on the contextual factors to have a more appropriate method-in-action for a project (Hofstede & Verhoef, 1997). This approach is highly desirable for agile projects where productivity and quality are critical factors for developing superior products.

Contextual factors are classified as internal or external factors of software development projects (Ewusi-Mensah, 2003). The internal risk factors are problems and issues associated with requirements definition, development teams, project managers, user involvement, project size, project management or project planning, while suppliers, top management support and organisational factors are classified as external risk factors (Wallace & Kell, 2004; Addison & Vallabh, 2002; Johnson, 2004; Mahaney & Lederer, 1999; Reel, 1999). According to Nathan-Ragu, Apigian, Nathan-Ragu, & Tu (2004), top management involvement and support significantly affects project portfolios. Sanders & Courtney (1985) revealed that the level of top management support was one of the key factors that determined implementation success. In agile development environments, management support is also critical for changing organisational, team and role

structures for swift learning of internal and external risk factors and adapting to have an appropriate method-in-action.

According to Raymond (1990), organisational size and maturity and product planning are organisational factors impacting project outcomes. Srinivasan & Kaiser (1987) identified the size of development teams, tenure of development personnel, level of technical competence, level of financial resources and external consultants as some of the main contextual factors for development projects. These are also critical contextual factors requiring adaptation to have appropriate method-in-action for agile projects.

2.5.1.3 Approaches for method adaptation

The method engineering and socio-organisational approaches are the two known points of view for method adaptation or tailoring in the information systems development community (Aydin et al., 2005; Nemuraite & Paradauskas, 2004; Baskerville & Stage, 2001; Fitzgerald et al., 2003).

The method engineering approach appears to be more talked about in the information systems development literature (Brinkkemper, 1996; Harmesen, Brinkkemper, & Oei, 1994; Henderson-Sellers, 2003; Karlsson & P. Ågerfalk, 2004; Kumar & Welke, 1992) compared to the socio-organisational approach (Aydin, Harmsen, Slooten, & Stegwee, 2005; Baskerville & Stage, 2001).

2.5.1.4 Method engineering approach

The method engineering approach is based on a theoretical model of the actual world (Iivari, 1989). Method engineering approach has a positivist approach, requiring a systematic means to adapt for having an appropriate method-in-action prior to its application (Baskerville & Stage, 2001).

Method engineering has an engineering approach to designing, constructing and testing practices, techniques or tools for adapting a development method and is heavily dependant upon documentation and processes to achieve adaptation (Hofstede & Verhoef, 1997). Most important to this approach is a repeatable and tested process to create an appropriate method-in-action. However, the strict requirements to follow the process and for providing documentation for change make this adaptation approach counter-productive in market-driven environments requiring speed.

The method engineering approach has a formal approach for method adaptation to achieve a situation-specific development method with tested components (Kumar & Welke, 1992). According to Kumar & Welke, this approach has four components; modular ISDM construction, stakeholder-value-based method composition, automated computer-based support, and a supporting organisational structure. However, this approach will make agile method adaptation a highly bureaucratic and time consuming process impacting the capability to compete.

The method engineering approach for method adaptation requires all the project-specific contextual factors to be identified, analysed, and understood. Only then can a relevant method-in-action can be assembled for a particular development project (Ayed, Ralyte, & Rolland, 2004; Bajec, Rupnik, & Krisper, 2006; Brinkkemper, 1996). However, market-driven product development environments also have emerging contextual factors requiring swift adaptation of the method-in-action during projects.

With the method engineering approach, a method-in-action is developed from predefined and pretested method fragments requiring a repository that can store method fragments (Ayed, Ralyte, & Rolland, 2004). This approach requires a scientific approach for creating or selecting method fragments from a repository and putting them together as a method-in-action according to the development conditions of a project (Henderson-Sellers, Gonzalez-Perez, Serour, & Firesmith, 2005). Method engineering also employs a contingency-based model for adaptation (Asadi & Ramsin, 2009), where the development context is the critical factor for selecting an appropriate method from a group of available methods or practices (Kumar & Welke, 1992). Agile market-driven development environments limit the repository's ability to provide appropriate fragments or method-in-action if the emerging contextual factors are unique.

2.5.1.5 Socio-organisational approach

The socio-organisational approach for method adaptation is driven by a practical model of the actual world (Iivari, 1989). The socio-organisational approach has an interpretative approach, requiring the method-in-action to be adapted as needed during its application (Baskerville & Stage, 2001). While this approach is highly suitable for agile development environments, it requires organisational behaviour for on-going learning of internal and external influences impacting swift application of the method-in-action.

The socio-organisational approach is firmly based on the concept of emerging contextual factors during projects playing a critical role in adapting the method-in-action and impacting the implementation outcome of projects (Baskerville & Stage, 2001; Fitzgerald et al., 2003). The socio-organisational approach is based on the view that the adaptation factors cannot be predicted in advance since they also emerge during projects (Baskerville & Stage, 2001). The context in which a particular development project is carried out and the organisational factors vary in a dynamic product development environment (Henderson-Sellers, Gonzalez-Perez, Serour, & Firesmith, 2005). For agile projects, the organisational capability for swift learning is most important for on-going adaptation of the method-in-action.

2.5.1.6 Static adaptation

Static adaptation, a key part of method engineering approach, involves adapting a method using tested and structured method fragments for having a relevant method-in-action for a project (Aydin et al., 2005). Static adaptation has a repository of tested method fragments from which appropriate or relevant method fragments are selected based on rules to build a project specific method-in-action (Ralyte & Rolland, 2001; Ralyte, 1999). The method engineering approach assumes that factors that impact the project will remain static throughout the project and no changes would be required to the method-in–action (Brinkkemper, 1996).

Static adaptation is driven by method engineers or project managers and software tools are used to select the needed method fragments for a project (Nguyen & Henderson-Sellers, 2003; Tolvanen, Rossi, & Liu, 1996). However, the actual method users are more appropriate for this task since they have first-hand knowledge of issues relating to the method-in-action.

2.5.1.7 Dynamic adaptation

Dynamic adaptation involves the method-in-action to be created and improved on the fly using un-structured method fragments (new fragments which are untested and unproven) and also using structured method fragments (Aydin, Harmsen, Slooten, & Stegwee, 2005). With this approach the method-in-action can change during the project, as the need arises. The socio-organisational approach for method adaptation is firmly based on the concept of dynamic adaptation since it recognises emerging contextual

factors as playing a critical role in the implementation success of projects (Baskerville & Stage, 2001; Fitzgerald et al., 2003). These contextual factors are both external and internal (Lyytineen, 1987a). Hence, this approach is critical for dynamic agile development environments.

Dynamic adaptation is driven by coaches or project managers to select, improve and create new method fragments needed for the method-in-action for a project (Aydin, Harmsen, Slooten, & Stegwee, 2005). However, the quick buy-in and confidence to immediately apply the adapted method-in-action by actual users are important as well.

2.5.1.8 Nature of agile method adaptation

A study of agile method adaptation by Aydin et al., (2005), shows that method adaptation has a pluralistic (engineering and socio-organisational) approach to achieve a relevant method-in-action for a project. Aydin et al., investigated adaptation of the DSDM agile method (dynamic systems development method) with a leading financial institution in Europe operating in a dynamic business environment and employing 2000 individuals for systems development work. They used agents, context, method fragments and process as constructs based on engineering and soci-organisation perspectives. They discovered that both static and dynamic adaptations were used to achieve a relevant method-in-action. Some of their other key findings were that the coaches or method engineers drove static adaptation, whereas the coaches and project managers drove dynamic adaptations. They also discovered that static adaptation involved adapting and adhering to an appropriate method-in-action to get started on a project. According to Aydin et al., dynamic adaptation involved adapting to have a relevant method-in-action and also creating new fragments to adapt the method-in-action on a project. While the specialist coaching role has been highlighted as critical for dynamic adaptation at the case study organisation, on occasions senior engineers will need to adapt to become coaches since such permanent specialist roles are not encouraged in agile teams.

A study by Henderson-Sellers & Serour (2005) suggests that agile method adaptation is best achieved through the method engineering approach. Their study involved two organisations where they successfully engineered agile methods using a repository of method fragments. According to Henderson-Sellers & Serour, these methods have been engineered with people in mind and can be rapidly changed and self-tuned by

developers to adapt to the requirement and environmental changes when needed to have an appropriate method-in-action on a project. They refer to the adapted method as a dual-agility method since in the future the methods will be adapted from a repository of method fragments by developers rather than as done by method engineers or project managers with the method engineering approach. However, the method engineering approach is not known to accommodate the social aspects of method adaptation.

Method engineering is process orientated focusing on structural aspects of methods (Baskerville & Stage, 2001). Specifically, the organisational factors such as the traditions, political views and communal interactions do not influence method adaptation but method engineering is driven only by technical considerations of method fragments (Oinas-Kukkonen, 1996; Harmsen, Brinkkemper, & Oei, 1994). However, communal interaction is the main underpining of agile philosophy where method adaptation is driven by a shared understanding and achieved through organisational-wide interactive behaviour to instantaneously adjust practices on the fly achieving a relevant method-in-action.

2.6 Summary

The review of the literature identified the adaptation framework i.e. theoretical model for this study. While this adaptation framework is mostly relevant to non- agile software development processes, the integration of appropriate literature relating to agile software development with the major categories of this adaptation framework (the profile of the development environment, overt factors covert factor, and methodology-in-action) shows that these categories at conceptual level are also applicable with agile method adaptation. Through the review of literature some differences have been identified between agile methods and the adaptation factors listed with the major categories in the adaptation framework. In agile development environments, in-house development is supplemented with outsourcing and sub-contracting, legacy system development has formalised planning practices, and productivity and rigor are equally important. Integrating agile literature into overt and covet categories of the framework suggest different motivation underlying these factors in agile development environments. While the literature confirms that method-in-action on projects are adapted as highlighted in the adaptation framework, in agile product environments the adaptation approach is dynamic.

Chapter three: Research methodology

This chapter presents the research approach used to investigate the adaptation factors of agile methods. It contains information explaining the research methodology to:

- explain paradigmatic assumptions which guided data collection, analysis and reporting of results
- justify using a case study approach to answer the main research question
- explain the research design, validity reliability and triangulation considerations
- provide the approach used for case study and individual participation selection
- show data collection sources, procedures and key instruments used
- explain data organisation and analysis from the multiple sources
- show the adopted ethical consideration
- highlight the reporting style used for this research
- present the methodological model for the study

3.1 Research paradigm

A paradigm is defined as a basic set of beliefs that guides how a qualitative or quantitative method and practices relating to data collection, analysis, and interpretation are used for conducting a study (Guba, 1978). This research was driven by a paradigm that enabled the researcher to conduct compelling research by collecting data directly from agile methods practitioners in real software development environments.

The research paradigm decision was made based on the nature of the inquiry and the questions being investigated, and was independent of the selection of the case study method to carry out this research (Myers & Avison, 2002). The research paradigm for qualitative methods may be positivist, interpretive or critical (Myers & Avison, 2002). This study adopted a positivist approach in an attempt to identify and provide understanding of agile adaptation factors based on predefined research questions and prior constructs (Paré & Elam, 1997; Benbasat, Goldstein, & Mead, 1987). Positivist study is defined as having a scientific approach for investigating a research problem. Reasons for using a qualitative rather than a quantitative approach to investigate agile adaptation factors are provided in next section of this chapter.

With the aim of providing pragmatic information on agile method adaptation, this research used an empirically-grounded adaptation framework for software development.

While a detailed description of this framework has been provided in Chapter two, Figure 4 in this Chapter (page 103) provides the model of the framework. The relevant components of this framework are used as constructs and to formulate the research questions, taking a positivist stance for this study. With a positivist approach, construct validity ensures that the reported phenomenon was accurately captured from a natural setting. Approaches to ensure construct validity with this research are provided in section 3.4.1.1 of this chapter.

Adopting prior constructs was an important factor to select case study research as the qualitative method to carry out this investigation. However, a detailed comparison of various qualitative methods in selecting case study method is provided in section 3.3.2 of this chapter. A positivist case study research is driven by a prior theory (Yin, 2002). Hence, prior constructs motivated a positivist case study method for this research.

Other positivist research practices stated by Creswell (2007) were also adopted for this research, such as: ensuring a chain of sensibly related steps, using multiple perspectives on reality, having a rigorous approach for data collection and analysis, adopting approaches to ensure validity and writing the report in a scientific manner.

3.2 The role of the researcher

Prior to the entry into the field, a qualitative inquiry requires a researcher to take up naturalistic themes (Hoepfl, 1997), develop a level of skills to adequately capture and interpret data (Strauss & Corbin, 2008; Yin, 1994) and prepare a research design suitable for qualitative inquiry (Creswell, 2007). In this study, the researcher adopted naturalistic themes (identified in section 3.3.1 of this chapter) to avoid hindrance and manipulation in agile development environments while collecting data from method users and placing no prior constraints on research outcomes on agile method adaptation. A key aspect of gaining cooperation from case study organisations was building trust and rapport by explaining the purpose of the research, which was to learn from them about agile method adaptation, and to identify agile method adaptation factors and explain then to the software engineering community.

Preparation to attempt this research was done through reading and understanding literature on agile methods, study design, and study methods and tactics which was followed by presentations of the proposed research at two international doctoral

consortia and a local annual post-graduate conference. This preparation also included a detailed presentation of the theoretical framework to a case study organisation. The preparation helped the researcher to build the competence to understand and give meaning to data and the ability to identify and separate the relevant data. A flexible research design was adopted allowing the researcher as a data collecting instrument to relate to a particular development environment to collect data and further investigate any unexpected findings.

3.3 Method selection

Qualitative studies are used to answer questions about the complex nature of phenomena that occur in a natural setting and end with tentative answers or hypotheses (Leedy & Ormrod, 2001). In contrast, quantitative research enables researchers to count and measure statistics of two types: descriptive, which explains quantitative data in a summary form, and inferential, which draws meaningful and significant conclusions from quantitative data (Gillham, 2000). With agile method adaptation, very little knowledge or understanding exists, requiring in-depth investigation of the entire agile development approach and product development environment to identify and understand agile adaptation factors. Qualitative study was more appropriate to investigate agile adaptation factors from a few successful agile development environments having experience in adapting their agile methods and providing a basis for future learning of agile adaptation factors.

Qualitative study enables the researcher to study selected issues in-depth and to collect data which are detailed descriptions of situations, events, people, interactions, observed behaviours, and quotations from people about their experiences, attitudes, beliefs, and thoughts (Patton, 1990). Hence, qualitative studies provide understanding of events, situations, experiences, and actions of participants in organisations including the context within which the participants act, and processes by which events and actions take place (Myers & Avison, 2002). Quantitative methods require the use of standardised measures so that the varying perspectives and experiences from a great number of people can fit into a limited number of predetermined response categories (Patton, 1990). Quantitative methods would not have enabled the collection of detailed information to provide insights into individual context and reasons for making adaptation decisions. Qualitative method allowed the researcher to make an in-depth study of agile production labs

through observation and asking related questions, learning directly from the individuals in regards to agile adaptation factors. Quantitative study will be an appropriate way to learn about the general applicability of agile adaptation factors to agile software development community, once these factors are investigated from a few successful cases.

The selection of a qualitative approach for this research was based on the understanding that it was the most appropriate way to investigate agile adaptation, on which very little knowledge currently exists.

3.3.1 Qualitative focus

The themes of qualitative inquiry identified by Patton (1990), listed in Table 3, were applied to ensure the validity and rigor of this research. They ensured an in-depth investigation without any prejudice or manipulation of adaptation data collected from the case study sites. These themes enabled a credible research strategy to establish significant and compelling findings on factors that influence agile methods adaptation.

Table 3 Adopted qualitative inquiry themes

1. Naturalistic inquiry	Observed the development teams; understood and documented their development environment, adaptation and application of agile practices without any hindrance.
2. Inductive analysis	This was an explanatory and exploratory study; the focus was to learn about agile adaptation from the development teams without imposing any prior expectations; asked method users legitimate open questions allowing adaptation factors to emerge.
3. Holistic perspective	Investigation took into account the entire setting for software development; methods, practices, tips, techniques, tools and available resources for the teams. It enabled gathering data on multiple aspects of the agile approach.
4. Qualitative data	An in-depth inquiry that produced thick description based on direct quotations by method users; they gave their perspectives and experiences. Resulting description provides understanding on agile method adaptation without any judgment.
5. Personal contact and insight	Data collection done at the development lab; got in direct contact with the team members, their development environment and method application. Provided the necessary insights to understand agile adaptation.
6. Dynamic systems	Expected the likelihood of unanticipated findings since the prevailing software development environment is dynamic. Naturalistic inquiry enabled the researcher to investigate, describe and understand agile adaptation in such an environment.
7. Unique case orientation	Two case studies are successful software development houses with international customers and competitors. Both have successfully adopted an agile approach; they provided rich information. The first level of inquiry captured the details of adaptation of individual cases. The cross-case analysis

	compared their method fragments and identified their adaptation factors.
8. Empathic neutrality	This research provides understanding on agile adaptation factors in its complexity (not advocating any agendas), learnt through association with development teams and by experiencing their method application. Gained insights to relevant data and took a unbiased stance towards emerging adaptation factors.
9. Design flexibility	Adopted a flexible and iterative design approach; it enabled the pursuit of new paths of discovery as the understanding of agile adaptation developed.

Source: (Patton, 1990)

3.3.2 Case study approach

There are several qualitative methods for research such as case study, ethnography study, grounded theory, and content analysis (Leedy & Ormrod, 2001). However, no research method is superior to others and the method selected to investigate an information systems phenomenon is based on current knowledge of the topic and the nature of the topic being investigated (Benbasat, Goldstein, & Mead, 1987).

Case study, ethnography study and grounded theory methods consisted of three kinds of data collection: in-depth, open-ended interviews; direct observation; and written documents (Patton, 1990). This multi-source approach for data collection, known as triangulation, makes it possible to confirm if participants actually practice what they have said (Gillham, 2000) determining convergences and giving credibility to the patterns that emerge (Yin, 1994).

Ethnography study and grounded theory methods require generating theory through research data rather than testing or applying ideas formulated in advance, prior to data collection and analysis (Dey, 2004: Linclon & Guba, 1985). Case study research is driven by prior development of theoretical propositions to apply, test or generate a relevant theory providing a more scientific approach for the research (Yin, 1994). Grounded theory requires no literature review prior to research, no taping of interviews and no discussion of theory before being written (Glaser, 1998). Ethnography research also does not require a prior model, emphasising that no theoretical model can explain in advance the likely findings in a new contextual setting (Yin, 1994).

These requirements of grounded theory and ethnography study were seen as a major obstacle by the researcher for conducting this study. The researcher required prior readings on agile publication (books and academic literature) to have a good grasp of

agile methods and to identify research problems. Having a theory or conceptual framework to guide data collection and analysis was critical for this investigation, in order to ensure the researcher's confidence to carry out this research and also for providing confidence in the findings.

The content analysis method focuses on investigating any recorded (books, websites, paintings and law) verbal, visual, or behavioural forms of communication and requires coding of material in terms of predetermined and precisely defined characteristics (Babbie, 2007). No formal documentation on agile adaptation decisions was kept by agile development teams, making it impractical to be solely dependent on this research method. However, an element of content analysis is part of the case study, ethnography study and grounded theory methods to analyse and code interview transcripts.

The case study method was determined to be the most appropriate and best suited qualitative method for this study, since the researcher investigated "how" and "why" research questions (Yin, 1994). In particularly, case study method is particularly suited to learn in-detail through an in-depth study of a few successful cases on a phenomenon on which very little knowledge exists (Dube & Pare, 2003; Patton, 1990). Hence, case studies are chosen for in-depth investigation of the agile product development environment and development approach to learn about agile adaptation from a few successful software development organisations.

Ethnography study also enables an in-depth study of a contextual setting to understand the day-to-day practical functioning of individuals but emphasises longitudinal involvement (more than a year) to observe and ask appropriate questions (Walter, 2006). However, a long term continuous involvement for observations at software production sites was seen as hindrance by development teams in meeting their short cycle delivery commitments. For this research, it took over a year to collect data through non continuous involvement. The grounded theory approach requires theory development to be repeated until the researcher reaches the data saturation point for theory creation from a new case. Grounded theory is more suitable for investigating a single aspect of phenomena and if the researcher has access to large pool of case studies to sample from as the study proceeds. Given the lack of a large number of local cases and the motivation for a holistic investigation of agile approaches to identify and

provide understanding about agile adaptation factors, the grounded theory approach was deemed not suitable for this study.

The case study method, similar to ethnography study and grounded theory enabled the collection and compilation of rich data and detailed descriptions of events, interactions, observed behaviours, and direct quotations from method users through direct access to the software production labs, allowing the researcher to see and experience agile method application (Donmoyer, 2000). This facilitated the identification of the relevant agile adaptation factors, which previously were not known (Leedy & Ormrod, 2002).

The case study, ethnography and grounded theory approaches tend to be idiosyncratic (Stake, 2000). This research used an idiographic (naturalistic context) rather than nomothetic (controlled environment) strategy to investigate and collect adaptation data (Fitzgerald & Howcroft, 1998; Lincoln & Guba, 2000). Idiographic research in organisations required the researcher to be absorbed within the organisation's culture for a significant period of time to understand particular events, enabling rich data to be collected. In the beginning, a week was spent at each case study site observing the application of agile methods and individual interactions in development environments as daily routines were performed.

Observation allowed a significant understanding of team cultures, application of agile method learning, various aspects of the methods (fragments) and their application, team environments, the team resources, individuals and make up of the teams, and the other stakeholders, which were further investigated with interviews.

3.4 Research design

A research design unfolds as the research proceeds and a sequential model is not a good fit for a qualitative study since the components of the design need to be changed due to new developments (Maxwell, 2005). Maxwell suggests five components for a research plan; goals, conceptual framework, research questions, methods, and validity.

This framework, known as An Interactive Model of Research Design, was adopted for this study with the understanding that it would be improved as understanding of agile methods grew both through empirical data collection and newly published materials. As

a result, sub-questions were modified as the researcher acquired further understanding of the agile approach through data collection.

These five components provided the guidelines to plan and conduct the various activities and tasks of this research. Figure 3 shows the research design plan used for investigating adaptation factors of agile methods. The goals and research questions of the research are addressed in Chapter one of this thesis. The conceptual framework component is addressed in Chapter two. This Chapter addresses the research methodology and validity considerations.

3.4.1 Validity consideration

Validity with qualitative studies is defined as how truthful the research results are or if research has truly measured what it intended to measure (Golafshani, 2003). Validity consideration ensures the accuracy, meaningfulness, and credibility of the data collected, conclusions and entire research (Leedy & Ormrod, 2001; Maxwell, 2005). With qualitative inquiry, validity is dependent on the skill set, competence, and rigor of the investigator (Patton, 1990).

The validity of qualitative research must be relevant to the relationship between a description of an account and something outside of that account i.e. the phenomena under study (Maxwell, 2002). Ensuring validity makes certain that there are no alternatives or rival explanations for your research findings (Huck & Sandler, 1979; Maxwell, 2005). For this research, literature was used to support the findings to ensure no alternatives or rival explanations for the research findings. Maxwell points out that validity is not an inbuilt characteristic of a method used for data collection but relates to description of data or accounts provided, or a conclusion reached in a particular context.

This research adopted fallibilistic validity to ensure reliability of the research findings, which is the test to determine if the accounts accurately capture the social phenomena to which they refer (Schwandt, 1997). The suitability of using the case study method to investigate agile method phenomena is discussed in section 3.3.2 of this chapter explaining how this validity was achieved.

Another approach used to ensure fallibilistic validity for this research was to ensure the appropriateness and familiarity of the adaptation framework (Yin, 1994; Marshall & Rossman, 1999; Patton, 1990), which is discussed with data collection methods.

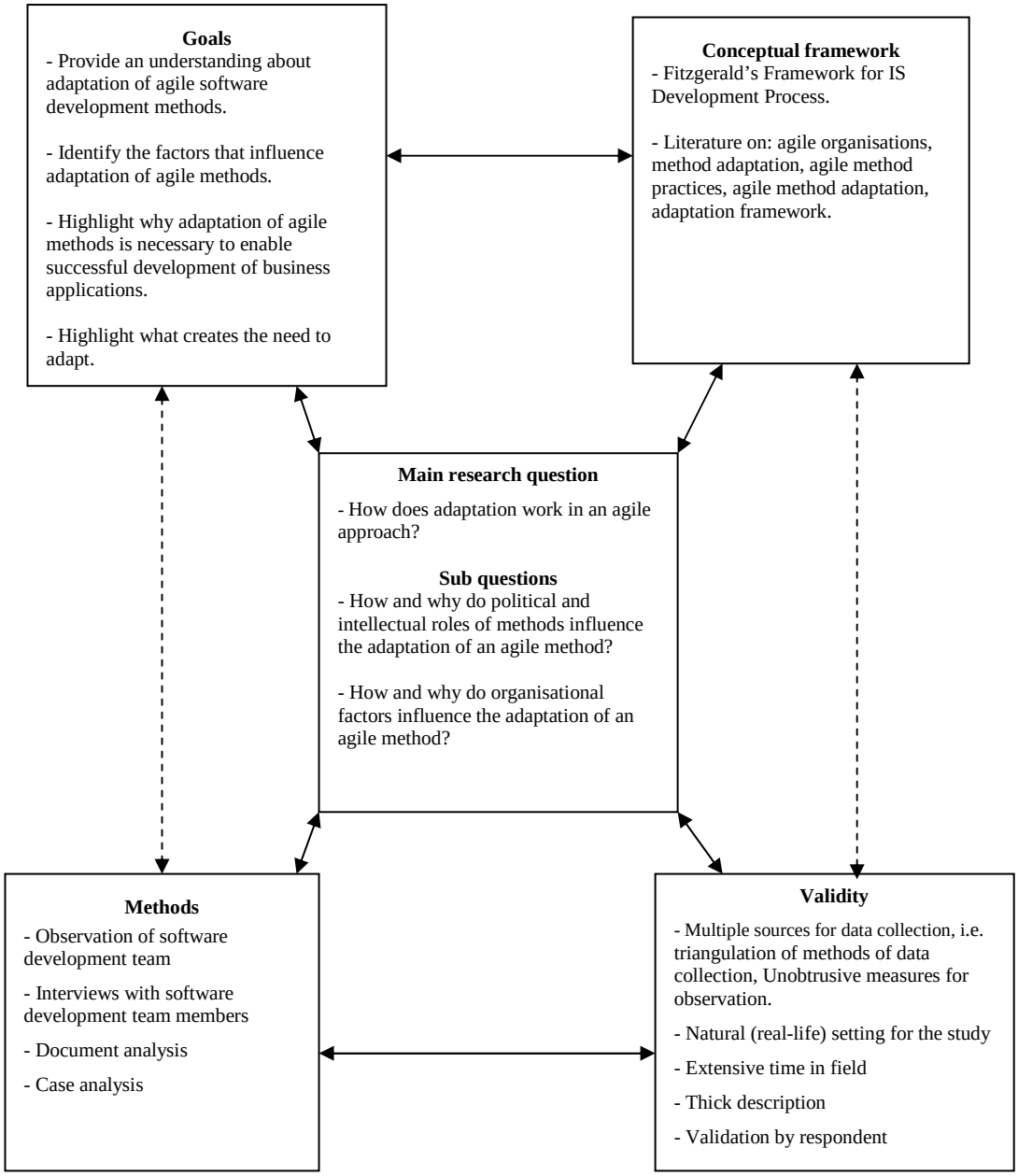


Figure 3 Research design - based on Maxwell's (2005) model

The other approach to ensure fallibilistic validity is to identify the validity threat, which is the way you might go wrong; the researcher bias and reactivity (Maxwell (2005)

- Researcher bias happens when the researcher selects data that "stands out" to the researcher or that fits the researcher's existing theory or preconception (Miles & Huberman, 1984; Shweder, 1980).
- Reactivity is the effect of the researcher on individuals studied or the setting.

The researcher had done substantial reading on agile methods, agile applications, and agile practices, which sometimes can lead to assumptions about how the methods are or should be used in real software development environments. The researcher had recognized this as a source of researcher bias and reactivity.

To counter these two validity threats the interview questions were designed based on observation of the two organisations' software development teams, ensuring that the questions were relevant to their agile practices. Listed in Table 4 are the strategies applied in this study (Creswell 2007; Maxwell, 2005; Sturman, 1999) to ensure the validity of research.

Table 4 Validation strategy

Validation strategy	Description
Natural setting	Interviews and observations were conducted in product labs (work environments). Non-manipulative, unobtrusive and non-controlling data collection.
Process of inquiry	Data-gathering procedures explained in detail in this chapter.
Intensive, long-term involvement	Data collection activities took over 12 months. Captured data from different sources and situations within the setting.
Rich data	Long-term involvement aided by interviews and observation helped collect rich and detailed data. Interviews with participants were recorded and transcribed. All observations of events or situations within the team's software development environment were noted.
Data presentation	Data presented clearly and in a format ready for analysis.
Rich, thick description	Provided description in detail. Readers could make decisions regarding application of findings to other similar settings.
Data types	The secondary data (websites) clearly highlighted.
Respondent validation and member checking	The case study report was given to study participants to validate their data as well as any interpretation of data to rule out any misinterpretation.
Discrepant evidence, and biases acknowledged	Each case study was analysed to identify any discrepant data and biases and if it would affect the conclusions on agile adaptation factors. These would have been highlighted for readers to judge and draw their own conclusions.
Triangulation	Multiple sources such as interviews and observations were used to collect data to explain agile method adaptation from each case study. In each of these methods vulnerabilities were recognized to prevent any bias in collecting and reporting of data.

Source: Maxwell (2005), Sturman (1999) and Creswell (2007)

An other strategy was to ensure that the research design met four quality tests, which were the construct validity, internal validity, external validity, and reliability (Cook & Campbell, 1979; Gillham, 2000; LeCompte & Goetz, 1982; Lincoln & Guba, 1985; Maxwell, 2002; Wiersma, 2000; Yin, 1994). These are explained in the next sections.

3.4.1.1 Construct validity

Construct validity refers to the validity of concepts or categories that a theory or a framework employed when applied to a phenomena (Maxwell, 2002). According to Maxwell, the theoretical concepts and categories, and their existing relationships enable the creation and presentation of the reality of a phenomenon in a particular case.

There are many ways by which reality may be constructed, these are objective reality, perceived reality, construct reality, and created reality (Lincoln & Guba, 1985). Construct reality was the goal of this study.

Construct reality assumes that there are multiple realities for a phenomenon under investigation. An infinite number of reality constructions might be produced. The constructed realities must match the real entities as closely as possible. The reported realities must be reinforced by society as a whole by an accord (Lincoln & Guba, 1985; Maxwell, 2002).

Three ways to increase construct validity are to use multiple sources of evidence, establishing a chain of evidence and having a draft case study report reviewed by key participants to get feedback (Yin, 1994). The multiple sources of evidence that were used to collect data on various constructs for this study were interviews, direct observations, and document analysis.

To establish a chain of evidence in the report, appropriate citations were used when quoting the actual data as collected from a document or an interview or from an observation. A database for this research using NVIVO was established where all data collected were kept as described in the case study protocol.

Reviews of draft reports of both case studies were done to get feedback from both case study organisations and peers. This enabled the comparison of the views of the reviewers with the researcher in terms of conclusions drawn about adaptation. This process also enabled validation of the factual information recorded for a particular case study.

3.4.1.2 Internal validity

The trustworthiness of research findings is the main criterion to persuade audiences of their validity (Gillham, 2000; Lincoln & Guba, 1985). The notions of trustworthiness

with research findings relate to the value placed on the finding (truth), applicability, consistency, and neutrality related to internal validity, external validity, and reliability (Lincoln & Guba, 1985).

The internal validity of the case study approach ensured that the results of the study were seen as accurate and were interpreted with confidence, helping to minimise other plausible and competing explanations of a result (Wiersma, 2000).

With explanatory case studies, before describing and concluding the relationship between the constructs, a thorough investigation should be done to establish if there exists a catalyst between them (Cook & Campbell, 1979; Yin, 1994). The cause of the relationship between the constructs must be identified. If not, then the research design has failed to deal with internal validity (Yin, 1994).

Another threat to internal validity is in regards to assumptions made regarding an event that cannot be verified through observation but was based on interview and documentary evidence (Cook & Campbell, 1979; Yin, 1994). The internal validity for this case study research was ensured through continually asking the following questions; are the assumptions being correctly made?; have various alternatives and possibilities been well thought-out?; does evidence link or meet with other research?; and are there possibilities for any criticisms (Yin, 1994)? These questions were addressed during data collection and analysis to ensure internal validity for this research.

The pattern-matching tactic was also employed for data analysis (Patton, 1990). Pattern-matching logic compares an empirically based pattern with a predicted one and if the two patterns match each other then internal validity of case study is strengthened (Yin, 1994). For this research, each adaptation factor in the adaptation framework was predicted as an agile adaptation factor and compared with the empirical data that were organised under it to see if they matched for both case studies. Key literature on adaptation was used to explain and link research findings to ensure internal validity (Marshall & Rossman, 1999; Eisenhardt, 1989).

3.4.1.3 External validity

External validity is defined as the extent to which the research findings are applicable to the general population. Internal validity is a precondition for external validity (Wiersma,

2000). The research findings and results can only be considered to be applicable and practical if they can be interpreted and seen as practical by individuals.

The threats to external validity are as follows: selection effects, setting effects, history effects and construct effects (LeCompte & Goetz, 1982). The selection effect threat is a mismatch between the construct and some of the selected group being investigated. The selection effect threat was dealt with by ensuring that the two case studies were rich and relevant since they were using and adapting agile approaches for a significant amount of time.

The threat of setting effects is where the results of the case study are a reflection of the contextual issues resulting from the organisational setup and cultural beliefs in the case. The context in which each organisation has been investigated must be provided for and described for each case study for individuals to understand the reasons for the results of each case study. The contextual issues are spelt out for both case studies in Chapters four and five with case study descriptions.

The history effects threat is where the unique historical experiences may weigh against comparisons for case studies. Historical effects, if any, would have been identified for both case studies for the readers in Chapters 4 and 5 for them to draw their own conclusions on the validity and reliability of the research findings.

The construct effects threat is where the constructs studied may be unique to the studied group. The discussions, presentations, and question and answer sessions on the adaptation framework with case study participants gave confidence on the constructs.

Replication logic was applied to achieve consistency to ensure reliability of the research (Lincoln & Guba, 1985; Yin, 1994). Reliability was achieved through the repetition of a similar inquiry process for both case studies.

3.4.2 Reliability

Enforcing reliability ensures that the study is free from partialities and inaccuracies (Yin, 1994). Validity directly influences the quality of the research but it does not necessarily ensure the reliability of the research findings (Lincoln & Guba, 1985).

Reliability issues were dealt with by defining a research protocol (Lincoln & Guba, 1985; Yin, 1994) showing the steps and procedures that were followed for undertaking

the research to ensure that any other person can repeat the research as accurately as possible.

The process used for the investigation of the adaptation factors of agile methods and all the artefacts used and produced such as process notes, raw data, transcribed data, categories for data reconstruction and synthesis etc., can easily be audited (Lincoln & Guba, 1985). In addition, the case study database was developed (Yin, 1994) using NVIVO to ensure accuracy and easy accessibility of the stored information (Lincoln & Guba, 1985).

3.4.3 Triangulation

Triangulation is the use of different and multiple sources, researchers and theories to provide confirming evidence (Lincoln & Guba, 1985). Triangulation is also a means to ensure construct validity (Creswell, 2007; Yin, 1994). The concept of triangulation with qualitative research is very important since it provides credibility and validity to the study (Maxwell, 2005; Sturman, 1999; Creswell, 2007; Patton, 1990). Hence, multiple sources for data collection are used for this research.

Interviews, observation, and document analysis, listed in Table 5, were the three different data collection techniques employed for this research. The interview method was the primary method used for data collection but the observation technique provided the researcher with the opportunity to understand both case studies' agile environments, identify their various method fragments, and observe their software development teams in action.

The interviews only followed after a considerable amount of understanding was achieved of both case studies' agile setup. The observation technique provided clues and information, which helped to revise and rewrite the interview questions.

Document analysis, the secondary source of information, enabled the collection of organisational and historical data related to each of the case studies from their Web sites. According to Yin (1994) using multiple methods to collect data enables a researcher to address a wide range of historical, attitudinal, and behavioural issues. Therefore, the findings and conclusions are deemed more accurate and reliable when multiple sources are used to collect data in a qualitative inquiry.

The researcher noted that the observation technique was necessary for this research and without using it less would have been achieved in learning and understanding the agile approaches of both the case studies. This helped greatly to prepare and write up questions relating to particular aspects of a method that was used by the software development teams.

Table 5 Data collection methods

Data collection method	Description
Participation observation	Explained the nature of the researcher involvement. The agile teams were observed in their production labs (methodology-in-action). Took notes and recorded events, behaviours, and artefacts. Development teams guided the researcher on day to day practices.
In-depth interview	Main method used to collect data. Based on observation of agile product development environment. Used an interview guide for in-depth interviews. Questions asked in a systematic way. Interviewed more than one person, assembled an extensive multiplicity of information. Had full cooperation from participations.
Document analysis	Complemented interviews and observation to collect data. Information from company website and industry publications. Content analysed and validated with both case study organisations to ensure reality.

Source: Marshall & Rossman (1999)

There are four types of triangulation techniques that are normally used with qualitative study: (a) data triangulation - the use of multiple sources for data collection, (b) investigator triangulation - the use of more than one researcher or investigator for data collection purposes, (c) theory triangulation - the use of multiple perspectives to interpret a single set of data, and (d) methodological triangulation - combining a quantitative method with qualitative inquiry for research (Denzin, 1978; Patton, 1987). It was not practical to employ all the four triangulation techniques.

Investigator triangulation was not possible for this research due to the nature of this study, which requires individual academic work and also due to budget, and time constraints. The researcher's investigation into literature suggests that no other comprehensive theory or framework relating to software development methodologies developed that could have been used for data interpretation.

Hence, data triangulation was the most realistic technique to achieve triangulation for this research. The three sources enabled a comprehensive perspective on adaptation factors of agile methods for both case studies.

3.5 Case study selection & context

According to Yin (1994) with multiple case studies, cases must be carefully selected so that they either predict similar results (a literal replication requiring 2 or 3 similar cases) or produce contrasting results but for predictable reasons (a theoretical replication requiring 4 to 6 different cases). For this research, literal replication was chosen because of lack of understanding on agile adaptation factors. However, with future investigation theoretical replication would be critical to provide a better understanding of agile adaptation factors due to the nature and sizes of agile software teams and their organisations.

Two software development houses, one from New Zealand and the other from Australia, were used in this study. These two organisations were selected because they have been using agile software development approaches for a significant period of time, one since 2003 and the other from 2004. Most importantly, these two organisations had experience in adapting their organisational, team and development structures including development practices since their agile adoption to maintain their status as market leaders with their software products.

Another important similarity (required for literal replication) between two cases which made them appropriate for this research was that both were in market-driven business environments facing similar types of challenges and requiring adaptation with their agile approaches to deal with them. Two cases were sufficient, since an adaptation framework (theoretical framework) was applied to each case study organisation to learn and identify agile adaptation factors.

The selection criterion was that cases must have adopted an agile approach for at least a year or had two or more agile projects lasting for a year while also having experience in adapting practices, team structure, organisational structure, or role structure to deliver an agile project. Locally, there were only a few organisations that had adopted all aspects of agile software development approach and had experience in adapting them. Two other agile software development organisations were approached for this study but they pulled out in the early stages of discussion due to their development commitments. In addition, eight other local organisations showed interest in this study. However, they were either exploring agile methods or were on the verge of making a decision on whether to adopt agile methods. These organisations were deemed not appropriate at

that time for investigation to learn about agile methods adaptation and a third significant case could not be added for this research.

However, two software development organisations which agreed to be part of this research provided an opportunity for in-depth study to collect data to identify and provide understanding of agile adaptation factors on which no investigation has yet been carried out. Such cases are referred to as revelatory cases because of their significant experience in agile adaptation (Yin, 1994).

There are two sampling strategies, probability sampling- used with quantitative study and purposeful sampling- used with qualitative study (Patton, 1990). However, the selected cases for qualitative inquiry do not need to be a true representative of the population, but must be information-rich to enable an in-depth study in order to learn in detail the problem under investigation (Kuzel, 1999; Patton, 1990). These two cases were information rich since they had successfully adopted and adapted agile approaches allowing the researcher to identify and provide understanding on agile adaptation.

For this research, the trade-off was the depth of study with the breath of study, where a wide range of agile practices from a few cases rather than limited number of practices from a larger number of cases are investigated (Patton, 1990). Hence, this research attempts to provide an understanding on a wide range of agile practice adaptation for the software development community by using two cases through an in-depth investigation.

There are no rules for sample size in qualitative inquiry but the time-frame within which the research must be concluded often limits the number of case studies that can be investigated (Patton 1990).These two organisations were determined sufficient enough for an in-depth study to carry out and report the research findings within the required time for this study. This in-depth study of the two case study organisations experienced the challenge for multiple site visits, which ended up with prolonged data collection from September, 2006 to January, 2008.

For this research, a small local sample size limited the number of case studies that were investigated to two rather than three (two or three case required for literal replication). However, the research design was done, keeping in mind that this research could be replicated with large number of cases. Most importantly, in-depth investigation of two

sites also provided a better understanding of reasons for individual site's adaptation decision making (Schofield, 2000).

Kuzel (1999) highlights the issue of the appropriateness of cases for a case study research. For this research, the appropriateness of the two cases was assured by making sure that the two organisations had the motivation and belief for the need to learn and provide practical information and knowledge on agile adaptation.

The selection of the two cases was done through their presence at the 2005 Agile Conference held in Auckland, New Zealand. The researcher approached them to be part of the research and both accepted the invitation with the view that the research findings may help them to further improve their agile approaches. Both organisations were major contributors to the 2005 Agile Conference where they presented their agile methods which convinced the researcher that both organisations had a firm belief in the agile way of developing software and were excellent case study organisations for this research.

According to Creswell (2007) the researcher must make a decision about "who" or "what" should be sampled, what form the sampling will take place, and how many sites and people need to be sampled. In addition, Kuzel (1999) lists event, person, artefact, activity, and time as various possibilities as to "what" could be sampled. One that was of particular interest to this study was various "individuals" with different roles, responsibilities, and job titles involved directly with software development and documentation.

The majority of these individuals from both the case studies were software engineers, who were the main source for information for this study. Others were director of software engineering, engineering manager, product managers, project managers, product analysts, team leaders, technical leaders, sales and marketing personnel, quality assurance manager, quality assurance engineers, usability and technical communication engineers. One of the case study sites had eight quality assurance engineers (agile testers) assisting their three software development teams.

While it was impossible to interview all the software and quality assurance engineers from both the cases, interviews were conducted with a mixture of individuals, selected

by their job titles. These engineers were at levels of graduate, post-graduate (engineer title), senior, and principal levels reflecting their experiences of software development.

3.6 Generalizing using two case studies

To give credibility to information produced and explanations provided, this research employed a two-case study approach. This had the distinct advantage of applying the adaptation framework separately with two case study sites to produce more persuasive evidence, to have more vigorous study and providing a firmer basis for conclusions on the adaptation factors of agile methods (Firestone & Herriott, 1984; Leedy & Ormrod, 2001; Schofield, 2000; Yin, 1994). This two-case study approach enabled a multiple perspective on the agile adaptation factors identified in the adaptation framework (Creswell, 2007).

Qualitative research may involve the extension or modification of a framework (Miles & Huberman, 1984). For this purpose, it was appropriate to apply the adaptation framework with more than a single case study to reflect emerging agile method adaptation factors.

Sample size in qualitative inquiry is dependent upon the purpose of the inquiry, what gives credibility to findings and what can be done with available time and resources (Patton, 1990). For this reason, the typical cases that matched the problem are selected.

Case study research requires no more than 4 or 5 cases (Creswell, 2007). Replication logic is applied to determine the number of cases to be investigated, which are two to three cases for literal replication and between four to six cases for theoretical replication (Yin, 1994).

The case study generalization provides useful explanation of past data but not an accurate prediction of future happenings and just provides explanation of a phenomenon which may be valuable in future for other organisations (Walsham, 1995).

A single case is good enough for empirical testing of a theory (Lee & Baskerville, 2003). Testing the theory in a setting allows generalizing the theory to that setting only (Lee & Baskerville, 2003; Lee, 1989).

Generalizing from Theory to Description (Type TE Generalizability) approach was adopted for this research (Lee & Baskerville, 2003). The two case studies were treated

as separate units to apply the adaptation framework to their individual product development environment and to identify their adaptation factors. Hence, two case studies were deemed adequate for this in-depth research to provide the necessary understanding on agile adaptation factors.

The Type TE Generalizability was most appropriate for this study because of the nature and purpose of software development. There were different types and sizes of software development teams, influenced by different factors. Hence, there could possibly be different adaptation factors for software development teams and organisations. For this reason, the adaptation framework was applied to a particular setup to determine its adaptation factors only.

3.7 Data collection skills

The strength of qualitative research is dependent upon the researcher's skill, intellect, discipline and creativity (Patton, 1990). The four key items that were considered for this study in preparation for data collection were desired skills, training, case study protocol and pilot case study (Marshall & Rossman, 1999; Yin, 1994). However, a pilot case study was not possible because of the limited number of cases available for this study.

Yin identifies several key elements for researcher skills such as question-asking, listening, flexibility, grasp of the issue and lack of bias as part of interviewing techniques. The tactic used to ask questions involved documentation of the questions in advance to be asked. This guided the interviewer to collect comprehensive data, for systematic data collection and to have focused interview sessions maximizing the interviewee time (Patton, 1990). The relevant questions were constructed based on observations of the methodology-in-action in the participant's production lab.

Listening was another of the essential elements of the interviewing technique used for this case study research. This enabled observation of the respondent behaviours during the interview session helping to gauge the impact of a practice or its adaptation. It allowed addition and modification of the questions to probe a new issue or deeper into issues but within the case study protocol.

Flexibility was the other element adopted with this research. Flexibility allowed for minor modifications of the research design as understanding developed through

interviews. Changes to sub-research questions were made as a deeper understanding of adaptation factors emerged.

Getting familiar with the agile approach for software development including prescriptive methods was the most important aspect of this study. This familiarisation involved reading published materials and books to get a good understanding on these methods, especially how these methods differed from one another. Attending agile methods conferences and seminars, and observation of the two case study participations production lab greatly enhanced the practical understanding of the agile methods.

The other important aspect was getting familiar with the theoretical framework to be able to ask relevant questions. A better understanding was gained through discussion of the adaptation framework with prospective case study participants. Discussions were also held with the senior managers of the agile development teams of two case study participations which had further helped the understanding of the adaptation the framework.

Another tactic employed was that a presentation was done to one of case study organisation's agile development team regarding the study and on the adaptation framework, which was also followed by question and answer sessions. This presentation had further helped to understand the adaptation framework and its relevance. The tactic included a week of observation of their methodology-in-action in their production lab. This case study innovation served as an alternative to a pilot study for gaining familiarity and to ensure rich agile adaptation data would be available.

The study adopted the openness strategy of reporting any unusual or contrary findings to avoid bias reporting. Such findings were further discussed with appropriate individuals from the two case study organisations to validate the interpretation. It allowed for reporting findings such as cost of choice factor with agile adoption for individuals under the raise the profile of IS department factor.

3.8 Study protocol

The study protocol involved creating procedures for data collection and organisation to facilitate analysis and report writing.

3.8.1 Data collection

The primary source for data collection was the interview technique. Observation was also used to determine the individual(s) to interview on a particular practice or issue. Discussions were held with the managers of both participating agile teams to determine the possible candidates. However, individual participations were voluntary. Table 6 and Table 7 provide a separate list of participants from the two case study organisations who took part in the study.

Table 6 List of participants from case study one (Akldevelopment)

Role in the agile team	Company experience as at 2006)	Key responsibilities
Engineering manager	3 years experience with company but a total of 15 years of development experience including substantial agile development experience	Responsible for all project deliverables of the agile team. Works closely with product strategy group and senior executives of the company to determine new feature developments. Co-located and takes part in all the development activities of the agile team. Other activities include field trips (overseas client sites), working with sales group to host potential clients and taking part in sales negotiations.
Senior product analysts (was product manager)	15 years of development and product management experiences with the company	Part of product strategy group but work full-time in agile development team. Plan and manage product backlogs, write user stories (development-task), write acceptance test, test implemented stories, provide domain support for engineers and plan sprint cycles.
Product analysts	5-6 years of development experiences with the company	Part of product strategy group but works full-time in agile development team. Plan and manage product backlogs, write user stories, write acceptance test, test implemented stories, provide domain support and plan sprint cycles with engineers.
Principal engineer	1 year with the company but 20 years of development experiences including 6 years of agile development experience	Provide team and technical leadership, and team coaching. Provide input into vision plans. Take part in product backlog and sprint planning sessions and scrum meetings. Implement user stories in sub-teams.
Senior engineers	6 or more years of development experience with the company	Provide team and technical leadership, and team coaching. Provide input into vision plans. Take part in product backlog and sprint planning sessions and scrum meetings. Implement user stories in sub-teams.
Engineers	4 or more years of development experience with the company	Take part in sprint and scrum planning meetings. Implement user stories in sub-teams.

Table 7 List of participants from case study two (Meldevelopment)

Role in the agile team	Company experience as at 2006)	Key responsibilities

Director of software engineering	10 years of development experience with the company	Responsible for all project deliverables of the engineering department. Works closely with product planning group and senior executives of the company to determine new product developments. Co-located and takes part in all the development activities of the team.
Product manager	6 years of development and product management experiences with the company	Part of sales and marketing group, co-located and works closely with project mangers and development teams; provides high-level requirements. Plan and manages vision and roadmap plans. Takes part in design phase with engineers.
Project managers (became product development managers)	10 years of development and project management experiences with the company	In-charge of projects and are official team leaders. Work closely with product mangers to implement vision plans. Plan and manage product backlogs, provide domain support, plan iteration cycles and take part in daily stand-up meetings with engineers. As product development managers they also work in field (product management function).
Principal engineer	7 years of development experiences with the company	Provide team and technical leadership and team coaching. Provide input into vision plans. Plan product backlogs and iterations, and attend daily stand-up meetings. Responsible for product architecture and enhancement of products. Implement tasks in sub-teams.
Senior engineers	2 to 20 years of development experience with and outside the company	Provide team and technical leadership, and team coaching. Provide input into vision plans. Plan product backlogs and iterations, and attend daily stand-up meetings. Implement tasks in sub-teams.
Engineers	2 or more years of development experience with the company	Plan product backlogs and iterations, and attend daily stand-up meetings. Implement tasks in sub-teams.
Quality assurance manager	7 years of company experience	Overall responsibility for quality of product releases. Works closely with project managers, product manager and documentation team.
Quality assurance engineers	2-4 years of company experience	Test iteration builds, take part in design phase (backlog planning), support software engineers to implement unit tests and coach them with quality assurance activities.
Communication engineer	Over 10 years of company experience	Provide documentation for releases. Coach and provide information to the engineers on usability issues. Take part in design phase.
Marketing manager	2 years of company experience (substantial marketing experience outside the company)	Provide product information to the engineering team. Attend stand-up and design meetings.

The participants who took part in the study were co-located with their agile teams. Case study one had made the quality assurance team redundant on their agile adoption and the documentation team worked separately from their development teams. The study participants from both case studies were agile method users and were empowered to

adapt their methods. Senior executives of both case study organisations were not requested for the interviews because of their busy schedules and field commitments. Likewise, the usability engineer with case study one could not be interviewed.

The following were the interview instruments or key themes based on which individual interview questions were formulated; product development prior to agile adoption; agile method adoption, adaptation and agile culture; planning practices; product backlog practices; test-driven practices; development setup, teams, roles and skill set; short development cycles and project sizes; integration practices; code development practices; domain knowledge support; development tool support; empowerment and productivity practices, training and learning practices; and communication and interaction practices.

Each interview session was planned for an hour and an additional session was scheduled if an interview was not completed in a session. Interview sessions were held in the meeting rooms at the production lab of the case study participants. In total forty sessions were held per case study organisations. Table 8 and Table 9 provide a separate list for the two case study organisations indicating number of interviews held with each participant.

The requests for interviews were made through the managers of the agile teams of two participating organisations. A timetable for interviews was agreed with both organisations. Initially, it was planned that all interviews were to be done within a six month period. However, this was extended to over a year because of the changes in product development commitments of both teams.

The following principles for interview technique were applied as suggested by Marshall & Rossman (1999), Yin (1994) and Patton (1990). A semi-structured interview method was used allowing the interviewees to explain their method practices. Observation was used to see methods in practice (achieved data triangulation). The best possible informants were used and interviews focused on method practice rather than personal belief. Questions were formulated as single questions. Interviewees were informed of their right not to answer any particular question they were not comfortable with. Participant anonymity was also ensured. The impact on their work schedule was minimised by having interviews early in the mornings, late afternoons or during lunch time (as convenient to the individual), and limiting the interview sessions to one hour at a time

Secondary sources for data collection were also employed. Interviews done by company executives for business publications and made available on the web were also used. These were validated with the team manager. With both case study organisations, it was also agreed that the case study report rather than individual interviews would be validated to minimise impact on their development commitments. Agreement was made on recording of interviews and individual consent was taken before a session began. These recorded interviews were later transcribed. Recording enabled accuracy for data collection and to be more focused on the interviewee (Patton, 1990). However, important notes or emerging questions were noted down during the session.

The interview questions were knowledge questions about their agile product development environment, development practices and related adaptation. Interview questions also included background questions relating to their organisation and previous development environments and development problems (Patton, 1990).

Prior to an interview session all questions were read and revised. Questions were also read out to colleagues to ensure their clarity. Any probing or follow-up questions resulting from the previous session were included.

Table 8 Number of interviews per participants at case study 1 (Akldevelopment)

Role in the agile team	Number interviewed	Interviews per Individuals	Total interviews
Engineering manager	1	9	9
Senior product analysts (was product manager)	1	2	2
Product analysts	2	2	4
Principal engineer	1	5	5
Senior engineers	6	2	12
Engineers	4	2	8

Table 9 Number of interviews per participants at case study 2 (Meldevelopment)

Role in the agile team	Number interviewed	Interviews per Individuals	Total interviews
Director of software engineering	1	7	8
Product manager	1	1	1
Project managers (became product development managers)	2	3 & 4	7
Principal engineer	2	2	4
Senior engineers	4	2	8
Engineers	2	2	4
Quality assurance manager	1	2	2

Quality assurance engineers	2	2	4
Communication engineer	1	1	1
Marketing manager	1	1	1

3.8.2 Data organisation- case study database

A process was adopted to create a repository for all the data collected for this research. This repository facilitated systematic organisation of data from various sources into categories to scrutinize data and make sense of it (Creswell, 1998). Systematic data organisation established a chain of evidence linking research question, empirical evidence and analysis approach used (Maxwell, 2005; Lincoln & Guba, 1985). This was done to ensure research validity.

Each interview was tape recorded, labelled (given a number and job title) and transcribed. The transcribed interviews were saved as Microsoft Word documents using tape number and interviewee title. This labelling allowed for quick access to the appropriate tape to validate transcription with audio. Separate folders were created for two case study organisations to save the transcribed interviews as Word documents. Any other relevant information (electronic) was kept under respective folders.

3.9 Unit of analysis

The unit of analysis to investigate agile adaptation was identified as the agile development method. The main item that helped the researcher to decide and define the appropriate unit of analysis was based on what the researcher was able to say about the investigation at the end of this study (Patton, 1990).

The choice of the agile software development method as the unit of analysis was based on the main research question. The research question defined the focus of this investigation to explain and provide understanding of the factors that lead to adaptation of these methods.

For this research, it was expected that the agile methods used by development teams would be different, influenced by different factors. The primary focus of data collection was on the factors that affect method adaptation in an agile software development environment. Key individuals directly involved with product development were sources for information on these adaptation factors.

3.10 Data analysis

Data analysis enables a researcher to scrutinize the data to make sense out of it, rule out any other possible interpretations, and to put together appropriate conclusions (Miles & Huberman, 1984).

There are several approaches for data analysis such as building case study data favourable for statistical analysis, creating a template of categories, recording the frequency of different incidents, doing statistical calculations to determine relationships, and using methods to classify information (Miles & Huberman, 1984; Pelz, 1981). Yin (1994) suggests using the theoretical propositions that are used to investigate the research problem.

This study employed the theoretical propositions approach using the adaptation framework to guide data analysis involving applying the adaptation framework to identify agile adaptation factors. The data analysis involved describing the data in a meaningful way by using adaptation components identified in the framework including the background of the case study to answer "how and why" research questions, which were the sub-questions to the main research question about agile adaptation.

This study used the coding scheme approach (Creswell, 1994). Chapters 4 and 5 present the two case studies using meaningful descriptive data, referred to as "thick description", for the reader to understand and to draw their own interpretations (Denzin, 1978; Patton, 1990).

This research used the Nvivo software tool (qualitative analysis tool) to organise and analyse the large amount of data. Separate projects folders were created for the two case studies. This was done since each case study was treated as a single unit for making generalizations. The transcribed interviews and other information saved as Word documents were imported into Nvivo.

Nvivo allowed the organising of data into various categories based on the adaptation framework (theoretical framework) and using new adaptation categories as they emerged by enabling data to be merged from different documents. Nvivo also allowed the flexibility to rearrange data as analysis proceeded and as new categories emerged.

For each case study, data was organised and analysed in categories according to Fitzgerald's (1998) adaptation framework (Figure 2) using methodology-in-action, development environment (organisational) factors, overt factors, and covert factors and their sub-categories. An additional category, company background, was included and further sub-categories emerged as analysis proceeded; these are identified in Chapters four and five. Similarly for methodology-in-action further sub-categories emerged as analysis proceeded, which are also identified in Chapters four and five. Figure 4 identifies the four main categories adopted from Fitzgerald's adaptation framework (methodology-in-action, development environment factors, overt factors, and covert factors) which were used to organise and analyse data. Figure 4 also includes the company background category and the sub-categories associated with methodology-in-action, which emerged during analysis.

The template used to collect data, to provide case study description and to do analysis is based on the adaptation framework. Hence, a single model strategy is used.

The use of multiple models or theories requires a more demanding research design for different types of data collection (Allison, 1971). While this approach is often regarded as providing more accuracy and explanation of interesting events, often it leads to awkwardness and unsatisfying results (Langley, 1999) and development of a complex theory (Eisenhardt, 1989).

Besides, the results are complementary when multiple models are used (Baskerville & Pries-Heje, 2001). Hence, the single model was considered to be most appropriate to identify the agile method adaptation factors as accurately as possible.

The use of a single model does not overwhelm and confuse the software development community with different models, and with non-practical and varying constructs. Fitzgerald's adaptation framework provides constructs and an analysis model, both relevant to theory and practice; it is an empirically grounded framework. This enables research convergence, hence achieving both theoretical and practical contributions (Markus, 1997), while maintaining simplicity (Eisenhardt, 1989).

Description and interpretation are two different tasks; interpretation activities followed after describing all the data in a meaningful way (Patton, 1990). Data interpretation (in-

case and cross-case analysis) answered the main research question “how does adaptation work in an agile approach”.

Patton points out that interpretation involves explaining the findings and attaching significance to particular results, and putting patterns into an analytic framework. The pattern matching technique was employed to explain the adaptation factors identified through empirical data using the key adaptation literature. This was used to confirm the internal validity of the research (Yin, 1994; Patton, 1990; Eisenhardt, 1989).

Creswell (1998) suggests that data analysis is a spiral activity (refer to Figure 5). This approach emphasizes that data analysis should take the following steps: (a) data is organised (b) data is scrutinized to make sense of it, (c) data is classified according to identified categories, and (d) synthesis- integrate; a summary that may include tables, figures etc to interpret the data. Using the approach described by Creswell, data analysis was done in a systematic manner as shown in Figure 5.

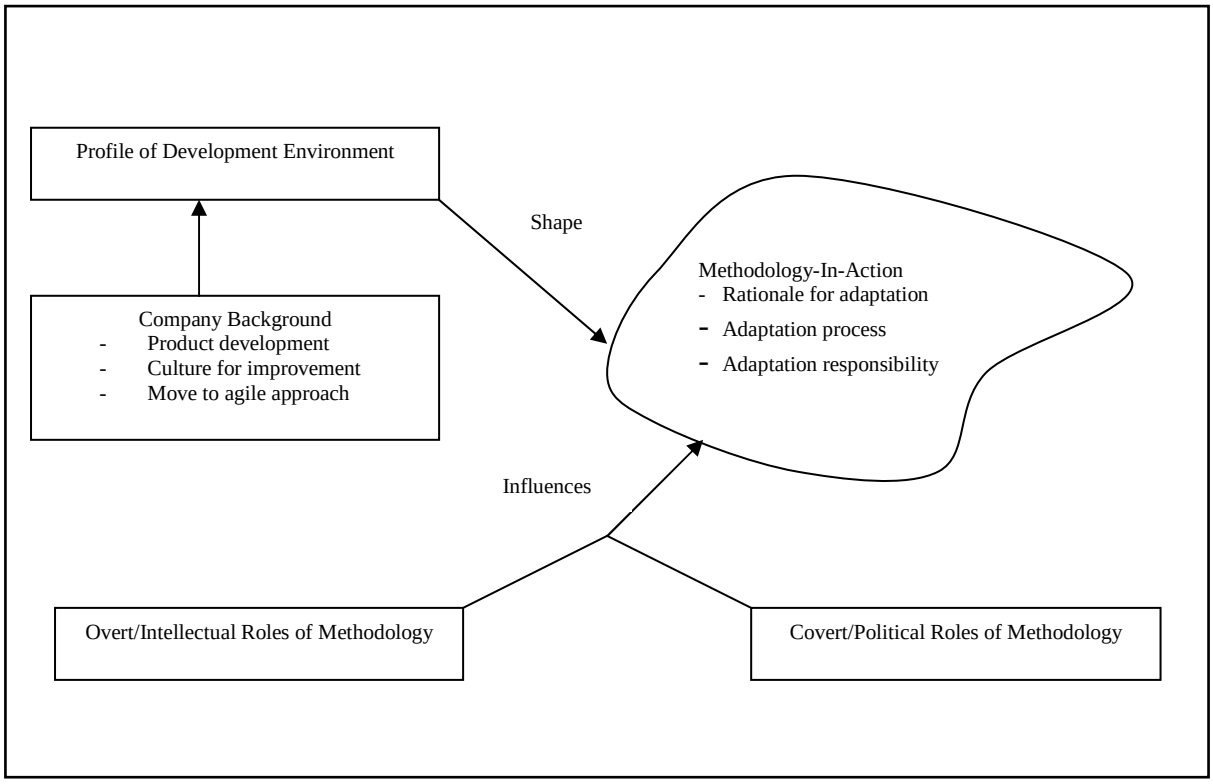


Figure 4: The main categories used to organise and analyse case study data.

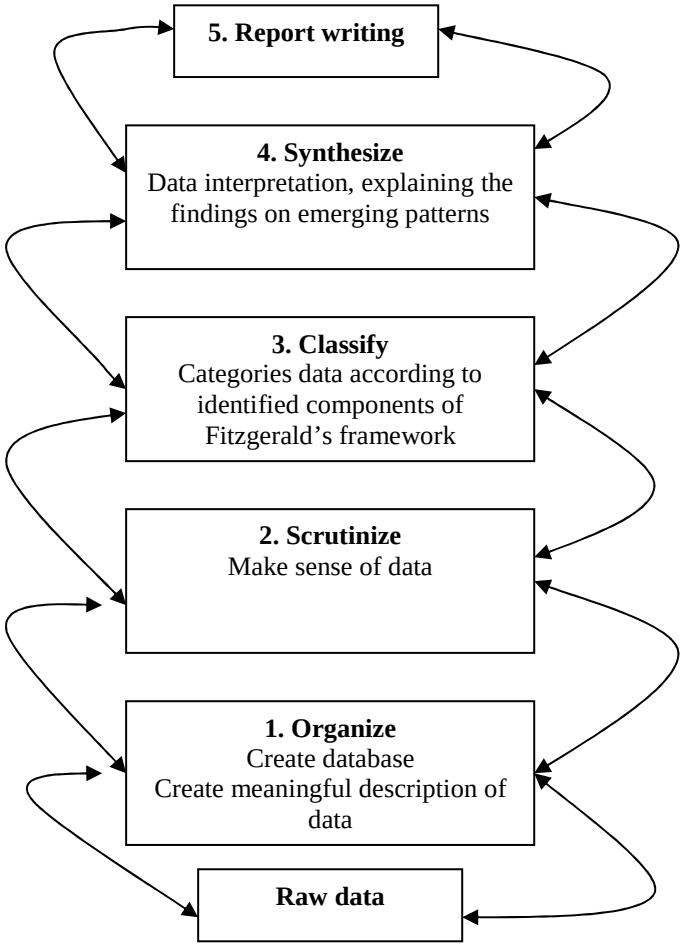


Figure 5: Data analysis approach based on Creswell (1998)

The identified steps for data analysis were applied first to write up individual case studies and in-case analysis for both case study organisations. The in-case analysis identified and explained the adaptation factors. Cross-case analysis provided a perspective on agile method fragments and practices that are employed by the two case study organisations.

Patton suggests two tests for components used to categorize data: (a) internal homogeneity- the extent to which the data that belongs in a certain category fits together in a meaningful way and (b) external heterogeneity- the extent to which the disparity among categories is bold and clear.

If a large amount of adaptation data was not able to be classified into any of the categories of the theoretical framework, it would have indicated a fault in the category system (Guba, 1978). According to Guba it would then have required working back and forth between the data and classification system to verify the meaningfulness and accuracy of the components and placement of data in components.

To flesh out the patterns, Guba suggests the following: (a) build on information already known, (b) make connections with different information, and (c) recommend and confirm new information that will fit in. Patton (1990) suggests that this process finishes only when it is deemed that: sources of information have been exhausted, new sources lead to redundancy, and the investigator starts to go over the boundaries of the guidelines used for analysis. These tests, listed in Table 10 were applied to check for completeness.

Table 10 Test for completeness

Tests	Description
1. Integration	Internal and external plausibility: (a) internal-individual components that identify different types of adaptation factors should be consistent and (b) external- the set of components should provide the complete picture of the adaptation of the agile method.
2. Inclusive	The components should be realistically inclusive of adaptation data that has been collected- the components of the framework should be able to match adaptation data from all cases that will be investigated. The other test is to match the components with the agile adaptation research questions and if there appears to be discrepancies between these two then the adaptation information about agile methods will be deemed to be incomplete.
3. Replication	Similar analysis results should be achieved by another independent capable investigator. The judgment that components of the framework, that identifies adaptation factors, make sense and data on adaptation has been appropriately classified into various components is important.
4. Credible	The complete set should be convincing and realistic to the participants who provide the adaptation information.

Adopted from Patton (1990, pg 404)

3.11 Quotation and description policy

Once relevant data was organised into various categories, they were reduced to use them as quotes to identify and describe adaptation factors for each case study. Prior to reduction of data the researcher thoroughly read the data organised under each category to gain clarity and to eliminate any preconception or bias. This required reading the interview transcripts and in some cases listening to the taped interviews to cross-check the interpretation. The following steps then were used to reduce the data and to write the description of the adaptation factors: (1) locate key phrases and statements that related to an adaptation factor, (2) interpret the meanings of the phrases and statements, (3) scrutinize the meaning of what they reveal about the adaptation factor, (4) write the related description and support it with quotes (reduced data) and (5) get review and

feedback on case description from the respective case study organisations to ensure accuracy of the interpretation. As a result, a minor change was made to the background description of a case study organisation.

3.12 Study confidentiality

A non-disclosure agreement was signed with both case study organisations to protect their identity. This was to protect any sensitive information relating to their technical knowledge of products and on personal data such as remuneration packages.

The non-disclosure agreement was an important factor for these two organisations agreeing to be part of this research. Therefore, this research has adopted fictional names; Akldevelopment for case study one and Meldevelopment for case study two. These names were agreed upon with the two participating organisations.

As part of the non-disclosure agreement, the case study reports were validated by the respective organisations. As a result minor changes were requested by both organisations which were incorporated. A copy of the final thesis will be delivered to both.

3.13 Case study report

This research adopts a single narrative to describe, analyse and report the two case studies separately (Yin, 1994). This format has been adopted considering the audience for this report; academia and the software development community (Patton, 1990).

A linear-analytic structure was adopted; a common and standard approach for writing research reports (Yin, 1994). Hence, the following chapters are the different parts of this thesis: chapter one- problem identification; chapter two- literature review; chapter three- research methodology; chapter four- description of case study one; chapter five - description of case study two, chapter six- in-case analysis of case study one and two and cross-case analysis; and chapter seven- conclusions and future research.

3.14 The study (methodology) model

The study model, represented in Figure 6, identifies the main phases of this research. Understanding the background to the research problem was done through the literature review. The literature review also helped to identify key literature on agile adaptation

and work processes including an appropriate adaptation framework. This enabled the formulation of research questions.

Presenting in departmental conferences and seminars and international doctoral consortia facilitated the review and refinement of research question and further exploration of adaptation literature.

The research question led to the adoption of the case study method as the research methodology. Discussions with possible case study participants on the appropriateness of the adaptation framework also assisted in identifying appropriate case study participants.

The case study protocol was developed and documented for data collection using multiple methods. The study protocol was applied to organise data in the research database. Separate case study reports were written based on the adaptation framework. In-case analysis and cross-analysis were done based on the adaptation framework. Finally, the study conclusion identified the adaptation factors of each case study.

3.15 Summary

This chapter provided a detailed description of the methodology used for this study and provides information on the appropriateness of using a qualitative case study research method. This chapter also includes the research design and measures taken to ensure research quality.

The next chapter presents the first case study for this research.

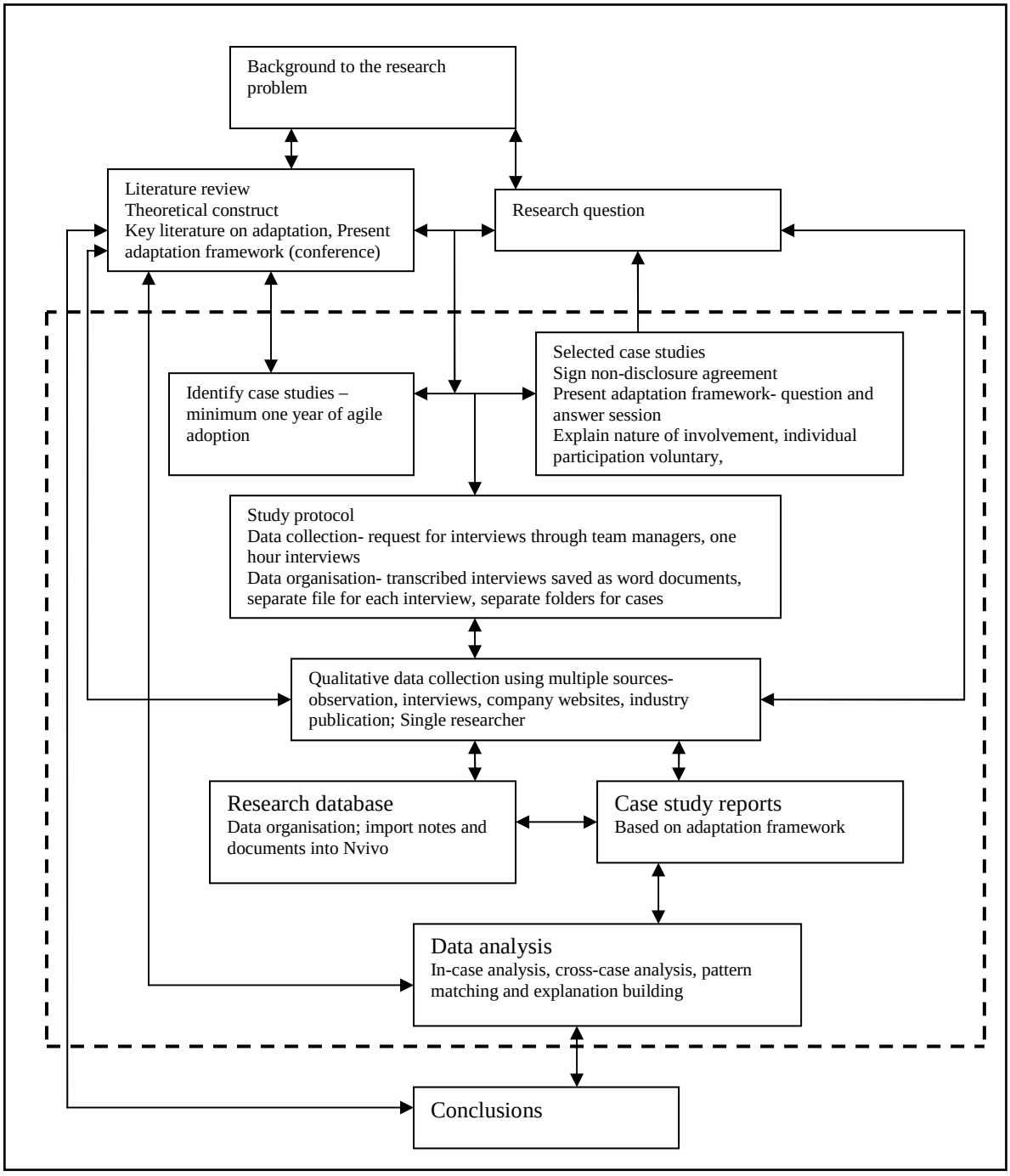


Figure 6 the study model showing the research methodology

Chapter 4: Akldevelopment case study

This chapter presents the Akldevelopment case study. First, the company overview is provided. It is followed by discussion on Akldevelopment's agile method adaptation. The data used to construct the case study were collected through a series of interviews held at Akldevelopment's development centre in Auckland. Other sources of information include the company website and online industry publications about Akldevelopment and its software products.

4.1 Overview

This section provides information on Akldevelopment's business background, its past product development environment and its agile adoption.

4.1.1 Company background

Akldevelopment is recognised as one of the foremost international software vendors with its highly strategic customer management product. Founded in the early 1980s in Auckland, Akldevelopment started off as a computer vendor. In 1987, New Zealand was the first in the world to deregulate its energy sector and Akldevelopment quickly seized the opportunity to develop a strategic business application for customer management for companies entering the deregulated market.

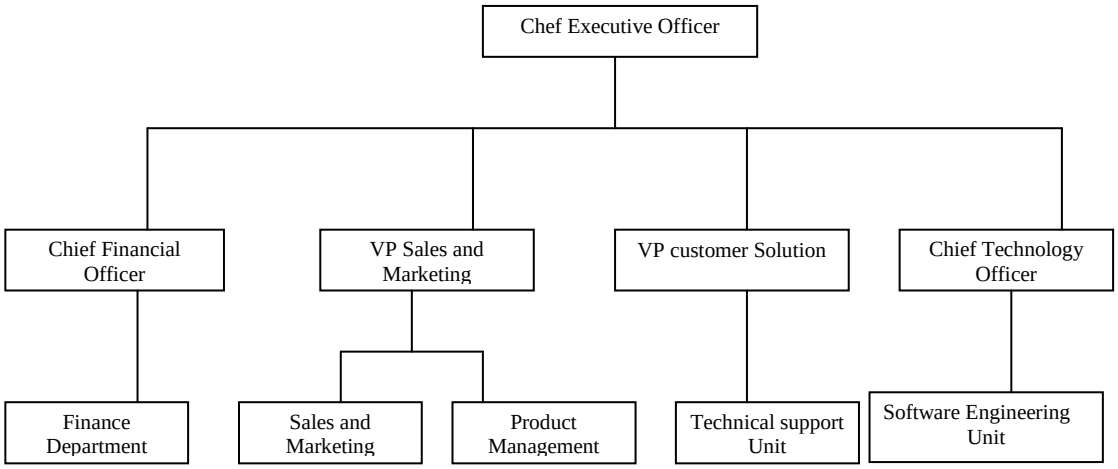
Akldevelopment became the domestic market leader for providing customer relationship management software. This product also provided the company with a great opportunity to enter international markets, as governments worldwide started to deregulate similar market sectors. The ability to compete extensively in the international marketplace led to the company opening offices in the United States, Australia, Canada, and the United Kingdom.

The substantial growth in the US market led the company to shift its head office there in 2002, close to its major customer base. Continuing product success was dependent upon their ability to interact with clients to discover and understand their requirements, and to deliver innovative products.

The executive management structure of the company is as follows; Chief Executive Officer, Chief Financial Officer (leader of the finance department), Chief Technology

Officer (leader of the engineering department), VP Sales and Marketing (leader of sales, marketing and product management teams) and VP Customer Solutions (leader of technical support team) (Figure 7). The Auckland office is their product development centre. When this study began in 2006, Akldevelopment had about 170 employees and nearly eighty percent of them were software engineering personnel.

Figure 7 the executive management structure of Akldevelopment



Akldevelopment has some of the world's largest corporations as its clients. Its clients have a customer base in the millions. One of its clients was signing up to a thousand new customers per day - thus product scalability was one of their prime quality concerns. At present, Akldevelopment's product has outgrown the local market requirement (a customer base of only hundreds of thousands) and New Zealand companies are regarded as relatively small users of their product.

Akldevelopment's product suite enables large corporations to set up automated customer management systems. It is recognised by industry experts to have a competitive advantage in scalability and flexibility with new technologies, ease of integration with legacy systems and speed of implementation.

Akldevelopment works closely with industry leading firms to ensure that they gain recognition as the vendor of the best software product in major international markets. In their list of global partnership networks are IBM and Oracle.

Akldevelopment was placed on the Software 500 in 2002; a list compiled annually by Software Magazine to rank the world's foremost software providers. The company

records show that it had a revenue growth of 150%, outpacing its major competitors. The industry specialists selected Akldevelopment with three others amongst 50 similar organisations as the leading vendors in the marketplace.

Akldevelopment's mainstream business is focused on growing their product suite by adding innovative features and increasing its capabilities with emerging technologies to manage extremely large number of customers. Their mainstream business also includes continuous enhancement of product quality in areas such as usability and performance. Their product delivers business value by cutting client costs through reduction in staff training time, increasing staff productivity and improving customer satisfaction.

4.1.2 Product development

With a small development team, Akldevelopment developed and sold its first product to a local company in 1987. The development success was attributed to the company hiring individuals with extremely high technical skills. In 1995, their development team implemented a self-service feature for their product based on internet technologies, making Akldevelopment the first software vendor to make such a feature available with a customer management product. Such success led to substantial growth of its development capacity. Despite shifting its head office to the United States, the company maintained its production lab in Auckland. In 2002, Akldevelopment had 300 developers based at its Auckland Office, organised into separate development teams. The following comment was made by the company's Vice-President Development in 2003, for an on-line publication with regard to their engineering talent.

> New Zealand is the right place … high calibre of IT people located here. It is their expertise, teamwork, motivation to succeed and global awareness that differentiate us on the world stage. (Vice-President, Development)

As the business grew, Akldevelopment boosted their development capacity. While high calibre engineering talents were sought both locally and from overseas, Akldevelopment had also put a strong focus on recruiting talented IT graduates from New Zealand universities. The company had established a strong association with one of the leading universities in Auckland since the early 1980s.

> Strong technical graduate base within New Zealand … adaptability and coherent and eloquent english communication [skills] … those are advantages to us. Staff are recruited and brought to

> the centre from around the world … maintains strong ties with New Zealand educational institutions. (Organisational Development Manager)

Next, some of the key features of Akldevelopment's past product development environment are described.

4.1.2.1 Specialised-skilled individuals

The earlier development work was guided by the life-cycle development approach. Later, most of their teams opted for the Rational Unified Process (RUP) development method with specialized roles. They had a clear separation of duties amongst the engineering individuals in teams, with the various job titles reflecting the tasks that an individual was responsible for. These roles were project managers, architects, analysts, designers, programmers, technical writers and testers.

> We had project managers, architects, analysts, designers, implementers, technical writers and testers … heterogeneous team. (Engineering Manager)

Akldevelopment expected their development teams to speedily react with innovative product features to meet individual client needs. To achieve this, the company not only focused on growing individual technical skills but also emphasized team work; the ability to work with others within the company and external collaborators was very important. Akldevelopment facilitated engineer exposure to new technologies through frequent visits of teams from IBM and Oracle, including the clients. Engineering individuals were also provided with the opportunity to work worldwide with the implementation teams at the client sites. The following comment was made by the founder of the company for an on-line publication.

> Leverage innovation and teamwork … retain the development centre in New Zealand where we can draw on innovation … as we push into new countries, our people can travel and grow with the company … [gain] experience … developing their IT skills. (Akldevelopment Founder)

Some key business roles emerged at Akldevelopment such as sales, support, marketing, and product management. These roles required an in-depth knowledge of their product. Of these three positions, the product manager role became important to the development function. The product manager role at Akldevelopment took responsibility for identifying the market requirements for current and future products through market research and external stakeholder collaborations, and also for the writing-up of the project specifications.

> Product managers do research, find out what the requirements are and what we need to have in the product … talking to current customers … our consultants … read the xxx magazine …

> articles … read papers … spend months out in the field, do things like conference calls, shows and tells. (Product Manger)

4.1.2.2 Large development teams

The development effort at Akldevelopment was organised into project teams. The number of teams had varied over the years. At any given time, Akldevelopment had between six to ten teams working on different projects. Some teams were assigned to customize the product for their clients. At Akldevelopment, the number of individuals in a team was based on the project size. They had large teams with each having a team membership of up to 15 development individuals. Akldevelopment had a pool of engineering managers and they were assigned to lead a development team and the project the team undertook. Mangers made sure that their team's delivery commitments were met.

> Team sizes were between 6 and 15 depending on the project. We didn't have project managers … called the guys engineering managers … 11 of them. (Engineering Manager)

4.1.2.3 Quality assurance team

Akldevelopment had a separate dedicated quality assurance team for undertaking manual testing to ensure the extremely high quality and completeness of their new product features. The quality assurance team provided certification for the functionality, usability, reliability, performance, and scalability of their products. They made certain that Akldevelopment's product was compatible with and ran smoothly on a client's hardware and software platforms. The team raised the quality standards of their products, enabling Akldevelopment to launch its product into international markets with their specific needs.

> There were about 10 testers in the QA team … whole bunch of testing at the end before product release. (Engineering Manager)

4.1.2.4 Documentation team

Similarly, a separate documentation team was established, mostly for the production of technical documentation; it developed manuals and online support for their software product. It was not practical for Akldevelopment to dedicate a technical writer to each development team since the company had a major release every eighteen months. This small team was sufficient to handle all their documentation work. Akldevelopment considered documentation an integral part of their software product that enhanced its overall quality.

> Had technical writers … there were about 4 back then, one person full-time and the rest contractors. (Engineering Manager)

4.1.2.5 Development challenges

Akldevelopment's engineering teams always had some major development challenges. One was to keep up-to-date with regulatory changes in different international marketplaces; these were the emerging requirements. Any regulatory changes had to be captured as requirements and related features updated immediately. Some of the new regulations were mission-critical having huge legal and moral implications for Akldevelopment's clients. The continued distinction of Akldevelopment's product was dependent upon their capability to identify and implement solutions for all the related regulatory change scenarios.

> Keeping up with regulatory change … anything that goes bad, people dying … our systems have coped with that particular scenario … not a new scenario … but they alter the regulations slightly it has different impacts. (Product Manager)

Another development challenge was providing for product adaptability to deal with emerging technologies. The advances in communication technology provided opportunities for customer management systems to be incorporated with new tools offering higher levels of performance. To maintain its leadership in the marketplace, Akldevelopment needed to provide for product adaptability with emerging tools to create major benefits for its clients.

> Making sure you can flexibly deal with communication technology … things like what used to require you talking to a person can now be done through IVR [Interactive Voice Response] systems … that's a very important one. (Product Manager)

With regards to emerging technologies, adopting web services architecture posed another development challenge. This required delivering strategic product capabilities such as features to link the client's business function with an appropriate third party concern. This kind of product capability enables their clients to access the customer information enabling them to make sound business decisions and to cope well in a modern economy driven by technological advances such as internet banking.

> Credit checking customers … know up front if they've got to turn someone down … no point signing up someone who is not going to pay the bills … getting that sort of thing early enough … needs to come through your web services as well as UI. (Product Manager)

A further development challenge that emerged over time at Akldevelopment was the lack of local customer collaboration on new features. Akldevelopment had mainly off-shore clients (their product became too big for local organisations), the easy access that they previously had with local clients ceased. This had an impact on the engineering team's ability to acquire a first-hand understanding of the actual needs of the end users. Previously with local clients, the development individuals spent time at the client-site. This involved doing consultation or technical support work to get experiences of the day to day business activities of clients such as how tasks were carried out, knowledge of the different user roles and the ways of thinking associated with each user role. Hence, the engineers understood the needs and priorities of clients much better.

> Had New Zealand clients, spent a lot of time with them … changed to having off shore clients, our engineering pool got less exposure to the clients … there is a big gap in comprehension. (Senior Product Analyst)

4.1.2.6 Methodology adoption

Akldevelopment's development teams had always attempted to adopt an appropriate method that best suited their work and for overcoming the challenges that they frequently encountered. From the team's perspective, the development method was important since it made their work more visible and transparent, helped to determine the resource requirements at various points during their projects, and enabled them to meet quality standards and productivity requirements.

> Methodology has to address … an upcoming release … can tell people exactly what will happen … identify the work that needs to be done, the scope, number of people needed … estimation, who should be participating in which activity … defining roles … identify tools fit …techniques fit … figure out, should we be doing or avoiding certain things. (Engineering Manager)

The method adoption decision was entirely left to the development teams, where senior individuals played a significant role in making the adoption decision. From the Akldevelopment perspective, the development teams had to deliver against an agreed upon set of features, regardless of the development method being used, and were judged based on delivery and the quality standards of their product.

> We choose whatever is suitable for our needs … [if] do not perform, would certainly lose that freedom … as long as we deliver good results, can choose whatever we think is appropriate. (Engineering Manager)

4.1.2.7 Development problems with their method

Akldevelopment felt that their development process had some issues that were impacting product development. Listed below are two of the major ones.

4.1.2.7.1 Unreasonable schedules

With their structured development approach, the project teams were usually not in a position to determine reasonable estimates to negotiate delivery commitments. This was normally experienced by the teams when dealing with customer change requests as they had fixed engineering resources for new projects. Each successful sales deal at Akldevelopment was subject to major customisation. Often it was based on a fixed price and get out clauses, adding extra pressure on the engineering teams to deliver a customized product within the agreed timeframe.

> Certainly pressures meeting customer demands … each sale there's multi hundred hours change requests … that itself adds pressure, have to deliver those changes … thousands of hours of costs. (Senior Engineer)

4.1.2.7.2 Undesirable requirement definition process

Another major development problem which Akldevelopment frequently encountered with their structured approach was that it would usually take longer than expected to document the requirements and get them reviewed for development. At Akldevelopment, it required producing upfront the following documentations prior to any implementation; the user requirements, the functional requirements and the detailed design specifications. In addition, rigid requirement definition and design phases created issues for incorporating any requests for requirement changes.

> More requirements change, harder to implement … often new requirements come up. Spend a lot of time documenting requirements up front … coming up with a solution … then the design, and coding … through all of that there is an awful lot of review in things … all becomes very inflexible. (Product Manager)

4.1.3 Culture for improvement

As a highly innovative company, Akldevelopment was driven by a culture of trying to maintain its status-quo as a market leader. They also have some major competitors. As result of development challenges and problems, a decision was made by their senior executives to improve the technology used for their product development. In particular, an improvement to the development approach was desired that would lead the company

towards having more frequent product releases, would improve the overall quality of the product without severe overheads and with gains in the productivity levels of their engineering individuals. Akldevelopment also recognized that the way forward for any future product growth and development was dependent upon the readiness of the company to adapt to the changes happening in their business environment and as result of technological advancements. This required the culture for accepting change and having the ability to adapt faster than their competitors.

> You need to step ahead … have some very good competitors … moving forward … need to market and propose a good solution. (Product Manager)

> Introduce a culture that is built around change … 3 reasons, time to market, quality, productivity … had major release every 18 months, release a new version every 4 months … substantial benefit in time to market … wanted to improve quality … zero defects as a quality gauge … third one is productivity ... anecdotal data, some indication to the change around productivity. (Engineering Manager)

4.1.4 Move to agile approach

In 2004, Akldevelopment hired a Technology Transformation Manager to implement the agile approach for product development. One of the major changes in relation to agile adoption that was carried out was abandoning the "New Zealand flavour of creating software" using a single development cycle to a progressive and productive approach involving multiple development cycles. The development teams embraced version control systems (VCS), for an automated management of their source code repository. A Test Driven Development (TDD) framework with a continuous integration approach was adopted with the VCS to achieve more frequent product releases. With adoption of TDD, the separate quality assurance team was made redundant; functional (unit) tests are done by developers during code implementation, and the systems (acceptance) test is also done by the developers prior to releases. These changes, with agile approach adoption in mind, were integrated by all the development teams with their structured approach.

> The intention is to have all the teams in the engineering environment to have core steps … basically an agile approach … includes not only the processes but also tools, architecture, design, technology … move all teams over the time into that environment. (Engineering Manager)

In early 2005, a separate agile team was created. This agile team was led by the technology transformation manager as the team's development manager. The

individuals were picked from the other teams by the senior management. These individuals were mostly programmers with development experiences ranging from 3 to 4 years. They were given the new job title of software engineers. The position is assigned with responsibility for undertaking all tasks relating to product development rather than just coding. The other specialist individuals that were included in the agile team as software engineers were the performance experts. The product manager role was brought forward into the agile team to drive the agile team projects and to provide domain expertise.

> Team was created in April, 2005 … had in total 24 head count … software engineers [18] … also had 3 performance experts and 3 product managers … [performance experts] were essentially like engineers … pairing and doing stuff. (Engineering Manager)

The other development managers at Akldevelopment were empowered to make agile practice adoption decisions for their development teams. They were also given the opportunity to have a team member assign for a project with the new agile team to take the experience back to their own teams.

> Have more than just my direct reports working on the project … there are engineers from other teams. (Engineering Manager)

4.2 Methodology adaptation

This section provides comprehensive findings of agile method adaptation at Akldevelopment based on the components identified in Fitzgerald's adaptation framework: original formalised methodology vs. methodology-in-action, profile of the development environment, overt factors and covert factors. Findings on each of these framework components on Akldevelopment are provided separately and in the order above.

4.2.1 Original formalised methodology vs. Methodology-in-action

This section presents findings in relation to original formalised methodology vs. Methodology-in-action at Akldevelopment. The findings in relation to this component are presented according to the themes that emerged from analysis of data collected from Akldevelopment.

4.2.1.1 Method-in-action

The Akldevelopment agile development team has assembled a hybrid agile method adapted from Scrum and Extreme Programming (XP) methods. This team's agile

approach is based upon the agile values and principles for software development. As a result of a new start-up team with no agile experiences including individuals who have never worked with each other, the engineering manager identified the appropriate development practices for adoption from Scrum and XP methods. Scrum provides practices on project management, whereas XP has more programming essence. The engineering manager had significant experiences of agile method implementation with other organisations and initially played the role of a coach helping the team to adopt this new approach.

> Chosen agile approach … delivering in a reliable way … following XP style … we pair program … we have test driven [development] … use Scrum for project management … doesn't prescribe any specific documents or things that [we] have to do … gives a time line … a structure for running the project … get to an incremental iterative approach … it runs in sprints … every single release is split up into Sprints … at the end of each sprint a review takes place … people sign up for what they want to do in the next sprint. Also in the sprint review meeting you reflect upon what went well and what went bad … improve how to work. (Engineering Manager)

> Our team is full Agile and using XP and Scrum … we will try to do as much agile or as much of the XP technique as possible … have iterations, retrospectives, story cards pair programming and test driven development. (Senior Engineer)

4.2.1.2 Rationale for adaptation

As the agile team undertook projects expanding the product capability they had to deal with challenges impacting the quality and delivery schedules of the new features. These challenges mostly arose as a result of Akldevelopment's need to continuously deliver features ahead of their competitors. From the onset, the team learnt to tailor their development approach to counter issues arising during the projects. They are aware that they can not achieve on-going success by applying the same approach end to end on development projects.

> If you don't adapt to the market, you are out of business … have to be faster than the relevant competitors … in terms of time to market … excellent in terms of quality … have to adapt as quickly as possible. (Engineering Manager)

> Have some very good competitors … to be the first in an area … need to market the need for [features] so businesses start thinking that they need it … [with agile] faster than how we were building before… run software all the time, which is fantastic … see what's happened. (Product Manager)

Their development projects are closely aligned with the company's business goals and objectives such as the company expanding into new markets or developing new features. The agile team ensures that the business goals and objectives are achieved. The goals and objectives play a strong role in formulating the product vision and the reason why the specific features are being implemented at Akldevelopment. These may change according to their market demands. Therefore, Akldevelopment's agile team cannot depend on strictly applying their hybrid method for all its projects and they have to adapt to achieve those goals and objectives.

> Have strategic objectives, different ones that we try to achieve … every single project has to line up against those [objectives] … if a suggested project does not contribute to any of the strategic objectives, it doesn't get the numbers [priority]. If there's a change, switch direction on a very short notice ... when we choose a method it has to address those requirements [adaptability]. (Engineering Manager)

> One of the best things about agile is that you can go part way and alter … with old fashioned cycle, there would have been a solution in design … extend project schedules and rework the whole thing. Very collaborative agile environment … if there is a challenge … we sit down really quickly, work it through, and then they (engineers) implement it. (Product Manager)

4.2.1.3 Adaptation process

Adaptation of their hybrid method is regarded as adopting suitable practices of published methods and the team modifying, improving or replacing them if problems are experienced. Method problems encountered are discussed during their weekly sprint review meetings. During this retrospective meeting the team (engineers, onsite customers and engineering manager) collectively makes decision on the issues which are immediately implemented. At Akldevelopment, sprint review meetings are compulsory for all team members to attend and are held on Friday afternoons before the start of the next sprint cycle. The adaptation practice at Akldevelopment is an informal process and the approach for adapting is based on consensus amongst the team individuals. There is no formal documentation either of their development process or any change that is brought about by the team.

> Adaptation … take something off the shelf and plan an agile approach … adapting it to your needs where you see fit … when feel the pain … look at the issue … identify the root cause … identify options and try one at a time, start with the most promising one. (Engineering Manager)

> We also have a sprint review as well …it normally takes half an hour … during the Sprint review we talk about things that went well and things that went wrong with the previous sprint … whether or not things could be improved. (Senior Engineer)

4.2.1.4 Adaptation responsibility

The agile team is responsible for their method; any decision to adapt it to suit projects is delegated to the entire team. It is a strong belief of the engineering manager that the team itself must deal with any issues arising with their method. The team has embraced the approach to learn and understand the underlying principles of the practices before making any changes to suit their development environment.

> They [agile team] make a reasonable decision … without any management intervention … makes decision making much faster … gives people a better sense of pride … also gives them a better sense of ownership … in charge of their own destiny. (Engineering Manager)

> Agile processes are very empowering … give you a lot of power … have worked for a process and develop a team that is in charge … [however] need to have people who are on to it …more in tune with what's going on, and more experienced and knowledgeable. (Principal Engineer)

The adoption decision for a new practice, technique or a tool is made by the team based on being tried out with their development environment. Even though the engineering manager has the final authority on what is to be adopted he implements what the team agrees upon. At Akldevelopment, the agile team is empowered to make decisions on a wide range of development issues.

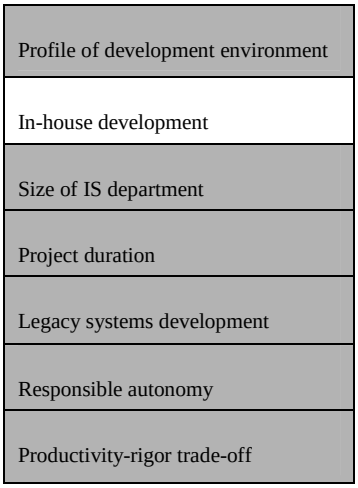
> They select technologies [and] tools … they can run Spikes … to apply, practice is an important ingredient … make most of the decision themselves … there is only a few things that I have the final call. (Engineering Manager)

> Participate in planning … contribute to the Scrum and retrospective meetings … input into the research and development of new technology that we are going to use in our product … I am pretty much involved in every part of the process. (Engineer)

4.3. Profile of the development environment

This section provides adaptation information on Akldevelopment's development environment (Profile of development environment) based on Fitzgerald's adaptation framework. Figure 8 provides its adaptation factors as identified in the framework. First, information on adaptation of Akldevelopment's in-house development is provided as shown in Figure 8. The findings in relation to this factor are presented according to the themes that emerged from the analysis of data collected from Akldevelopment.

Figure 8 In-house development – Akldevelopment



4.3.1 In-house development

Akldevelopment's in-house development is a planned option to keep growing this capability in maintaining product leadership and technical knowledge. They have never been in danger of exposing the product's technical knowledge outside the company. Akldevelopment's in-house projects are short term, one month to six months in duration. Agile developments allow lots of flexibility. As such, Akldevelopment's agile team adapts its approach to overcome issues impacting quality and timely product release. They now are capable of having 3 to 4 major releases annually- a strategic capability in their product domain including the ability to change priorities for feature implementation and set achievable release dates. Agile development provides them with the advantage to market new features without competitor threats.

> This industry, it's a slow one … spend years working before you get it out … have to be faster than the relevant competitors … need to be fast in terms of time to market … have excellent quality… to adapt quickly, if there's a change. (Product analyst)

Akldevelopment has outsourced some of its work through contract development, primarily to boost its development capacity. Akldevelopment outsourced client-commissioned work, which did not require specialist skills and experiences working with new technologies. They had co-located the contractors from India to the Auckland office for nine months to up-skill them and for them to learn their development process. They now work as a team from India.

> Outsource things that are repetitive, straight forward to explain … to get lower costs and also increasing [development] capacity. Contractors fully understand how we work … had them on site for an extended period of time. (Engineering Manager)

> When you're driving changes to a client … you often have to manage the price specification, a fixed price quote … go over budget and it costs you a lot of money, go over time and it costs you a lot of money. But with an agile approach it is particularly [important] to adapt because you know you've got fixed price development for a customer … this company sells such large scale software there's large scale changes to go along with it. (Senior Engineer)

The company does not use contractors to implement new innovations since it will limit the growth of their in-house technical capabilities and product knowledge. For this reason, Akldevelopment has its own engineers while supporting their fluctuating development needs through contractors.

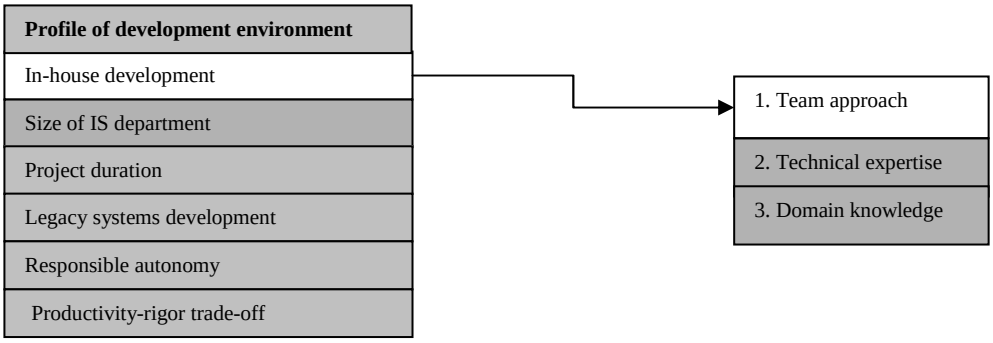
> There are parts that require substantial knowledge and skills and experience … we would like our people to work. (Engineering Manager)
>
> A lot of times we try to use sort of new technologies … that we are going to use in our product … have the opportunity to learn new things … do something experimental …want to use this new web interface technology … start putting it into our application to see if it works … if it starts working then we will try it on other parts of the application. (Developer)

Akldevelopment has focused on team approach, technical expertise, and domain knowledge to build highly skilled and innovative in-house development capabilities. These three sub-factors associated with in-house development function emerged from the analysis of data collected from Akldevelopment. First, information on the team approach is provided. Figure 9 illustrates the adaptation factors associated with in-house development.

4.3.1.1 Team approach

As an innovative software vendor, a team approach is absolutely critical for their engineering function. Their team approach includes input from the sales, marketing and product departments including from their clients. Akldevelopment's product development requires functional units to work as a team on projects to deliver quality products in the shortest time.

Figure 9 Team approach as part of in-house development – Akldevelopment



At Akldevelopment, their agile team requires plenty of insights to implement new features. Their product managers have first hand information on the current state of their

industry. They are the key source for domain knowledge and client related information. The marketing and sales departments also require a close liaison with product managers so that they have reliable information on feature availability. Information on implemented features helps marketing and sales departments with starting the marketing campaigns or sales processes with prospective customers. For these reasons, product managers at Akldevelopment are co-located with the agile team. They are the information sources for the business and engineering functions.

> Works best if people contributing towards a project are co-located and work as one team, the engineers, product manager, marketing, sales, and testers … success happens more profoundly if they have collaboration. (Engineering Manager)

> Once you have your story card …the first thing you do is you talk to the [on- site] customer ... have to talk to know exactly what he wants… once you get out your requirements, start thinking of how do you actually write a test for it (Developer)

To achieve an effective team approach, the company promotes team spirit, encourages and provides a fun work environment; they strongly focus on providing a work environment where individuals enjoy and feel part of a team. This makes them non-hesitant and freely able to contribute, help others, and seek assistance when the situation arises. For team building the company caters for several events such as setting up a gaming station, and providing for table tennis and pool tables. The company also caters for Friday afternoon drinks and frequent team lunches.

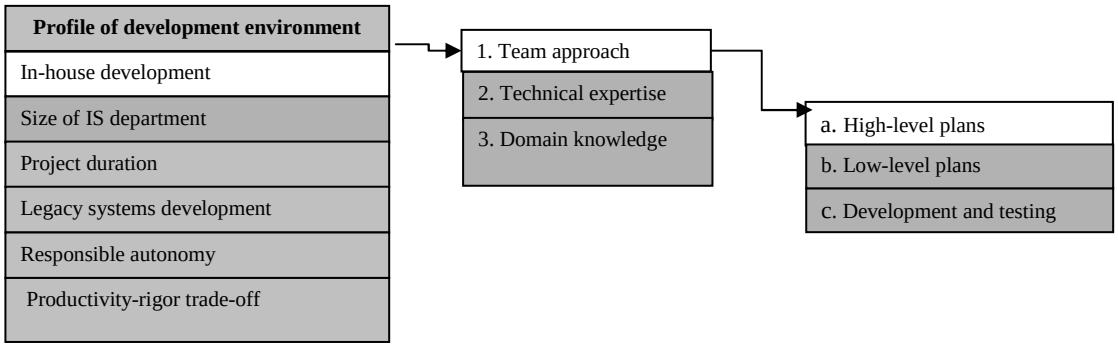
> People from across teams get together … kind of community building thing … it's not work related in the sense of the activity but it gives them a break … it means fun … they feel like a team. (Engineering Manager)

> Had a couple of sports events, I personally didn't go … usually have drinks on a Friday after work for 2 or 3 hours … will stay and have a few drinks and socialise. I wouldn't say scrum meetings are always socialising … things that aren't work related tend to come out now and again … it definitely adds a social element to work. (Engineer)

Akldevelopment's in-house agile product development has a three phase development cycle; high-level planning, low-level planning, and development and testing. Next, information is provided how each of these phases is adapted.

This section provides information on Akldevelopment's high-level planning task and adaptation to achieve an efficient team approach. Figure 10 illustrates the three key phases of Akldevelopment's agile cycle of its in-house development.

Figure 10 Team approach in-relation to high-level planning – Akldevelopment



4.3.1.1.1 High-level plans

Akldevelopment requires executive level approval for implementing any new product features. Their new developments require high-level (product) plans such as the vision and roadmap plans to state business cases for proposed features. Akldevelopment has adopted a team approach for this process requiring input from the key stakeholders in developing vision and roadmap plans. To develop these plans the product managers conduct research requiring them to read various industry publications, investigate the market, and talk with clients and industry consultants to identify high value features. They also collaborate internally with the sales, marketing and engineering departments. While their vision plan identifies new features, the roadmap plan provides the likely release dates over a period of time.

> Business side of it is coming out of product managers … technical side is provided by the software engineering … product vision planning … every single person has the opportunity to contribute. It could be improvements … to address the market … business cases are presented to executive … looks at each of them based on what is the cost and benefit. (Engineering Manager)
>
> Have idea collections … then have a screening … occasionally there's some conceptual analysis … [then] go into business casing and investment approval … following that there may be more detailed analysis. And then you get into the engineering. (Senior Product Analyst)
>
> Look at it from the iteration, they are limited … see only a week ahead of you … can't see very far. You can't see where you are going and you don't know the big picture … the first release planning that I attended … came out with a bigger picture of our roadmap. (Engineer)

4.3.1.1.1.1 Adapting product planning

In November 2006, the product planning process was adapted to enhance the team work in developing a more achievable product vision and roadmap plans. This adaptation involved creating a product strategy group and setting up a project approval steering committee. The product manager who had co-located with the agile team since 2005 to provide domain support felt high work pressure in simultaneously carrying out product

management tasks. In June 2006, this was raised in a sprint review meeting and the agile team collectively decided to ask the company executives for a full-time on-site customer. Hence, the idea to adapt the product planning approach was raised by the agile team manager and discussed with other development managers. The development team managers agreed that product planning also required engineering input on high-level requirements. This involvement was seen as a major benefit for the engineers since it would introduce the engineers to proposed innovations much earlier in the product development process. The idea to adapt the planning process was presented collectively by the development managers to executives of the company. The company executives accepted the proposal and created a product strategy group, headed by a product strategy manager and consisting of product managers and product analysts. This product planning adaptation in November 2006 required some of their product managers to adapt to a product analyst role. The product analysts are permanently co-located with the agile team. The product managers were made responsible for compiling vision plans through wider consultations. In December 2006, the company executives also made a decision to create a steering committee to make project approvals. Since November 2006, the agile team contributes to the vision plan by providing a ball park figure for developing new features. Based on the figures, the steering committee decides if they should pursue the new development. Hence, an environment for more effective collaboration between the business function and agile team is created.

> Created a product strategy group, within that there's the product analysts … our role more internally focused. We [product managers] struggled with the workload of doing work with the engineers … having to go out and do research and other things, and industry trends and market changes and that kind of thing … so taken that part of the role off us. (Senior Product Analyst)

The previous requirement elicitation process had a structured process involving different roles working separately from one another. The project specification was written by the product managers stating what new features must be implemented

> When we started … a thick document, sometimes up to 100 pages … that was the requirements … got away from that. (Engineering Manager)

> Given a project … product manager gives me some very high-level requirements … my job to break those down into workable stories … define what it is exactly that is required to be done [built]. (Product Analyst)

The high-level requirement elicitation requires a collaborative approach between the product strategy group and the engineering department, also involving the sales and

marketing individuals. At Akldevelopment it ensures that high value requirements are within the capabilities of the agile team to implement them. This model is based upon face-to-face interaction. The product managers are responsible for defining the high-level requirements through the vision and roadmap plans. The product analysts ensure that they are implemented. The agile team at Akldevelopment has business representative providing requirements and acting as an interface with their business function.

> Product analysts take care of what gets implemented [high-level requirements] … are co-located with the team. Product Manager … gives very high-level requirements. (Product Analyst)

To determine the business value of some of the proposed features requires collaboration with a client organisation. This involves installing a simulated version of the feature in production at the client site. Getting feedback and input from them helps to evaluate the high-level requirements. Adapting the product planning approach to include client contribution provides Akldevelopment with real credibility in marketing the new features. In January 2007, the idea to create prototypes for fuzzy innovations was proposed and agreed by senior engineers during their sprint review meeting to provide more accurate high-level estimates. This was proposed to the product strategy group, which also took it as an opportunity to use it as a simulated version to get feedback from clients on proposed features. However, such developments require taking approval from the steering committee because separate budget has to be allocated for it.

> Something new … work with forerunners at the client end, put into production and running … have an actual live test case … bringing new things to the market having some credibility. (Senior Product Analysts)

4.3.1.1.2 Low-level plan

This section provides information on Akldevelopment's low-level planning task and adaptation to achieve an efficient team approach.

Analyzing the high-level requirements to elicit low-level requirements or stories in creating a product backlog is done collaboratively between the business function and agile team. This planning session is driven by their product analysts. Previously, one worked as a developer and the other as the tester for the company before taking up product management roles. Their responsibilities are to work with engineers to create and manage product backlogs. As the product backlogs owners they take the on-site

customer roles, assigning priorities to the stories, providing domain information, writing acceptance tests and testing implemented stories.

> High-level requirements … break them down … with engineers get them all sized … the product analysts detail it out, write the stories and work with the engineering team, answer questions, test and do the operational reviews (Senior Product Analyst)

Adapting low-level planning

Described below are some of the key adaptations of Akldevelopment's low-level planning phase.

4.3.1.1.2.1 Adapting with usability professional

In January, 2005 Akldevelopment had incorporated a usability role with their agile team to adapt their team approach for planning and implementing the product backlog. The usability engineer's contribution in planning the product backlog helps to shape user experiences of stories. The development manager had decided that a specialist was needed to assist the engineers to address the usability related concerns and provide clarity of the stories since Akldevelopment no longer had a quality assurance team. The development managers at Akldevelopment collectively agreed to this support role for their teams, which was requested and approved by the company executives. The usability specialist works with the product analysts and engineers to plan the product backlog and to provide the engineers with his expertise. His contributions are extremely valuable for establishing optional interfaces for stories, since some interfaces are too complicated and expensive to implement.

> Usability is a specialist skill set … consider usability aspects and factors from day one … design an interface, if our engineers find very complicated and extremely expensive to implement… the usability person [makes] suggestions … get involved in the stories … has domain knowledge and personas … there is three way collaboration. (Engineering Manager)

> He plays a pretty big role … requirements, we would mock up a screen … would run it by him … has all the knowledge on how it would look … comes up with some ideas on functionality … if he sees it just very tedious, has ideas on how to make it smoother for the user. (Product Analyst)

4.3.1.1.2.2 Adapting backlog planning

Initially, the product backlog planning involved the entire agile team. However, it led to unacceptable overheads so the agile team agreed to adapt the rule to involve only the senior engineers. The engineers themselves realised that involving the entire team for backlog planning was putting time constraints on implementing the stories committed

for a sprint. In August 2005, they collectively agreed with the on-site customer (product manager) during a sprint review meeting to involve only a few senior engineers in planning the backlog. However, involving all engineers with their backlog planning was regarded as extremely valuable by the team since it introduced them earlier to the development projects. Now, senior engineers and on-site customers who take part in backlog planning are expected to communicate stories to the entire team. To ensure that all stories are being explained to the entire team, the engineering manager in September 2005 proposed a product backlog review meeting before the start of a project. The agile team agreed to this practice and adapted the backlog process by introducing a review meeting to go through the backlog so that engineers can visualise the project.

> The product specialists come with high-level stories, the senior engineers would be working on backlog … we had the entire team, but that involved too much overhead. Before the project starts have a review meeting to go through the backlog … know everything that is going to be done on a project. (Engineering Manager)

> The [on-site] customer or one or two engineers plan the release [backlog plan] … then they just invited all engineers to the meeting and show them. (Engineer)

4.3.1.1.2.3 Adapting tool support

The low-level requirements in the product backlog are captured as stories. Their requirement for a story is that it must be discrete for implementation, that it stands alone, is testable, can be used as a function and provides business value. The stories are further analysed and negotiated during sprint meetings involving the product analysts and engineers before the team make implementation commitment. A story may be regarded as an epic and broken into smaller stories. The estimate of a story may also be negotiated if engineers judge it as unreasonable. Earlier in their agile adoption, the agile team used index cards to write stories. In March 2005, the team decided to use a software tool, XPlanner to enhance their team collaboration on backlogs. Some of the software engineers felt that it was better for them to use a software tool to support collaboration on user stories. Hence, a collective decision with the product analyst was made during a sprint review meeting to use a software tool for this purpose. A month later, after a collective team investigation and agreement, the engineering manager got an approval for funding from the senior executives to purchase XPlanner as an appropriate planning tool for their agile team.

> We started with story cards … some people felt that we would need some professional tool … use XPlanner which is as close to an agile tool. (Engineering Manager)

> Very important that we have a good knowledge about the tools at our disposal and how to use them … if it gives a tangible benefit then we would probably use one … used to log defects on XPlanner. (Senior Engineer)

In June 2005, the product manager discovered that XPlanner was not a good fit for their environment since she experienced issues around story searching, tracking and changing priorities. During sprint planning meetings, the product analyst found difficulties in displaying the entire backlog of stories all at once to the engineers for discussion. Adding to this complexity, the engineers realised that the stories captured in XPlanner were extremely detailed and epic stories. The engineering manager pointed out that detailed story description was not facilitating the required on-going collaboration on a story with the product analysts. During a sprint review meeting the product analyst suggested using index cards again, which the team agreed to. The product analyst identified the benefit of able to easily lay all the cards down on a table for discussions during sprint planning meetings. The principle engineer also suggested an added benefit of using index cards for recording bugs and sticking them on the wall for engineers so that they do not lose the focus on them. In June 2005, the team adapted back to using story cards. As test driven development became more prominent with this agile team, product analyst wrote tests for each story on the back of each index card, requiring engineers to implement it as acceptance test. The acceptance test required each story to be in its smallest form since small stories with their tests are much easier for engineers to implement. In November 2006, the team collectively decided to have a rule for a story to be no more than 3 points, where 1 point is equivalent to 8 implementation hours of the total hours available for each sprint. As a result, this agile team adapted to having one week sprint cycles and also experienced the ability to implement more stories in a sprint.

> XPlanner was terrible to go back and look at previous stories … found tracking really difficult … also searching was really difficult in XPlanner. (Senior product analysts)

> Extremely detailed and large stories … we learned over time to specify requirements in a slightly different way … product analyst started to use story tests then we started to think about smaller stories. (Engineering Manager)

The team adapted back to using index cards for writing stories and also using a spreadsheet for creating and managing the backlog. Here, with the index cards, what is written is only what fits on the front of a card when writing with a marker. The product analyst also writes additional details such as algorithms, field names and other

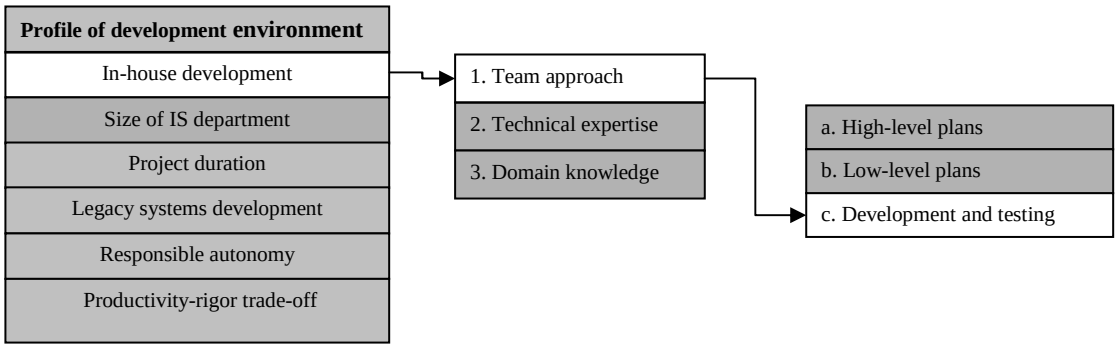
validation information including story tests on the back. Story tests provide worked examples for the engineers. The story cards are usually written by the product analysts but on occasions an engineer is required to write technical requirements. With this agile team, the product analysts have a software engineering background.

> The front isn't in pen … most likely in 10 words … back of the card … list things like a number, it has to be 0 or greater … the algorithm is written in pen … often they will look at the example in the story, on the back of the card.… stories about technical things, works better if an engineer has written them. (Senior Product Analyst)

4.3.1.1.3 Development and testing phase

This section provides information on Akldevelopment's development and testing phase and adaptation. Figure 11 illustrates this phase of Akldevelopment's agile development cycle in relation to its in-house development.

Figure 11 Team approach in-relation to development and testing – Akldevelopment



The cross-functional team effort of the agile team extended to the product development and testing phase. Their agile development efforts are well supported by the product analysts and usability engineer. During implementation, these two roles are permanently with the team available for discussion and to provide an extra bit of information on the stories. Having a team philosophy, they are motivated to provide immediate feedback to engineers on implemented stories.

> Sit on the same floor as them … regularly come to me with questions … might want me to review the work they've done before they commit it … to test something as soon as I can. I am 100% dedicated to this project. (Product Analyst)

The engineers work as a team, adopt a team attitude, responsibility and ownership of the work that they commit to deliver by the end of their sprint cycle. Their interaction and collaboration are face-to-face, working together through pair programming to

implement stories. The following comments provided by a few individuals of the team highlight the nature of their agile team effort.

> Easily ask someone questions … easier to communicate … more enjoyable … get feedback and help on a solution, help deal with problems … lot of people here have great experience … makes me feel sort of belonging to something … when the team's successful you're successful. (Software Engineer)

> If pairing with someone ... someone else [partner swap] can take it, and hack it apart … that takes a bit of getting use to. (Principal engineer)

> Try to avoid focusing on individuals … have people who are willing and able to help each other …work with each other regardless of ranking. (Engineering Manager)

Described next is the key adaptation of Akldevelopment's development and testing phase.

4.3.1.1.3.1 Pair programming

Pair programming is one of the practices adopted in January 2005 by the agile team to foster collaborative rather than competitive behaviour amongst the engineers. Using the practice, they expect to develop a team culture where individuals are willing to work together and help each other. This agile team implements stories related to new features and it is expected that pair programming will also enable them a quick transfer of specialist knowledge to others within their heterogeneous skilled team. Another key factor for its adoption by this agile team is to implement better quality code.

> Pair programming for knowledge exchange, training on the job, quality improvement, exchange of ideas … when 2 people propose an idea … discussing the 2 ideas the 3rd one emerges … better than any of the 2 … an improvement of code quality … it is essentially a continuous code review … avoid a lot of defects in the first place. (Engineering Manager)

> Learn so much more with pair programming … quality is higher both in the sense that there is less bugs … bugs are found earlier … [and] the general design of the code is better … there is no real down side to pairing … other than maybe your pair might be a little bit slower. (Senior engineer)

The individuals in the agile team had no pair programming experience. A practical approach involving demonstrations, practice and coaching sessions was used in helping engineers to adapt to a collaborative method for writing code. With a hands-on approach for introducing this practice, the agile team required little convincing to adopt it. However, it was expected that some individuals will have difficulty with the practice, resisting its adoption.

> Pair programming … not a lot of convincing required … a minority, that either [were] not capable or willing to adapt. It was more about demonstrating how to do it … the person who did not seem to adapt, had to look for a different role. (Engineering Manager)

> I do like it … because of the communication and the learning … might be other better ways. But compared to what I was used to this is pretty good … learn so much more just from working with other people than I was. (Engineer)

Adapting to this approach for story implementation was helped by the fact that the team had individuals with different skill sets. They had at least one skilled individual for each of the different areas for which the team was required to have skills and knowledge to undertake development projects. Pair programming was viewed by engineers as an opportunity to learn from one another. At the beginning, pairs were formed on the basis that they were capable of implementing a story. However, difficulties were experienced in allocating some stories since the skills required were only held by a single individual.

> There was a good blend … different areas that we needed to work on we had at least one person … there was at least one pair capable of working on a task … was hard to find a suitable pair … specific knowledge was only available through one individual … improved over time. (Engineering Manager)

> Pair with someone every day, so it doesn't take long before you get to know someone's strengths and weaknesses … people graduate towards different skills …you leverage off each other . (Senior Engineer)

Over time the engineers adapted their technical skills to cover a more general skills base. One initial engineering skills adaptation that was required was communication skills. The engineers became aware that without the ability to communicate effectively, the pair effort will break down and impact the team commitment. They realized that communication skills are really important since a pair has an on-going discussion about their story with the on-site customers and the usability engineer.

> Have a reasonable level of technical ability … to have that base there so we can communicate on same level … much more focused on communication … the personal interaction … got to have a lot more social skills … good at communicating things to the whole team … talking to the on-site customers and usability engineer is new thing. (Engineer)

To deliver team commitments, they had to build a good understanding regarding individual contributions and responsibilities when pair programming. While having no set rules regarding role swapping, it requires engineers to adapt from a driver role to a navigator role and vice-versa. The team has built an understanding that the frequency of role swapping in a pair will be dependent upon the individual understanding of a story,

skill set and experience. Regardless of these differences, it is expected that both individuals will actively take part and communicate with each other the entire time rather than the navigator just sitting and passively watching the driver.

> The role [driver and navigator] to change at least twice, both have turns … both at the same skill level, it changes a lot … easier to communicate by just grabbing the keyboard and saying this is what I mean … they [driver] need to know what they're doing … good at communicating what they're doing … design story, just talking through it … navigator to ask questions, look out for little mistakes … make suggestions … writing down notes that you need to come back to. (Engineer)

The agile team's pair programming approach also requires good cooperation with the other pairs. While the pairs are expected to help each other, they have some common practices which all pairs have to abide by to have an effective team approach.

> Have team practice … expect them to uphold … fixing the build if it is breaking … not committing [submitting codes] if the build is breaking … expect them to ask for help. (Software Engineer)

4.3.1.1.3.2 Adapting pair programming

One major pair programming adaptation that the agile team implemented in August 2006 was regarding a daily change of the individuals in the pairs. The idea to have daily swapping of partners was suggested by the principal engineer during a sprint review meeting since they were regularly adapting into small sub-teams of 4 to 6 engineers on projects. The team agreed to the suggestion and brought in this rule to avoid having the same pair working together for several days on a story to enhance the team's skill set. Adapting to a daily change of the engineering pairs enabled the development of a more homogenous team, with individuals having a more general skills base.

> People pair for several days in a row … not really provide you with the value … the pair working on a story, one will move off and signs up to a different story … have a swapping of partners every day. (Engineering Manager)

> We don't want one person to own all the UI, database or performance work … have everybody working on all the code … rather than have one pair work together on the story for 3 days …rotate 2 or 3 people through that story in a week. (Senior Engineer)

> Change pairs every day, especially for the last few months … familiar with most of them … their soft skills and communication skills … [technical skills] as soon as you work with them you are going to find out … there would be some people who I would rather not pair with. But I still would do it. (Engineer)

The daily partner change enforced team rather than individual ownership of a story. The stories committed for delivery are the responsibility of the entire team. However, on occasions a single pair is assigned to a set of stories relating to usability enhancement. This agile practice was suggested in October 2006 by the principal engineer and was accepted by the agile team. The decision on which two individuals will pair to work with usability consultants are made by having a meeting between the team leaders, technical leaders and engineering manager. This pair is responsible for the usability stories and requires working with the usability engineer to ensure a complete implementation. Adapting to this approach is seen as a better way to build usability knowledge of the entire team since the pair imparts the skills and knowledge gained when they work with others. This is also regarded as a more efficient use of the usability expertise when assigned to a pair.

> The usability [engineer] is working with all teams … will have a pair of engineers working on just usability … he can get an additional significant input. (Engineering Manager)

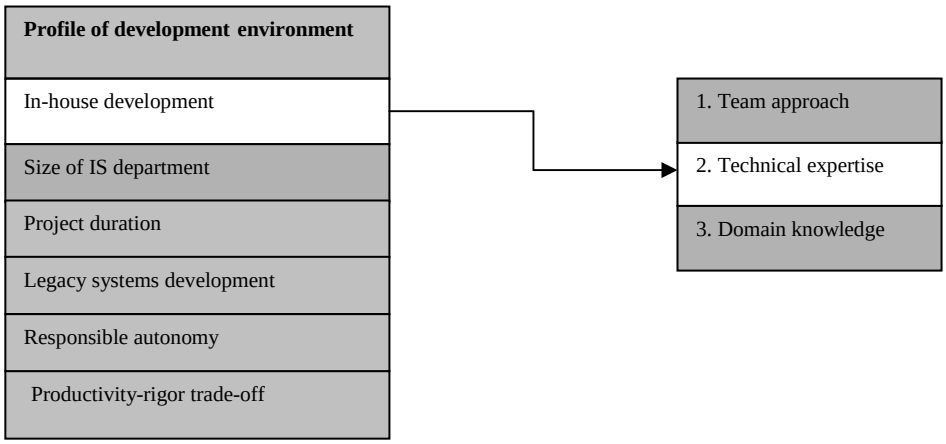
> My skills have increased a little bit … we have still got maybe one or two people that really know the UI [user interface] side of things … slowly trying to pair people with people that have got the skills to try and get their knowledge. (Engineer)

Next, information on adaptation of Akldevelopment's in-house development with regard to its technical expertise is provided, as shown in Figure 12. The findings in relation to this adaptation factor are presented according to the themes that emerged from the analysis of data collected from Akldevelopment.

4.3.1.2 Technical expertise

Akldevelopment adopted software engineering roles with the adoption of an agile approach. This section provides information on their engineering roles and adaptation to keep enhancing their in-house technical expertise.

Figure 12 In-house development in relation to technical expertise – Akldevelopment



4.3.1.2.1 Engineering roles

Akldevelopment's agile team has different levels that are associated with their engineering roles. These levels are used to define technical competences, which are graduate engineer, engineer, senior engineer, and principal engineer. Here, the levels also reflected a person's past development experience and expertise. However, they are not used as a basis for assigning stories to the engineers. These levels help them to maintain an appropriate number of engineers at various levels for successfully undertaking projects; they help to get the right mixture of experience when creating sub-teams.

> Have graduates, engineers, senior and principal software engineers… mixture of these different levels within our team … new release in a specific area … certainly look at who would be working in there. (Engineering Manager)
>
> All of us are engineers … some of us are more senior … but we are engineers in the end. (Senior Engineer)

4.3.1.2.2 Engineer responsibilities

Regardless of their levels, engineers share responsibilities in helping to plan and implement new features. Here, the engineers have broader responsibilities, such as for on-going architecture design, and for story planning, implementing, testing, and integration. Amongst the responsibilities of engineers is to help others when one's expertise is required in solving implementation problems. With this agile team, the senior engineers shoulder most of this responsibility.

> An engineer gathers the requirements, talks to the customers, analyses requirements, designs, architecture, builds, tests everything … responsible for tools and deployment … an intermediate person knows how to do most things [and] cover most situations … but there are some situations where you really need someone more senior. (Senior Engineer)

4.3.1.2.3 Engineering career path

Akldevelopment provides a career path for its software engineers. The three levels for technical expertise provide a path for progression. Here, it is provided with a view to retaining local experience and knowledge. Progression is based on performance merit rather than on years spent in an engineering role. Beyond these three levels, an engineer can become a product manager or a product analyst. An engineer can also become a principal engineer, a level above the senior engineer. The principal engineer role at Akldevelopment requires substantial development and technology expertise and a very good understanding of the product and its growth path. The senior roles are expected to provide leadership regarding technology, architecture, design and processes, besides pairing with engineers.

> Hire people … intention is to keep them long term … have people who have been with the company for 14 years … provide with a career opportunity. (Engineering Manager)

> A lot of experienced developers …people on this team have been working for this company for quite a while … got a track record …delivered multi million dollar large software solutions. (Principal Engineer)

> You get put into a band, a senior, intermediate, or junior … experience, it's most important that they work in agile set up. (Senior Engineer)

4.3.1.2.4 Engineering talent

From the outset, the agile team has focused on recruiting extremely talented engineers. Individuals with outstanding Java, database, user interface and testing backgrounds, including experience working with an Integrated Development Environment (IDE) are highly sought after as IDEs have a long learning curve, but make development efficient and productive. Employing individuals with diverse backgrounds helped this agile team to achieve workforce agility and flexible capacity through "cross-training". In addition, they also require individuals to effectively work with others requiring engineers to be intelligent and to learn things quickly with a good work ethic.

> Degree in computer science … programming skills is important … experience with Java … solve problems or with other people … has a database, testing and user interface background … worked on any of these different layers of technologies … worked in IDE. (Engineering Manager)

> If people are not able to cope then they're in trouble … someone who is perhaps more in tune with what's going on … my responsibility to bring the guy along. (Principal Engineer)

4.3.1.2.5 Hiring criteria

With this agile team, excellent interaction skills and the ability to contribute effectively are extremely important criteria for hiring new engineers requiring them to have an open mind, deep problem solving and communication skills, and the ability to adapt to various domains. Their team effort requires engineers to work with any other engineer on a daily basis. Here, the engineer's interpersonal skills such as interaction, collaboration, and co-operative skill sets are as important as their technical skills for spontaneous collaboration making the individuals more proactive and resilient. All of their recruits are expected to be delivery focussed having the right attitude, commitment and the ability to think outside the square to help the team deliver features on regular basis.

> Communication and collaboration is the most important bit … ability to learn new things … adapt to work with team … an introvert, very unlikely find on this team … is about communication … does not lean towards collaboration, they will struggle to be successful. (Engineering Manager)

> For some people agile process just doesn't suit … they might be great people … strong technically … but maybe not great communicators or they just don't feel comfortable pairing and working together. (Principal Engineer).

4.3.1.2.6 Adapting hiring process

To ensure that new recruits have the type of skill sets required for their agile development environment, their hiring process was adapted in January 2006. The prior hiring process involved giving candidates a programming task to gauge their technical capabilities. This was followed by interviews before the offer was made to the best candidate. However, the senior engineers observed that some of the new recruits were not able to interact effectively with the other engineer during pair programming. Hence, the team with the engineering manager decided to adapt their hiring practice with a design interview which has a pair programming and design session to test the candidate's interactive and collaborative talents as measures of their ability to work effectively with the team. The adapted approach enables the interviewers to test the ability of candidates to communicate and to handle code and design critiques from others. In agile development environments, it is important that individuals give up their own viewpoints to agree with the team decision.

> Send a programming task … usually complete within 4 to 6 hours … make a decision on first interview … if successful, invite the candidate for a second interview … an assessment for 4

> hours … consists of two parts, one is pair programming , second part is a design session … identify the design and interaction skills … it is important that people are open-minded, willing to contribute, but at the same time are also willing to look into ideas that other people contribute … ability to work in a pair. (Engineering Manager)
>
> You should be able to work on any job with anyone … we are pretty particular about who we hire … we have [design interview] for 3 hours … not be able to pair with them we won't be able to work with them. (Senior Engineer)

Another major adaptation was empowering the agile team in April, 2006 with the responsibility of hiring new engineers for their team. This decision was made by the engineering manager to make the entire agile team to be absolutely responsible for planning and delivery commitments. Since 2005, the engineering manager had noticed that team was becoming more effective with each release cycle. The engineers and product analysts agreed that they had sufficient experienced individuals to collectively be able to determine and select new team members. Their recruitment process involves a group of senior engineers who are selected by their agile team. These senior engineers themselves conduct the interviews and decide the best candidate.

> Engineers do the recruitment, interviewing and selection process … they're senior people … automatically creates the buy in … hire the wrong person they have to live with it. (Engineering Manager)
>
> Have to be high performing … people are paying us money … could pay that money to someone else … within [Akldevelopment] there is a competition … trying to improve the velocity… in a team everyone [must be] competent. (Senior Engineer)

4.3.1.2.7 Adapting senior engineer roles

While they employ the philosophy of a self-led team, a senior or principal engineer adapts to become a lead engineer for a release or for the entire project. This decision for a lead engineer role with sub-teams was made by the engineering manager soon after the agile team was created in 2005. As the agile team started to function, the engineering manager realised that lead engineer roles were required to provide the leadership for the resolution of technical and architectural issues so that quick team decisions can be made. At times, he observed that sub-teams were spending more time in meeting rooms to come to an agreement. The decision to appoint an individual in this role is collectively done by the senior engineers, product analysts and engineering manager. This specialist role is an informal role and is rotated between the senior and the principal engineers for a temporary period of time.

> Have lead engineers … people with a very long tenure within the industry or company … essentially promoted to become lead engineers. (Engineering Manager)

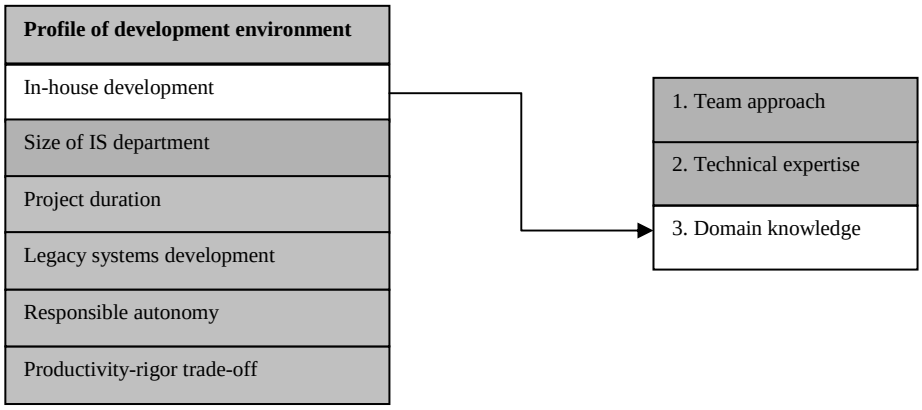
> Senior people … more reliant …more proficient than others … really good at the domain … really know the process. (Senior Engineer)

Next, information on adaptation of Akldevelopment's in-house development in regard to its domain knowledge is provided, as shown in Figure 13.

4.3.1.3 Domain knowledge

The agile team at Akldevelopment makes a significant effort to understand their intended product application environment. This section provides information on their learning approach and adaptation to keep enhancing their domain knowledge.

Figure 13 In-house development in relation to domain knowledge - Akldevelopment



Domain knowledge helps the company to understand the overall market needs, competition challenges, and individual customer requirements. Most importantly, it helps them to identify innovative features. This agile team has developed its own in-depth domain expertise through their product mangers.

> The product managers look at the market and the prospects [clients] … talk to market analysts and existing customers … how the business works, how the industry works. That is the information and the knowledge that they provide. (Engineering Manager)

The acquisition of domain knowledge is critical for their entire agile development. This knowledge enables them not only to understand stories but also to provide estimates for defining the high-level stories. Domain knowledge helps the team to create their product backlogs to flesh-out, estimate and to prioritize the Low-level stories; meet commitments for story implementation; ensure quality, implement user experience; and to write appropriate unit tests.

> Domain knowledge, don't get a common knowledge, can not communicate … domain model that we try to use is exactly the same model that the client, the backlog owner is coming up with … we developed our domain knowledge … expect the engineers to translate everything into business language. (Engineering Manager)
>
> Another skill that I learned was how to actually negotiate or talk to the customers about the stories that I am doing … you have more feedback from the customer. (Engineer)

The agile team has built up domain knowledge through their highly experienced product manager (who had 15 years experience), who is co-located with them. The product managers were also required for the client visits offshore, usually away for a week to a month. However, the agile team also has some vastly experienced engineers in the domain. One senior engineer has a background as a client consultant and steps into the product manager role. At Akldevelopment, adapting the product manager role to product analyst role in November 2006 has permanently co-located a business skilled individual with the agile team.

> Co-located [are] the product analysts …improve communication … have a question … you go to her … asked to provide expertise around the domain … to get a much clearer understanding There are other people here [engineers] who have excellent domain knowledge. (Engineering Manager)
>
> Slightest misunderstanding, you just need to talk to the [on-site] customer immediately … developers must be constantly talking to the [on-site] customer. (Principal Engineer)

4.3.1.3.1 Adapting with multiple product analysts

To have an excellent grasp of the stories, the agile team engineers require frequent communication with their product analysts. This support is vital for the agile team to deliver their commitment for each sprint. As their development capacity grew, the agile team had to adapt to include additional product analysts to maintain a similar level of domain support for the engineers. The engineering manager and principal engineer had recognised the need to have more product analysts to create sub-teams as they were being allocated high value projects by the steering committee, requiring additional engineers. In September 2007, the agile team had hired two more product analysts to have four in total with two at a senior level. Here, the product analysts have company experience and bring in a business perspective to the product development. Most of them started their career with the company in the engineering department, having an information technology background and qualifications.

> Stories tend to be short, easy to understand … but the business process and logic behind the story might be quite complicated … the story in itself is just a place holder for a conversation … maybe 10 to 20 times a day going back to the customer proxy [product analyst] … if the customer proxy isn't rushed off their feet answering questions, there is probably a smell in an agile project. (Principal Engineer)

> What we found works best is having a customer proxy between 4 and 8 engineers … because of the amount of ongoing interaction that's required. (Senior Product Analyst)

The approach that the product strategy group took was that the project manager works very closely with the product analysts. The product manager provides the product analysts with the high-level requirements for implementation. The product analyst is responsible for all the work between the high-level requirements and the individual tasks for every sprint.

> Product manager set scope … product analysts write the stories, work with the engineering team and have an ongoing discussions with the product manager … gets status updates at least once a week (Senior Product Analysts)

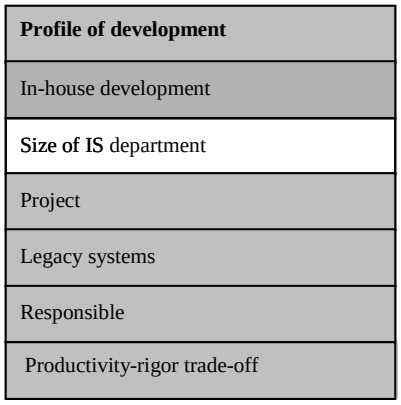
4.3.1.3.2 Adapting domain knowledge source

For the agile team to consistently build high value innovative features they require a good grasp of the domain. They have adapted their approach so that their product analysts and engineers keep enhancing their domain knowledge. Since they no longer work in the field, the need to keep learning the business domain of their clients from individuals who deal with them was identified by the senior product analyst as critical to able to support engineers. This learning approach was supported by the product strategy group and the engineering manager. Since January 2007, the product analysts continuously liaise with the company's marketing strategist and the sales team to learn about new happenings in the industry. In April 2007, the agile team was awarded a highly innovative project. Due to the nature of the project, the senior product analyst identified the need to be supported by the company's domain experts. Hence, two domain experts were co-located with the agile team on request by the engineering manager to the company executives. These domain experts provide specialist knowledge on novel features.

> Recently pulled in two domain experts into our project … help people understand the domain more … market strategist … brings important input … have people with different experience in the organisation … get input from them … the sales team. (Senior Product Analyst)

Next, Information on adaptation of Akldevelopment's IS department (engineering function) is provided. This is identified as an adaptation factor under the profile of the development environment in Fitzgerald's adaptation framework, as illustrated in Figure 14.

Figure 14 Size of IS department – Akldevelopment



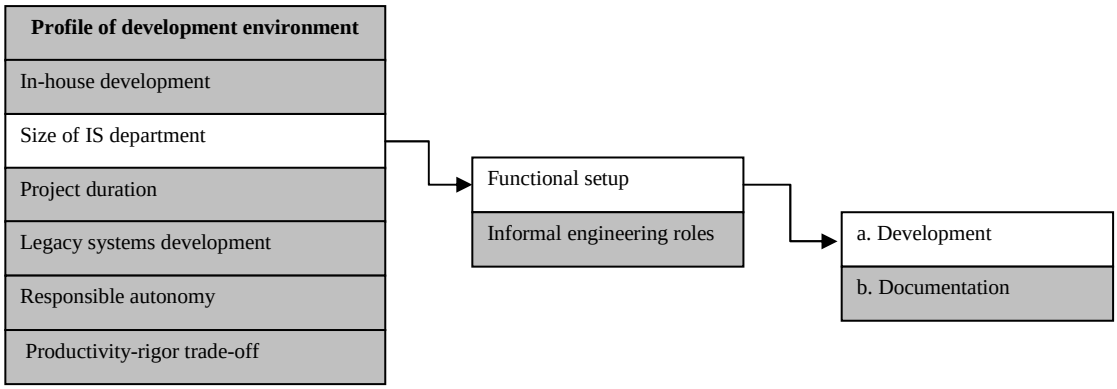
4.3.2 Size of IS department

The findings in relation to this adaptation factor are presented according to the themes that emerged from the analysis of data collected from Akldevelopment; these are functional setup and informal engineering role adaptations.

4.3.2.1 Functional setup

The engineering department of Akldevelopment has adapted into two functional units; development and documentation units. They had six other autonomous development teams when they created their agile team in early 2005. Their documentation unit is also an autonomous team. In January 2008, Akldevelopment's agile development team had 36 software engineers, 1 usability consultant, 1 agile tester, and 4 product analysts being managed by an engineering manager. First, adaptation information is provided on Akldevelopment's development function (the agile development team), as illustrated in Figure 15.

Figure 15 Development function in relation to size of IS department - Akldevelopment



4.3.2.1.1 Development

For this agile team, work is allocated into small teams. Here, the teams are organised based on the development projects. In January 2005, they adapted to having 4 self-organising teams with 6 individuals each implementing different parts of a project. The agile team itself decided on this initial setup so that they as a group immediately became productive. However, the idea to adapt into sub-teams was driven by the engineering manager. This agile team adapted to work in small teams to have effective and efficient communication and also better team work and coordination. The agile team did experience better time utilization with small teams, especially when having the stand-up meetings in the mornings.

> Communication and co-ordination is easier … had a daily scrum where we had 25 people … got 15 minutes … a similar scrum done by 4 people, spend 5 minutes … 5 minutes versus 15 minutes, save 2 thirds of the time. (Engineering Manager)

> One team [had] a dozen people on it … split that to 2 teams of 6 … is easier for 6 people to communicate than it is for 12 … the backlog split it into two. (Principal Engineer)

4.3.2.1.1.1 Adapting number of sub-teams

Adapting to a number of sub-teams within their agile setup is dependent upon the number of projects or the different implementation areas the agile team is allocated. The senior engineers, product analyst and engineering manager are responsible for identifying the sub-parts of a project. These distinct parts are used to adapt into a certain number of sub-teams. In August 2005, they adapted from 4 sub-teams to 2 sub-teams based on two new but distinct areas of a new project for implementation. Here, the number of sub-teams is adapted from one project to another.

> Driven by client [product strategy group] demand … given 2 major areas, we split into 2 teams … asked to work in 4 areas then we set up so that we have those areas. (Manager Engineering)

4.3.2.1.1.2 Adapting teams during the project

The size of a sub-team for a project is normally fixed. However, during a project the development capacity may be boosted with specific skills that the team is lacking. The lack of a specific skill is identified by the engineers themselves based on the stories they will be implementing. The team collectively decides to request an individual with the required skill from another team. For this reason in September 2005, this agile team adapted with a practice which included calling a team and technical leaders meeting to decide the approach to use to re-assign engineers. Every so often, the entire team, with the product strategist and the usability engineer, are involved in identifying options for adapting the team. This approach to involve the entire team is regarded by engineering manager as critical in motivating engineers to volunteer to move from one sub-team to another.

> Team decides … a certain number of stories that are coming up … require specific knowledge … is not represented in that team … a person in a different team may have that specific knowledge … decide to re-assign people. (Engineering Manager)

> Have slightly formal feedback … a weekly or a daily team lead meeting … only take 10 minutes to get together … certainly have inter-team communication… can also rotate people through teams. (Senior Engineer)

4.3.2.1.1.3 Rules for sub-teams on same projects

Due to inter-dependencies in different parts of a project, the agile team has the architecture and design implemented in such a way that multiple teams can work on it at the same time. This is achieved through a modular and layered architecture approach that enables one sub-team to work on one area and a different sub-team to work on another area of a project. There is always some shared infrastructure or code base. The sub-teams require the discipline to understand that working with their product infrastructure or shared code requires lots of collaboration and co-ordination with other sub-teams.

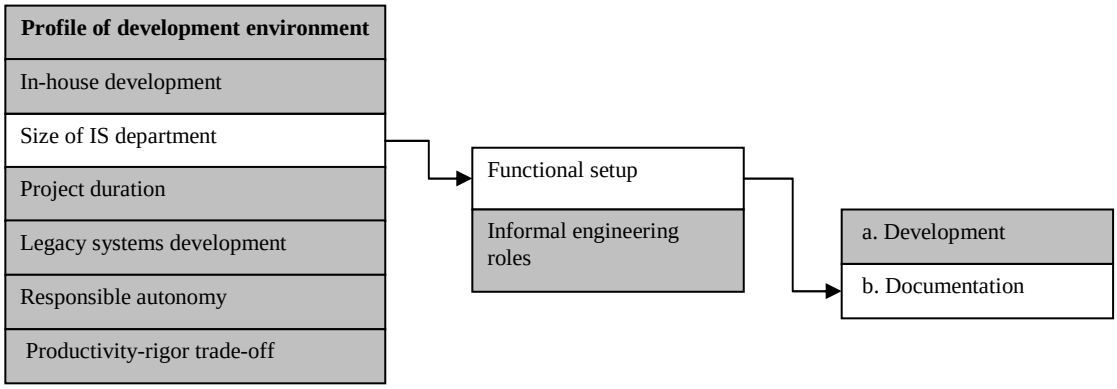
> Need to have the trust between teams … will let [others] know … people can say this is the worst sprint for this particular change … teams can negotiate the best point in time to do these things. (Engineering Manager)

> Team leads to get together … what are the issues that we need to think about in the future … working on these stories this week, this is the function area that we're working on … can we do so. (Senior Engineer)

4.3.2.1.2 Documentation function

Akldevelopment has a small documentation team employing three contracted technical writers with a permanent team leader. This section provides adaptation information on Akldevelopment's documentation unit in relation to Size of IS department, as illustrated in Figure 16.

Figure 16 Documentation function in relation to size of IS department - Akldevelopment



Here, the documentation functional unit still operates as a separate team post agile adoption. Their work involves writing the user documentation and on-line help, describing each feature in detailed steps for accomplishing an associated task. Since June 2005, the on-site customers (product analysts) work closely with the documentation team, while they also communicate directly with the agile team for clarifications.

> We kept it the same … meetings are scheduled on an as-needed basis… provided [them] an environment that contains the latest version [implemented stories]. Technical writers talk to product analysts or engineers … issues classified as bugs, they would create [story] cards and give it to the development team. (Engineering Manager)

> Keep on liaising with documentation … have to take on a bigger responsibility that it will be what they want. (Senior Product Analyst)

At Akldevelopment, the documentation role is formalised by the technical writer position. Their writing skills, their understanding of new technologies and their product domain knowledge are considered necessary to work in their agile environment. The nature of their product does not require technical writers to have a technical background.

> Understand needs of the target audience, e.g. an administrator versus an average user of the system … have an understanding of the domain and an ability to work in a team environment. (Engineering Manager)

> You need some good communication skills … how would the user see this … what would appear on the screen … technical skills I don't think they need as much of that. (Senior Product Analyst)

In June 2005, the agile team adapted by creating stories for writing release notes for each build to describe new functionalities that were implemented and tested for a sprint. The release notes provide the needed information to compile the user documentation. This practice to write release notes by engineers was decided by the engineering manager based on meeting with the documentation team since the documentation team had no prior involvement in planning and implementation of features and also they were not co-located with agile development team.

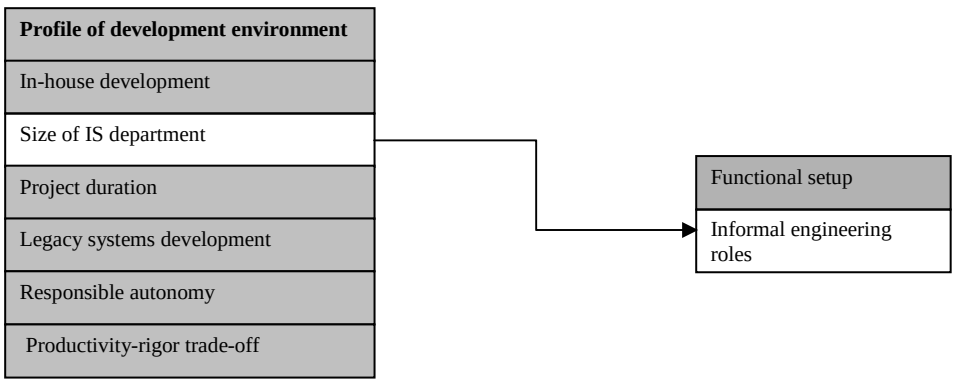
> Release notes are written by engineers. Separate stories are created for this task. (Engineering Manager)

> Release notes …basically it's broken down into tasks … story cards are written. (Senior Product Analyst)

4.3.2.2 Informal engineering roles

This section provides information on adaptation of engineering roles in relation to the size of IS department (engineering function) at Akldevelopment, as illustrated in Figure 17.

Figure 17 Informal engineering role in relation to the size of IS department - Akldevelopment



4.3.2.2.1 Leadership roles

Initially, Akldevelopment's agile team had self-organising sub-teams with technical leaders helping their teams with decision making. This approach was to make the whole

team responsible and to work together to deliver for every sprint. In October 2006, they adapted to have team leader roles for their sub-teams as the team grew in size. The need to have a team leader was proposed by the principal engineer so that an engineer can work with the product analyst to organise and get the sprint backlog ready for a sub-team. This idea to have a team leader was discussed in their sprint review meeting and agreed by the agile team. A senior engineer is selected to be a team leader for a sub-team through a joint meeting between the existing team and technical leaders, product analysts and the engineering manager for a release or a project. This is an informal role similar to their lead engineer role.

> The team lead, it has to do with the team size …initially we had 18 people, now it's 30 plus … for continuity the team lead [does not change] every week … person doesn't get a good view of the backlog … [unless] if one person constantly owns that. (Principal Engineer)

The two leadership roles are not combined into one role enabling the incumbents to have sufficient time to work as an engineer in their sub-teams. These agile roles at Akldevelopment have a more fluid set of project responsibilities when adapting to a team-based work structure. Here, both leadership roles are expected to pair with engineers to implement stories.

> Team leader but also a full time engineer … pretty much runs most of the processes. (Principal Engineer)

However, the leadership roles for their agile teams are determined on a case by case basis. In a sub-team with no more than 4 engineers, the two leadership roles are adapted into one. If teams of the same size are implementing parts of the project standing on their own, they adapt to have separate leadership roles. Through the experience of working in sub-teams, the team themselves recognised that for a small sub-team the two separate leadership roles were not required. In such situations, considerable effort goes into identify a senior engineer who can adapt to perform in multiple roles in a sub-team. Since January 2007, these two criteria have been used by this agile team to decide informal sub-team leadership roles.

> A team consisting of a tech and team lead … not necessarily mean that we follow by the letter in every single case … it has to serve a purpose … have a team that can be successful. (Engineering Manager)

> A good leader will help engineers to understand the [on-site] customer better… able to negotiate … motivate us to actually achieve our goals. (Engineer)

4.3.2.2.2 Role expectation: team leader

With this agile team, a team leader is an experienced engineer with good communication, people and technical skills. Here, the leader is not only required to be delivery focussed, but also able to facilitate meetings, moderate between people, figure out and solve issues, organise sprint backlogs and guide the team to make the right decisions. Adapting with team leader roles enables them to maximise development effort rather than engineers constantly being taken away for other team tasks or for meetings.

> Team lead role … make sure that the onsite customer has got stories organised for next week, people are following the process, running stand ups, and retrospectives. (Principal Engineer)

4.3.2.2.3 Role expectation: technical leader

The technical lead role facilitates and makes design and technology decisions, including co-ordination and collaboration with other technical leads to ensure that the resulting code base is consistent and of high quality. This role also provides guidance for consistency with regards to the design implementation and styles. At Akldevelopment, the individuals who take this role are regarded as highly competent persons with tools and technology. This role requires the person advocating test driven development and ensures that it is consistently applied by the engineers in the sub-teams. The technical lead is also expected to raise additional stories such as stories for refactoring or code cleaning.

> Makes design decisions … arbitrator if there is a disagreement or complicated architectural question … responsible for communicating outwards … raise additional stories around refactoring … provide expertise and knowledge in the use of tools and technology … identify issues that need to be discussed with other teams. (Principal Engineer)

4.3.2.2.4 Adapting ScrumMaster role

In March 2005, this agile team adapted with a ScrumMaster role. The decision for this role was made by the engineering manager to make the agile team self-organising, without his contributions. The team collectively decided that any individual could volunteer for this informal role for a release while also being an engineer in their sub-team. Here, the role facilitated the daily stand-up meetings and was responsible for taking actions on issues raised by the team. The ScrumMaster also made sure that rules were followed, such as a Scrum starting on time and not exceeding a 15 minute time limit. The other key responsibility was to be the tracker of stories that the team had committed to delivering for each sprint. In October 2006, the team collectively agreed

to incorporate the ScrumMaster role responsibilities with the team leader role based on a suggestion by the principal engineer. Only senior engineers are selected for this role. Hence, a major benefit for the sub-teams is that they are confident that the ScrumMaster tasks are now being carried out more efficiently.

> Beginning, we [had] a Scrum Master … any person that would come up and facilitate the meeting … but now we have a team leader. (Engineering manager)

> ScrumMaster role is to help the team to be self organised … very much a coaching role … … is not a permanent thing … at the moment I am working mainly as a team leader, like a sprint tracker … organising most of the meetings, the backlog … also a full time developer (Principal Engineer).

4.3.2.2.5 Adapting with paired BuildMaster role

The engineers were given immediate messages on their screen when the code failed the unit tests. In December 2006, the agile team adapted to include a BuildMaster role that monitors all the failed unit tests and ensures that they are immediately fixed. This role was proposed by a senior engineer since the agile team was experiencing frequent broken builds in a sprint as a large number of automated tests were being accumulated. The broken builds required immediate fixing for the effective functioning of their continuous integration system. Hence, the team collectively agreed to have a pair of engineers to volunteer during sprint planning meetings to work as BuildMasters for a sprint. Previously, no one tracked unit tests during sprints. Now, with their extremely large number of test cases that run constantly with their build process, this role ensures that bugs are not allowed to accumulate. The BuildMaster role is another of their informal roles which is assigned to a pair for a sprint. They ensure that their team maintains continuous and successful builds. As a BuildMaster, the pair is responsible for their entire build process and for managing their version control system.

> We observed that something [did not] perfectly fit … thought how we can handle it … decided for a BuildMaster, consist of one pair of engineers. (Engineering Manager)

> Different build systems … they're very important … spend a lot of time maintaining those build environments and making sure they're working. (Senior Engineer)

4.3.2.2.6 Adapting with coaching role

In January 2008, their agile setup had adapted with a coaching role. The coach works directly with a sub-team to provide suggestions for process improvements, on self-management and on problem-solving approaches. This is another of their informal roles that a senior engineer is expected to undertake. This role was proposed by the principal

engineer due to the agile team recruiting ten additional engineers in the last quarter of 2007. The agile team collectively agreed to have a senior engineer in a coaching role to ensure that all sub-teams and engineers develop universal team behaviour and follow agile practices. While a senior engineer could volunteer for this role, the team and technical leaders, product analysts and the engineering manager collectively selected the person for this role. Here, the individual in this role works with sub-teams, pairing and coaching them for a period of time to build a shared agile culture.

> Started recently to use a coach … certain team behaviours are not beneficial … someone looking from the outside … having a coach associated with different teams is extremely beneficial. (Engineering Manager)

> Process coaching is what's important rather than the technical coaching … it's important to coach the tech leads and the team leads … everyday have a stand up, and every day rotate pairs … work on small stories … to deliver the most important business value first. (Principal Engineers)

4.3.2.2.7 Adapting with consultants

In April 2005, the team hired a consultant to up-skill the engineers with test driven development (TDD) practices for implementing unit and acceptance tests. With no separate quality assurance team, TDD was one of their most important agile practices and required individuals to quickly learn this approach for implementing code. The decision to hire a consultant was made by the engineering managers since the team had no one with test driven development experience. However, a senior engineer proposed that the best approach for the team to learn was to pair with the consultant for a period of time. The team agreed with this approach and during their daily stand up meetings they themselves decided which pair of engineers would work with the consultant for the day.

> For specific technology … more efficient to skill the entire team [than] to work with one individual … this fosters competitive instead of collaborative behaviour … we identified areas where we lack the knowledge and the skills (Engineering manger)

> Consultant … chance to pair with him … you see how he does things … you learn from practice …everyone gets a chance. Ken Beck [author of XP method] had a meeting with all of our team members … help us with estimating. (Engineer)

4.3.2.2.8 Adapting with agile testers

Akldevelopment had adapted to developer led quality assurance since improving its development technologies and infrastructure in early 2005. Their agile setup does not

have dedicated testing roles. In October 2007, they adapted to include an agile tester with their agile team.

> The role of a [Agile] tester is different … make changes to the tests [engineers write unit tests] … guide and help the engineers set up the test frameworks … automate a whole bunch of systems testing. (Principal Engineer)

The idea to have agile tester was proposed by the principal engineer and was accepted by the team including their engineering manager. The team had discovered that the engineers were limiting their thinking for writing the unit tests to story cards only and the product quality was being compromised. They needed this role to provide development support so that a wider view is considered by the engineers on issues such as performance and usability. Engineers now have support to write unit tests. The idea to have this role was also supported by the product analysts since the agile tester would also provide them with support for writing acceptance tests.

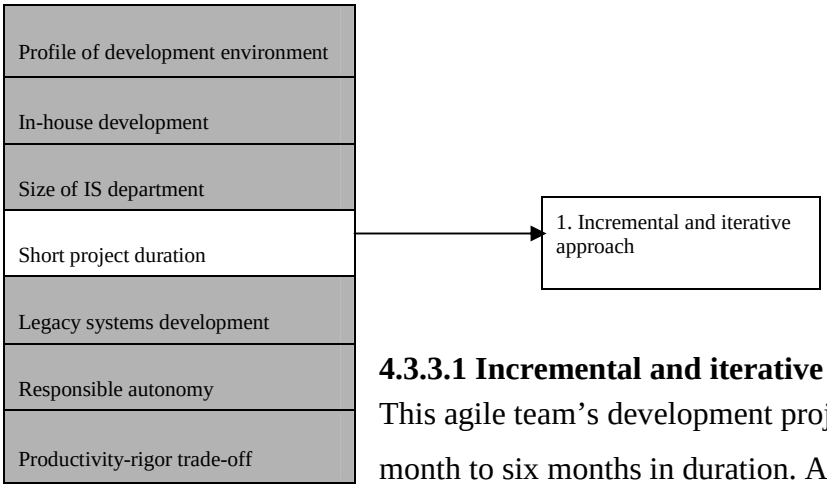
> Developers think of quality as to get this screen or piece of function or calculation working … happy to drive [it through] unit tests … tester can think about performance, about the edge conditions … help product analysts think how to make the application testable … help write acceptance criteria … get quality at the very beginning. (Principal Engineer)

Next, information is provided about the short project duration at Akldevelopment. This is identified as another adaptation factor under the profile of the development environment in Fitzgerald's adaptation framework, as illustrated in Figure 18.

4.3.3 Short project duration

The finding in relation to this adaptation factor is presented according to the theme that emerged from the analysis of data collected from Akldevelopment; the incremental and iterative approach

Figure 18 Incremental and iterative approach in relation to short project duration - Akldevelopment



4.3.3.1 Incremental and iterative approach

This agile team's development projects are mostly one month to six months in duration. As a result, they have three to four major internal feature releases per year. To achieve their release targets, they have adapted their agile development with an incremental and iterative approach. As a result, all their projects are story driven with short development cycles. Each of their short cycles is known as a sprint, which combines to make a release cycle. Here, this practice makes it possible to package new features that are implemented up until a major release date.

> Have short release cycles, every 3 months a new release … decision from marketing what kind of label they want to stick on it … a major or a minor release … is a business decision. (Engineering Manager)

4.3.3.1.1 Adapting sprint cycles

In January 2005, Akldevelopment's agile team adopted one week for a sprint cycle. For each sprint cycle, this agile team negotiates and commits to deliver the prioritised stories from their sprint backlog. This decision for a one week cycle time was made by the engineering manager. The team accepted this cycle time since it provided more frequent and regular check points allowing better tracking and reporting for undertaking projects. However, the team found that the majority of the stories were taking them more than a single sprint cycle to implement.

> Adapted the length of sprints … started with one week … if the sprints were longer [need] more check points. The story has to fit within one sprint. (Senior engineer)

In June 2005, the agile team decided to adapt from having a week for a sprint to two weeks for a sprint cycle. At that point of time, one week for a sprint was regarded as too short since most of the stories required more than a week for their implementation. Despite adapting to two weeks sprint cycle time, the team experienced taking more than

two weeks to implement some of the stories requiring rescheduling in the next sprint for their complete implementation. In October 2005, the team collectively decided to adapt to three weeks for a sprint. However, the senior engineers noticed over a period of time that three week sprint cycle time had little productivity benefit since stories were being implemented two or three days before the finish of a sprint. The team also found that the majority of their stories were being estimated to be implemented in less than two weeks. During a sprint review in December 2005, the team decided to adapt back to a two week sprint cycle time. As the team accumulated development experience and capability, the majority of their stories were broken down to be implemented taking less than a week in time. Hence, the team agreed to the senior engineers' proposal to adapt to a one week sprint cycle time in August 2006. With this adaptation, the team became very productive implementing more stories in one week sprint cycles than they did in two or three week sprint cycles.

> Have one week sprint, works best for us … make sure that stories fit into two thirds of a sprint … tried 2 and 3 weeks, didn't work out … reason for longer sprints were the size of the stories … smaller sprints are easier to handle … sign off more stories … get a regularity of progress. (Senior Engineer)

Next, information is provided on legacy systems development at Akldevelopment. This is identified as an adaptation factor under the profile of the development environment in Fitzgerald's adaptation framework, as illustrated in Figure 19.

4.3.4 Legacy systems development

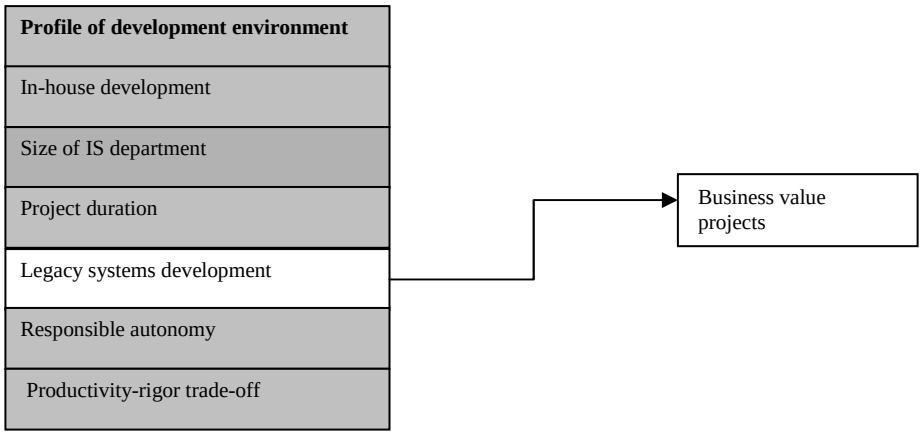
The findings in relation to this adaptation factor is presented according to the theme that emerged from the analysis of data collected from Akldevelopment; the business value projects

4.3.4.1 Business value projects

Since 1987, Akldevelopment's single code base for its core product is continuously enhanced with new features including improvements to the existing features and integration abilities with third party software. All their development is related to enhancing their legacy product. Since the adoption of the agile approach at Akldevelopment, the process used to request budgets to carry out improvements require stating business cases for such developments, just like for any new feature development.

> Build a business case to ask for the funding … give some indication of how many dollars is [required]… with agile you get approval to change the software. (Senior product analyst)

Figure 19 Business value projects in relation to legacy systems development - Akldevelopment



4.3.4.1.1 Adapting product backlog

The requests for change come from various sources but most are made by the clients. The internal source for change is mostly from sales and marketing teams. Since November 2006, the product strategy group compiles the request for changes while the agile team provides the business case for each of them. The product strategy group recognised that they must adapt how they prioritise the requests for improvements when they have both internal and external requests for changes at the same time. At Akldevelopment, customer requests are usually assigned higher priorities regardless of the cost. The customer-driven improvements enable a better product quality and often new customers show interest in their product leading to signing purchase deals. The cost of such changes may vary from one engineering week to even bigger engineering investments involving significant improvement of the product usability. Any such improvements are funded by the company itself rather than by the existing clients but they still receive improvements as patches. Hence, adapting how product backlog is prioritised ensures business value implementation.

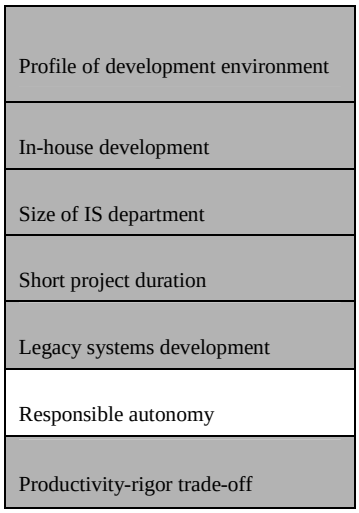
> Very important [is] client's feedback … deliver that through our own investment … [clients] make ideas and suggestions … goes into the ideas for investment … broken into buckets [priorities], things that would make our client's life easier. (Senior product analyst)

Next, information is provided on responsible autonomy at Akldevelopment. This is identified as an adaptation factor under the profile of the development environment in Fitzgerald's adaptation framework as illustrated in Figure 20.

4.3.5 Responsible autonomy

The findings relating to this adaptation factor are presented for Akldevelopment.

Figure 20 Developer autonomy - Akldevelopment



4.3.5.1 Engineer autonomy

This agile team initially had to adapt to have an empowerment culture for product development. While Akldevelopment had autonomous development teams, they were highly structured and managed by engineering managers. Here, this role essentially is a project manager role usually having an authoritarian management style. However, the agile team engineers are empowered with project management and implementation responsibilities, including a wide range of decision making in relation to their development process.

> Engineers do the job of a project manager … from a set-up perspective, I am the project manager … responsible for money … a time plan … risks and issues … but the daily operations done by the teams … a project manager would be inward and downward focused … [now] the opposite, which is upward and outward. (Engineering Manager)

> Delegated estimation … signing up for work for a particular sprint … selecting technologies … selecting people for the team … changing the design all the time … refactoring the code (Developer).

The engineers are entrusted with accountability for their agile process. The belief is that when engineers contribute and implement ideas for their practices, it is much easier to get the team to buy in. Such autonomy makes their team a lot more passionate and motivated when the manager does not micro-manage the team.

> Autonomy contributes towards motivation, commitment, dedication and high performance … in charge of their own destiny, so getting the buy-in of the team is easier … [with] autonomy reflect upon how they work, what tools they use, is the technology right, is it right to be designing it this way. (Engineering Manager)

> Decision making, of course I do all the time … it's kind of hard at first because you seem to have too much freedom … after a while you definitely feel confident … definitely appreciate the freedom … motivates you to produce more ideas … don't have the ability to contribute then how can be given empowerment to contribute (Engineer)

4.3.5.2 Adapting engineer empowerment

This agile team had full delegation for all their development practices and decision making within their agile setup. One key adaptation with their empowerment is that the

engineers are allowed to negotiate story estimates during their sprint planning meetings. This decision was proposed by the engineering manager and collectively agreed by the team in August 2005 when the team decided that only a few senior engineers would be involved with on-site customers for backlog planning. Despite the senior engineer involvement in story planning, this adaptation was made to make sure that the entire team takes full responsibility for delivery of their sprint commitments. Through this adaptation the team has experienced that more reliable estimates are being made and also, they are regularly delivering their team commitments.

> Engineers sign up for an amount of work …they determine the actual velocity … asked to come up with their own estimates for every single story … do whatever it takes, long hours and weekends to deliver what they have estimated and signed … everybody else is basing their plans upon what they have committed. (Engineer Manager)

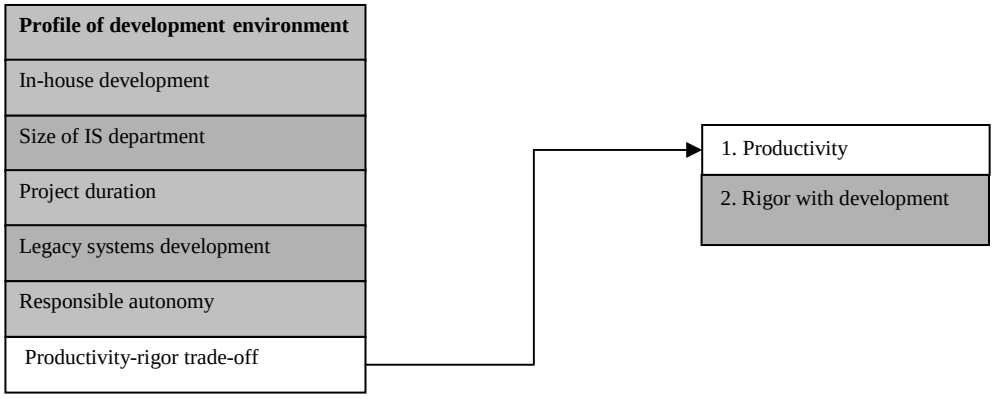
> If a team is committed to a certain amount of work this week the people outside of the team need to make sure that they're not interfering … manager has to provide that framework. (Senior Engineer)

Next, information is provided on the productivity and rigour trade-off at Akldevelopment. This is identified as an adaptation factor under the profile of the development environment in Fitzgerald's adaptation framework as illustrated in Figure 21.

4.3.6 Productivity and rigour trade-off

The findings in relation to this adaptation factor are presented according to the themes of productivity and rigor within the development process, which emerged from the analysis of data collected from Akldevelopment. Productivity and rigour are as important as each other at Akldevelopment.

Figure 21 Productivity in relation to productivity-rigor trade-off - Akldevelopment



4.3.6.1 Productivity

Productivity with this agile team means to have the development agility to implement innovative product features cost-effectively and to achieve their target of three monthly internal releases. At Akldevelopment, improving productivity is one of the key product development goals enabling them to keep ahead of their competitors in the international marketplace.

> Need something that can sell … successful in the marketplace … provides value to the customers … deliver that in a short timeframe, at least faster than the competition … we have accomplished our job. (Engineering Manager)

> Software is hopeless if it's not delivered … it is no use to anybody … a team that communicates well … carries the water for each other … there's absolutely shared responsibility … one pair didn't finish a story … everyone has to work to get that stuff done. (Senior Engineer)

The following are some of their key productivity practices:

4.3.6.1.1 Effective sprint plan

The sub-teams are required to have a good sprint plan clearly indicating the commitment for a sprint cycle. A good sprint plan contributes effectively towards the team mental state, facilitating coordination, fostering efficiency and promoting predictability. A sprint planning meeting involves negotiating the story estimates with the product analysts so that the sub-teams are as close as possible to what they think will be required to implement it. An effective sprint plan enables the agile team to implement as many stories as possible in their release cycles.

> They size [estimate] them [the stories] at the start of the week … this is what we're going to do for the week. (Product Analyst)

> Able to tell people exactly what we planned for an upcoming release … we know exactly what we [will] deliver … plan for it as good as possible. (Engineering Manager)

4.3.6.1.2 Team work area and co-location

The agile team is provided with a separate team work area. Engineers also have their own desks in the personal work area. Their work time is divided into engineering hours (team meetings and pair programming) and personal hours (dealing with matters such as checking emails, doing research, and learning a tool). The sub-teams and their product analysts with the usability engineer are all co-located in an open space. The co-location provides for effective face-to-face collaboration and coordination with instant access to others for discussion in their agile product development environment.

> People are co-located … improve communication because people are closer … whenever you have a question; go to person in regards to providing that detail. (Engineering Manager)
>
> Work really well when everyone is co-located … if the customer is in the room listening, talking, and asking questions, it really helps. (Senior Engineer)

The team work area is where the engineers are expected to spend most of their time. This area has dedicated work stations with internet access for job related tasks. In the team work area there is no email access, instant messaging systems or phone facilities. The team work area ensures that there are no outside distractions. Individual phone facilities and internet access for personal use are provided for in the personal work area.

> The team area is non-office work and non email area … this is the actual development area … have all tools available … as opposed to the individual work space, do their office and personal work. (Engineering Manager)
>
> Open space does matter to me … see other people working and sometimes I'll slack off a bit … makes me feel bad … I'll try keep up to pace with everyone else … need help I can always go to them … makes me more confident. (Engineer)

4.3.6.1.3 Pair programming

Here, the pair programming practice is used as way to have a productive agile team. The driver and navigator roles help engineers to focus on what is required to be done.

> Work in pairs, can't lean back … able to achieve within 6 hours of pair programming more than most people achieve within 12 hours of working solo …there's no deviation into other adjacent areas. (Engineering Manager)
>
> No skills whatsoever in the platform … pairing is an accepted way for to get up to speed … might not be the most productive … people tend to go off at high speed and massively refactoring the code that they don't like … with some senior person, they just charge without discussing … certain personalities can be difficult. (Principal Engineer)

4.3.6.1.4 Minimum design

The agile team has no separate design phase, since all of their functional requirements are fleshed out as stories. The stories are written as individual tasks which require a minimal design effort for pairs implementing them.

> Don't have much of design phase … our stories are very short , a small amount of work … just a note pad with a couple of scratchings is good enough to get on the same ball park. (Engineering Manager)
>
> You want to go in there and start coding … if you are doing unit testing then you have to know what interface is going to be, this is what is going to go in and come out. … that forces you to do a bit of design. (Engineer)

4.3.6.1.5 Good tool support

The agile team provides tool support for the engineers making sure that they have everything needed to develop and test software in a productive manner. They believe that good tool support removes the overheads and barriers that may impact on development efficiency. Their development environment has been adapted to include powerful version control systems and an Integrated Development Environment (IDE) with source code editor, a compiler, build automation tools, and a debugger.

> Our goal is that the build should finish in 10 minutes … whenever it goes up we find another way of bringing that down. (Engineering Manager)

> Have also got end targets for our build process … run all the tests as part of the build … engineers must get knowledge of broken builds as soon as possible in shortest time to be productive with TDD. (Principal Engineer)

While these practices have been consistently applied, the team had recently adapted a few other practices to make their agile development more productive. These are discussed next.

4.3.6.1.6 Adapting story size

Their requirement for a story is that it must be discrete. Now, at Akldevelopment each story must be sized up to two days or less for its implementation. This adaptation was agreed by team in August 2006 based on the proposal by the senior engineers for such a story size to enable one week sprint cycles. With this adaptation the team found a story this size much easier and faster to implement and test. This adaptation also enables them to commit more stories in their one week sprint cycles, becoming more productive.

> 1 week sprint, do 10 or more stories … each story is a very gradual piece of work … like 1 or 2 days worth of work … break your tasks down to that small amount of work. (Engineering Manager)

4.3.6.1.7 Adapting with no comments for code

The principal engineer proposed the practice of writing no documentation for code which the engineers implemented to enable the team to effectively deliver their commitments in one week sprint cycles. In August 2006, the agile team adapted with a development standard where the code must itself describe to the reader what the function is about and not requiring engineers to write code documentation. However, after a period of time some engineers identified that for complicated implementations this practice was not effective. In October 2006, for such situation the team agreed to

provide minimum comments to make the code readable if engineers thought it was necessary to do so.

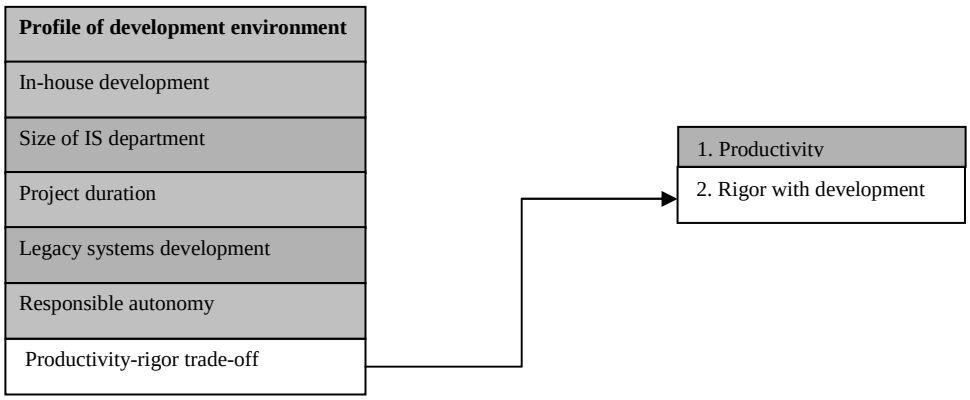
> We use no comments in our code … there is a rare occurrence, might need to put a comment … so that's another principle. (Principal Engineer)

Next, information is provided on rigor with the development process, as illustrated in Figure 22.

4.3.6.2 Rigour with development process

The agile team has a rigorous testing process with different types of automated tests to ensure high quality products. This is achieved through adapting to use a test driven development (TDD) approach with an automated testing environment. Prior to agile adoption, Akldevelopment had no automated testing environment.

Figure 22 Rigor with development in relation to productivity-rigor trade-off - Akldevelopment



4.3.6.2.1 Unit tests

The agile development engineers are required to write unit tests for stories that they implement. All their code is driven by unit tests where the engineers write unit tests to validate individual functions. Here, they write a unit test before writing any code for a function related to a story a pair is implementing. This practice ensures that their quality standards are being met and to identify and eliminate defects early in their development process. This practice also provides them with the benefit of refactoring (making the code as efficient as possible) as a story is implemented.

> Always write unit tests before code … it is not like we spend 3 hours writing unit tests and then 5 hours coding … spent 30 seconds writing a unit test, one line of code … refactor it … it happens every minute. (Principal engineer)

Akldevelopment's agile test driven development has an automated and continuous integration system which helps to detect integration problems, gives an early warning of

broken code, instantly evaluates unit test changes, and facilitates the availability of builds for systems testing, product releases and marketing.

> Our software sells for multi million dollars … is huge, can't run all the tests every single time you change a line of code … need to use integration system to make sure that when I commit some code, don't break [other] modules. (Principal Engineer)

4.3.6.2.2 Adapting unit tests with mutation tests

This agile team has adapted to using the JUnit testing framework for writing unit tests. In November 2006, based on a suggestion by the engineering manager, the team adapted to enhance their TDD practice by including mutation testing to help determine the effectiveness of their unit tests. Using mutation test, the engineers are now able to determine the quality of their unit tests during implementation and can improve them to enhance the overall quality of the new features.

> Mutation testing … mutate the code and run all the unit tests again, if they still pass then something is wrong … definitely have a defect … is basically about assessing the quality of the unit tests. (Engineering Manager)

4.3.6.2.3 Systems testing

Before a major release, Akldevelopment has a four week systems testing phase for their product. With this agile team, they have adapted to use an automated acceptance testing environment for their internal releases. They have adapted to use a complete automated acceptance testing environment for their end-to-end functional tests replacing manual systems testing. The acceptance tests run constantly with their continuous integration systems to make sure that their code achieves quality gains.

> Don't do system testing whatsoever … have automated acceptance testing … alternatively run manually, not the best approach towards improving productivity … in that sense we have adapted, built in the acceptance tests from day one. (Engineering Manager)

> But they [on-site customer] would go at their own time and just look at the screens and make sure things look fine. And then try a few of the functions on the screens and then they leave. (Principal Engineer)

The team adapted to use the Fitnesse testing tool to automate story tests, enabling them to test business logic. This tool allows their product analysts to explain a story using a tabular set of data. The engineers implement the code behind the test and the product analysts will test to see if that code is working.

> Fitnesse test … we use a tabular set of data inside a Wiki that lets the customer express concepts through tables of data … tests become part of the normal continuous integration and build system. (Principal Engineer)

4.3.6.2.4 Adapting acceptance tests

The agile team adapted to use Selenium as a test tool for testing the user interface by running it directly in a real browser such as Firefox or Internet Explorer. Selenium involves writing HTML pages that contain Selenium tables. Selenium is preferred to other testing tools that simulate a browser. However, some engineers found implementing the Selenium test challenging.

> It is important that we write functional tests in the browser that tests the stuff that we can't test in Java …tried Selenium, a much better way of testing the UI … strong and powerful. (Engineering Manager)

> For me to write a Selenium test straight off and start coding the web page is very difficult because I don't know what the IDs will be and some of the web page stuff is automatically generated. (Principal Engineer)

In July, 2005 the agile team agreed to adapt their Selenium tests by writing Java code that got executed during the build process to produce HTML code. This adaptation was proposed by engineers since they could not refactor HTML code. The agile team accepted this adaptation as necessary for them to be able to refactor all their implementation to make their code more effective, as there is no refactoring tool for HTML code.

> It is easier for us to refactor that code, write that code in Java. (Engineering Manager)

> Writing J unit tests and then writing Java code … is fine for me because I understand Java. So writing a J unit tests for me isn't really that difficult. (Principal Engineer)

4.3.6.2.5 Adapting build system

With this agile team, the committed code was put into a single instance of CruiseControl, a Java-based framework for a continuous build process with their version control system. Overtime the code base grew so that it was no longer efficient. In August, 2005 their build system was adapted into smaller CruiseControl groups where major components have their own CruiseControl. The agile team adapted their build system to make it more efficient. Initially, their build process would take up to four and half hours to build once code was committed. This duration for build process was quickly recognised by the engineers as inefficient because they had to wait for long

periods of time to know about the broken builds. This resulted in engineers not able to deliver their sprint commitments on time, fixing the build from the previous sprint during the next sprint cycle.

> When we started, the build would take up to 4 ½ hours … have adapted in such a way that we can split it up … [for] major components one cruise loop each … have about between 13 and 15 cruise loops. (Engineering Manager)

> Running the build, we need more hardware … takes too long to run the build … so that individually we could run our builds over many machines and it would be much quicker. (Engineer)

This adaptation allows more frequent builds where they are able to run ten to fifteen builds per day from two or three builds per day in the beginning. The engineers get feedback on the broken builds more quickly helping to increase the team's productivity in meeting their sprint commitments. In the beginning, the agile team experienced a lot of broken builds, which now has improved considerably.

> We could have built the entire system in a single cruise group, within 10 minutes … but we would not build more often … slow down teams. (Engineering Manager)

4.4. Overt factors

This section provides detailed information on Akldevelopment's agile approach adaptation driven by the overt factors or intellectual roles of software development methods. Figure 23 shows these overt factors as identified in Fitzgerald's adaptation framework. First, information on project management adaptation at Akldevelopment is provided

4.4.1 Project management

This section provides adaptation information to improve visibility of development projects as illustrated in Figure 23. Improved visibility is part of project management as an adaptation (overt) factor in Fitzgerald's adaptation framework.

4.4.1.1 Improved visibility

At Akldevelopment, both planning and executing projects are regarded as equally important; their agile product planning tasks maximise the visibility of upcoming features. The product vision and iteration plans are discussed. Information on adaptation of these plans to improve visibility is also provided.

Figure 23 Improved visibility in relation to project management - Akldevelopment

Overt factors
Project management: improved visibility, reduced risk
Reduction of variety and complexity
Economic: skill specialisation, division of labour
Epistemological: transfer of knowledge, template for inexperienced developers, learning from past projects
Facilitation of intercommunication among developers

4.4.1.1.1 Product vision (high-level plan)

At Akldevelopment, considerable effort goes into developing high-level plans. It is an important artefact to establish and to communicate visions of their new developments company-wide. The product managers have this responsibility. Their business-level planning strategy to identify high value features creates the awareness for the new features within and outside the organisation. With the agile development team this is achieved through their planning practice of creating a release backlog, which is driven by the product analysts. Information on high-level and low-level planning including how and why they are adapted has been provided with the in-house development factor in the profile of the development environment.

4.4.1.1.2 Sprint plan for story implementation

With this agile team, a sprint plan identifies the schedule of all the stories that they commit to implement in a sprint. Here, sprints are allocated stories from a release backlog according to the priorities, which are market-driven. The sprint plan is an outcome based on agreement between their engineers and the product analyst. The business functions such as the sales and marketing departments are dependant upon the sprint output, the build. At Akldevelopment, the visibility of each sprint is extremely important since the stakeholders need to know in advance the availability dates of the implemented stories. However, this is dependent upon the accuracy of the story estimates.

> Break down the functionality and lay it out in terms of releases … once the sprint has started … team tracks that every single day … walk to the whiteboard and see exactly if they are ahead of, behind or on schedule (Engineering Manager)

> Each Sprint we are not really tracking the amount of time it took to do certain tasks … we are not collecting data … people just kind of roughly remember how much time it took to do something … don't even know if that would be useful, every task you do is quite different. (Principal Engineer)

4.4.1.1.3 Adapting story priority setting

This agile team has adapted how the priorities are assigned to the stories in order for the team to easily schedule stories within a sprint. They were having stories of equal priorities making it hard for them to reshuffle the order of the stories for implementation in a sprint. Some of the senior engineers recognised that two stories with the same priority caused unnecessary debate about which story should be implemented first, resulting in precious time being wasted during sprints. In August 2005, the team collectively agreed with the product manager that no two stories should be assigned the same priority. Previously, stories were given a priority of high, medium, or low. Now, all their stories differ from one another in priority.

> Can't have two stories with exactly the same priority … to decide between two, because able to do one … it becomes even harder and it takes up more of the team effort. (Senior Engineer)

4.4.1.1.4 Story estimates

At Akldevelopment, there are three points for story estimation. One is with the vision and roadmap plan, where they estimate on a large scale the releases that they will do for the next 12 months. This is a high-level story estimate involving a few senior engineers. The next estimates are done when planning a release backlog, which also involves only a few senior engineers. The final estimates are made in sprint planning meetings when the agile team engineers collectively provide estimates for stories against which they will work in a sprint.

> Higher level stories get broken into 2 or 3 different stories each [backlog planning] … the engineer sizes [estimates] them … go through each story [sprint planning] … team determines how many hours they need. (Product Analyst)

4.4.1.1.5 Adapting estimating practice

Despite having three different points for estimation, on occasion it is quite a challenge for them to deliver their sprint commitments. Over the projects, the engineers in this agile team experienced that the most of stories relate to new technologies to improve client performance. Several of their stories require a good understanding of such technologies for the engineers to successfully implement them. Hence, some of these stories require further investigation before their implementation. For this reason, the engineering manager proposed to the engineers that they provide estimates in points rather than in hours, giving them a buffer for gaining a better understanding of a story before its implementation. As a result, this agile team in January 2007 agreed to adapt

their practice of providing estimates from hours to points. Here, one story point is equivalent to one ideal day or 8 solid hours of pair programming.

> Moved from hours into story points … look at all the stories … identify the smallest story in terms of the least effort …compare that story with every other story … it can be equal size … it can be 1.5 or it can be 5 … can apply any scale. (Engineering Manager)

> 0.5 is half an ideal day … doesn't mean like half a real day. It could be like 1 or 2 real days … an ideal day means an engineer sits there for 8 hours. (Engineer)

Previously, an engineer proposed an estimate for a story. However, it was observed by some of the senior engineers that for the majority of the time during sprint planning meetings an estimate suggested by an engineer was quickly agreed by others without much thought or discussion. The principal engineer suggested using the planning poker game to generate discussion before the team committing to a story estimate. Hence, the team agreed in March 2007 to adapt their estimation practice for sprint planning meetings with planning poker. This adaptation has enabled the team more discussion helping them to identity issues with stories and providing more accurate estimates for their implementation.

> All have cards with different story points on them, from 0 to 5+ … all draw out one of the sprint poker cards … place them on the table at the same time, if 2 people bring out a story point of 1 … 8 people bring out 2 points for that story , more likely put 2 down on the story card. (Engineer)

Next, adaptation information to reduce risks in relation to project management at Akldevelopment is provided, as illustrated in Figure 24.

4.4.1.2 Reduced risk

Discussed next are some of the key risk factors that may impact Akldevelopment's new development. Information is provided on adaptation of their agile approach with new practices to minimise the impact of these risk factors.

Figure 24 Reduced risks in relation to project management - Akldevelopment

Overt factors
Project management: improved visibility, reduced risk
Reduction of variety and complexity
Economic: skill specialisation, division of labour
Epistemological: transfer of knowledge, template for inexperienced developers, learning from past projects
Facilitation of intercommunication among developers

4.4.1.2.1 Unfamiliar technology

It is a constant challenge to provide accurate estimates since stories frequently involves new technologies for which the team lacks knowledge. For this reason, the team has adapted its agile development with a practice known as a 'spike'. On occasions, the team runs a spike; it is a practice of allocating time to learn about and consider technological issues before being able to implement stories within a sprint. If realised before or during a sprint planning meeting, a spike is run as a story with other story implementations before any reasonable estimates are provided for it.

> The risk … new technology the team is not yet as familiar … a spike; you need to invest a few hours in order to determine the actual effort. (Engineer)

4.4.1.2.2 Outdated backlog plan

For this agile team, there was also a constant challenge to allocate time for re-evaluating backlog plans. Having adopted sprint cycles as a method for allocating and carrying out their development tasks, the engineers are constantly implementing stories from one week to another. There is always the risk that the senior engineers may not re-evaluate and provide an updated backlog. This leads to decisions being made based on outdated estimates. They have adapted their agile approach with a practice (it makes sure that time is set aside) that involves evaluating the estimates of the next release in their current release cycle.

> Our strategy … regularly update and revisit all the estimates [for] next release or next 2 releases … estimate still looks okay. But this is not only about the size of the different stories, it's also about the content and scoping of stories and the prioritisations … revisit all of those. (Engineer Manager)

4.4.1.2.3 Missing low priority story implementation

The agile team have missed out implementing some of the low priority stories. These missed items are the missing functionalities for a new feature which Akldevelopment is trying to provide with its product. These types of stories are low priority but are considered to be important, adding value to new features. For this scenario, they incorporate their agile approach with a shorter iteration cycle (less than a week long) to implement those stories.

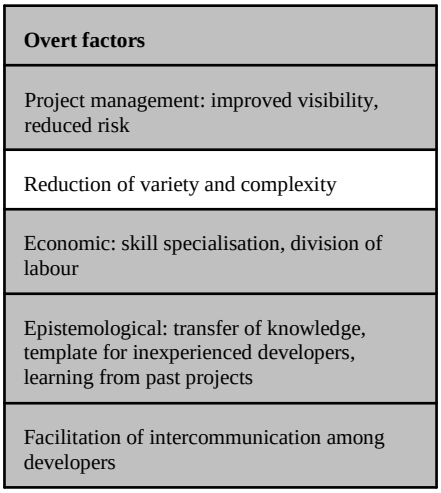
> The risk, miss some items of low priority … have short iteration … allocate it for next week … mitigate that risk by making sure that system is always completely refactored, clean and simple design … add whatever you might have overlooked. (Engineering Manager)

Next, adaptation information is provided on reduction of variety and complexity (Figure 25).

4.4.2 Reduction of variety and complexity

This section provides adaptation information on release and daily plans at Akldevelopment.

Figure 25 Reduction of variety and complexity factor – Akldevelopment



4.4.2.1 Roadmap plan; release and daily plans

Preceding Akldevelopment's vision planning is their product road map planning. Their roadmap plan identifies the quarter releases in which the new features will be included. This plan usually covers a roadmap for up to twelve months. The initial reduction of variety and complexity is achieved through their roadmap plans. It includes the creation of separate backlogs for individual releases with a list of stories.

> Have a roadmap … contains large functional areas that we would like to address at a certain point in time … within these have release plans … break down the functionality and lay it out in terms of release … towards the end of a release look at the next release. (Engineering Manager)

> The [on-site] customer and one or two engineers plan releases … invited all engineers to the meeting and show them … basically broken down into stories and also are prioritised … like high priority, medium and low. (Engineer)

The other important aspect of a release plan is that it identifies all the sprints leading up to a specific release date. A further reduction in variety and complexity is achieved through their sprints. At a micro-level, sprints provide for further reduction in variety and complexity through their daily scrum meetings, which require the engineers to communicate their individual plan for the day. Next, information is provided on their daily scrum practice including how and why they are adapted.

4.4.2.2 Scrum meetings

Since January 2005, the daily scrum meeting is considered an integral part of their agile approach where the pairs provide an update on their story implementation from the previous day and provide a plan for the current day. This agile team considers scrum meetings to be one of their key practices that enable them to behave and feel like a team, facilitating the engineers to pick whom they will pair with for the day or for a story. The team's scrum meeting, held at 9 am and time-boxed for 15 minutes, is when they all meet for the day. This meeting is also open for individuals from other teams or departments to attend as observers.

> The most efficient one … igniting a team spirit … a synchronisation point, everybody gets a sense of who is working on what, announcements can be made … people report what they did since last scrum … also the plan for the next thing. (Engineering Manager)

> In scrum … talk to other developers and on-site customer … what you're going to do … you know about what other people are doing … have a better picture of how the entire team is going. (Engineer)

4.4.2.3 Adapting daily (scrum) plans

The agile team themselves acknowledged that since they are co-located in a common work area, they are aware of each pair's daily progress. Hence, the engineers felt that updating their progress in scrum meetings did not provide any value. During a sprint review meeting engineers collectively agreed to adapt the format of their daily scrum meetings based on the suggestion by the engineering manager. In November 2006, the engineers decided to identify issues and items of team interest in their scrum meetings together with identifying who is pairing with whom for the day. The team also decided to adapt to have another meeting on the issues identified during the scrum meetings immediately after involving those who were are interested to discuss and identify possible actions. This adaptation for a follow on meeting became a popular means of helping a pair by another pair on their previous day's issue.

> Identify issues [and] item of interest for the rest of the team … the second element; switch partners … with regards to who is moving and who stays with the story … determine who is pairing with whom. (Engineering Manager)

The daily scrum meeting was further adapted in January, 2008 due to the increasing size of the agile team which had doubled in size to have thirty-six software engineers, one usability consultant, one agile tester, and four product analysts. Some of the senior engineers realised that it was not practical to have scrum meetings involving the entire agile team. Hence, when creating the sub-teams for projects the team leaders, technical

leaders, product analysts and engineering manager jointly made a decision to adapt the agile team's scrum meetings to having sub-team scrum meetings. This agile team agreed to this format since a sub-team works on a distant part of a project having no shared components that need to be attached as part of the stories. The engineers now experience the benefits of having scrum meeting in their sub-teams which are simple for coordinating their daily pair swapping and short in duration focusing on identifying issues relating to their sub-teams only. In March 2008, the agile team had another adaptation in relation to sub-team meetings where technical leads of sub-teams now have joint meetings. Despite working on distinct parts of a project, engineers discovered that some issues definitely required communicating to the other sub-teams. Hence, a collective decision was made by the agile team for a practice requiring joint meetings between technical leaders when required to communicate major project issues to engineers in sub-teams. There is now collaboration between product analysts on story priority settings for their sub-teams and more cross team collaborations between engineers during sprints avoiding frequent broken builds.

> One large stand up meeting … it becomes a waste of time … people [are] working in different areas … also there is just a small amount of time … once a tech lead identifies an item that needs to be covered with other teams … tech leads get together and have a chat … works quite well. (Engineering Manager)

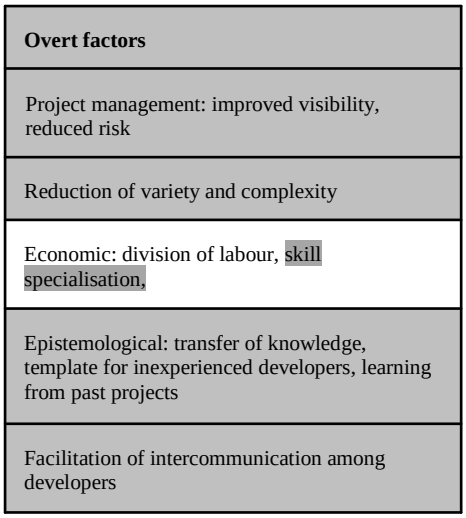
> Scrums are for only for people who are actually involved in the things that they're doing with a [on-site] customer … people working on two different parts hardly overlap … so there is not much point putting the two teams together for scrum meetings. (Engineer)

Next, adaptation information is provided on economic: division of labour and skill specialisation (Figure 26).

4.4.3 Economic: division of labour and skill specialisation

First, adaptation information is provided on Akldevelopment's division of labour with their agile approach.

Figure 26 Economic division of labour - Akldevelopment



4.4.3.1 Division of labour

Akldevelopment now implements any new feature through their two distinct functional units within the engineering department; the development and documentation teams.

4.4.3.1.1 Adapting with business contribution

Their agile development now includes daily contributions from the product strategy group. Their agile setup is adapted with full-time product analysts assigned to sub-teams. In January 2005, a product manager was co-located to provide domain support for the agile team. The engineering manager, who made this decision, identified product manager support as critical for creating effective sprint backlogs and for providing engineer support for swift story implementation during sprint cycles. However, this support was not available when the product manager was in the field working with clients. Upon realising this, engineers collectively made a request for a permanent on-site customer role. Based on the decision made by the company executives, the product manager adapted to become product analyst and permanently co-located with the agile team in November 2006. This adaptation enabled a permanent cross-functional agile product development between engineering and business functions at Akldevelopment. This agile setup now has a clear division of labour at engineering level, whilst still working together as a team. Their agile projects require a full commitment from their software engineers and the product analysts.

> People who are contributing towards a project are co-located and work as one team … not only the engineers … [but also] product [analysts] … these people have to collaborate … have co-located them [to] make sure that we spend as little time on communication overhead. (Engineering Manager)

> With agile, communication thing is so important … everyone is generally co-located (Principal Engineering).

4.4.3.1.2 Adapting development function

Their agile development is adapted to have three distinct engineering roles. These roles now are software engineer, agile tester and usability engineer creating a division of labour within their development function (reasons for adapting with agile tester and

usability engineer roles have been provided earlier). In January 2005, the development roles were adapted into a single software engineering role enabling this agile team to have multi-skilled and flexible individuals. This decision was made by the engineering manager incorporating testing and quality assurance tasks up front with development to deliver implemented features in short development cycles. As such, there is no division of labour amongst their software engineers, despite having a vast majority in this role. As a result, engineers now carry out all the development tasks. The Engineering Manager explains the reasons for adapting to a single software engineering position with their agile setup;

> Agile tester… provides feedback … on how the quality [must] be built into the product … user interface expert, influence achieving [usability]. Architects, analysts, designers, implementers and testers, aggregated into software engineer [role] … can't separate them, start changing the code, essentially changing the design … testing and implementation can't be separated, otherwise responsibility for quality will be distributed … has to remain with engineers … also architecture is something that emerges over time. (Engineering Manager)

> Waterfall method … somebody would write the requirements … someone would come up with a solution … someone would write a design and would throw it over the fence … developer would pick it up and go why did the designer do this … where did the designer go … can't go back and redo this bit … if somebody [agile engineer] makes a mistake … know about it immediately and they can fix it. Rather that it not being noticed for 6 months. (Senior Product Analyst)

Next, adaptation information is provided on skill specialisation (Figure 27).

4.4.3.2 Skill specialisation

Figure 27 Economic; skills specialisation – Akldevelopment

Overt factors
Project management: improved visibility, reduced risk
Reduction of variety and complexity
Economic: division of labour, skill specialisation,
Epistemological: transfer of knowledge, template for inexperienced developers, learning from past projects
Facilitation of intercommunication among developers

The software engineers are required to adapt to broad technical skills along with their specialist skills sets. Adapting to a general skill base ensures that their agile setup has an on-going capability to deliver their sprints as expected. Prior to agile adoption, development individuals focused either on the user interface, database or coding and would stay with their preferred area only.

> ruck factor effect … people over time always have affinity to certain areas … if that's the only person knowledgeable in a specific area … it might actually get the entire development to a halt … always make sure that we have a back-up. (Engineering Manager)

> If someone leaves and they were working on their own without pairing it is going to be detriment … lost a whole heap of intellectual property. And that's the whole point of why we want discipline of pairing, rotating and small stories. (Principal Engineers)

4.4.3.2.1 Adapting to generalist skill sets

The decision for engineers to have general software engineering skill set was initially made by the engineering manager in January 2005. For this agile team to deliver implemented features in short development cycles it was critical that individual engineers were skilled enough to carry out any development task. To develop a general skill base, the engineering manager made the decision requiring sharing of all work at development level. Pair programming is one of their key practices for developing a general programming skill base since their stories require different sets of technical skills. In August 2006, pair programming was further adapted with a daily change of the individuals in the pairs to enhance the skill base of sub-teams. Through task sharing, engineers have also adapted their technical abilities with estimating, refactoring, and test driven development skills. The engineers feel that they now have sufficient technical skills in their team to implement innovative features.

> Probably 70% of the way there, everyone has got a basic idea of the different technologies and things that we need to alter [adapt] to get things done. (Software Engineer)

> Easier to work on any given [story], had to up skill substantially … not only coding, it is design, architecture, and testing … different functional areas, require [skills on] Java … database, user interface, performance, security. (Principal Engineer)

For achieving productive pair programming effort, the agile engineers are also required to adapt with excellent interpersonal skills, enabling spontaneous collaboration. This agile team's belief is that effective collaboration helps to contribute towards their team effort and success.

> Interpersonal skills … for pair programming it can be quite challenging … have to vary how you interact, depending on the different kind of personality. (Software Engineer)

Next, information is provided on the epistemological role of methodology with Akldevelopment's agile process.

4.4.4 Epistemological role of methodology

At Akldevelopment, the agile team has developed some key product development beliefs. They relate to their pair programming and post-mortem practices. Regardless of the number of sub-teams, they choose pair programming and conduct regular sprint

post-mortems. The epistemological role of Akldevelopment's agile method is presented using its sub-factors; the template for inexperienced developers, transfer of knowledge, and learning from past projects. First, adaptation information relating to transfer of knowledge is provided, as shown in Figure 28.

4.4.4.1 Adapting the practice for transfer of knowledge

Since January 2005, the pair programming practice is used by this agile team as a method for up skilling and training partners when pairing. In March 2005, this practice for up skilling individuals through pair programming was adapted with mentoring duties to provide local development experience and domain support for the new recruits. The need to pair with the same individual for a period of time was identified by a new recruit during a sprint review meeting early in their agile adoption. Engineers from outside Akldevelopment were also recruited to form this agile team rather than selecting all from the existing development teams. The agile team collectively agreed to pair a new recruit with same individual for a period of time until the individual became comfortable with the product domain.

Figure 28 Transfer of knowledge in relation to epistemological role of methodology - Akldevelopment

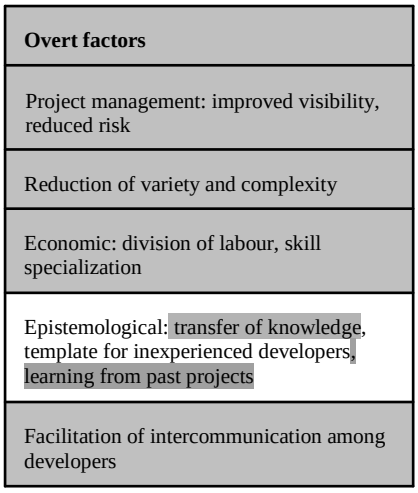
Overt factors
Project management: improved visibility, reduced risk
Reduction of variety and complexity
Economic: division of labour, skill specialization
Epistemological: transfer of knowledge, template for inexperienced developers, learning from past projects
Facilitation of intercommunication among developers

Through mentoring, the graduate engineers at Akldevelopment swiftly learn and gain sufficient knowledge, skills, technical know-how and experience of their development process and comfortably pair with others in their agile setup.

> We also have pair programming sessions … they have a mentor and stay with the same person for often as other people do. (Engineering Manager)

Next, adaptation information relating to the template for inexperience developers is provided (Figure 29).

Figure 29 Template for inexperienced developers in relation to epistemological role of methodology - Akldevelopment



4.4.4.2 Adapting the template for inexperienced developers

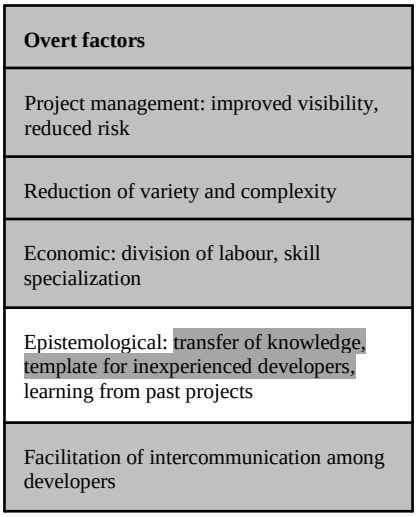
Since January 2005, all the new recruits for the agile team are provided with on the job training through pair programming as they are immediately assigned and included in all aspects of the work assigned to a sub-team. Here, a graduate engineer takes up to six months to become a competent engineer. They have adapted their agile approach with training sessions so that in the first few weeks, they get acquainted with their development environment. This learning practice was adapted with show and tell in June 2005 and also with sales based training programmes in October 2005 where new recruits are given tutorial sessions on the product to acquire domain knowledge. The sales based training program, normally used to train their customers on using the system, was suggested by their product manager. A major benefit of this training program is that the new recruits are instantly exposed to their product, helping them to acquire domain knowledge. The show and tell practice was proposed by senior engineers to present on something useful the engineers discover which they think the entire team should also learn, enhancing their engineering skills. The show and tell presentations are held after their sprint planning meeting on Fridays.

> Takes up to 6 months to get up to speed … have different techniques … introductory sessions within the first week, where there is 1 to 2 hour sessions to get up to speed with the environment … sales based training, look at very specific technologies … show and tell once in a while. (Engineering Manager)

> I don't think it was difficult to learn … especially when you have a group of people around you doing the same thing … you are feeling that you are involved. (Engineer)

Next, adaptation information relating to the learning from past projects is provided, as shown with Figure 30.

Figure 30 Learning from past projects in relation to epistemological role of methodology - Akldevelopment



4.4.4.3 Adapting learning practice from past projects

The sub-teams have a reflection meeting at the end of each sprint to learn about the issues encountered and to determine the drive forward for the next sprint, armed with lessons learned. In December 2006, this practice was adapted with an additional practice which involves monthly project reviews to identify the likely issues based on past sprint cycles and the necessary actions for moving forward. This new practice was proposed by the engineering manager and collectively agreed by the team in their sprint review meetings since the on-site customers (product analysts) did not work in the field to have appropriate understanding of current market needs. Adapting with this practice helps them to determine the emerging scope to take the necessary actions during their projects. Now, engineers can incorporate emerging and changing requirements to deliver high quality features in their current release cycle if they are not able to determine them at the start due to the lack of information early in the projects.

> Sprint review … talk about things that went well and things that went wrong with the previous sprint … whether or not things could be improved. (Engineer)

> A monthly review for projects … scope may change …because we learn more as we go, might have to do more scope, need less things or it might mean that we need different things ... so some additional effort might be required. (Engineering Manager)

Next, adaptation information is provided on facilitation of intercommunication among developers. It is identified as the final overt adaptation factor in Fitzgerald's adaptation framework, as shown in Figure 31.

4.4.5 Facilitation of intercommunication among developers

The agile team has an open workspace to facilitate effective interaction and communication between the engineers. They are co-located in an open workspace for achieving face-to-face interaction breaking down interaction barriers and encouraging

the engineers to collaborate with one another. Co-locating the product analysts in their open space also provides the engineers with easy access for on-going collaborations.

Figure 31 Facilitation of intercommunication among developers - Akldevelopment

Overt factors
Project management: improved visibility, reduced risk
Reduction of variety and complexity
Economic: skill specialization, division of labour
Epistemological: transfer of knowledge, template for inexperienced developers, learning from past projects
Facilitation of intercommunication among developers

Team work area [open space] for interaction and communication … discussions … white board sessions … design discussions … asking and answering questions … asking for help, … that everybody has an ideal and free access to any kind of information. (Engineering Manager)

4.4.5.1 Adapting workspace with clusters

Their open space is adapted with clusters, created using desks. This decision was collectively made by the team in their first sprint review meeting in January 2005. Senior engineers suggested forming clusters to have sub-teams co-located in separate areas of their open team workspace minimising noise interference but allowing easy access and cross sub-team interactions. As a result, engineers now have more cross-team collaboration and help since they can easily move around in their open team workspace which was restricted due to a single cluster where tables were joined together in the previous setup of their team workspace. The sub-teams choose a cluster for the day to work from as no one is assigned a permanent cluster.

> Ideally, have all people close together … physically is impossible … very inconvenient because we have the whiteboards … also the noise level … have people spread out … more space and the noise level decreases … easier for people to move around. (Engineering Manager)

> Work through something quite tricky, then the noise bothers me … can't think … that's not too often … being close to people we can just ask them for help … hear things … learn a lot that way too. (Engineer)

4.4.5.2 Adapting workspace with whiteboards and noticeboards

At Akldevelopment, whiteboards are regarded as an important tool for facilitating effective interaction and communication amongst the engineers and sub-teams. The decision to provide engineers with whiteboards in their open workspace was made by the engineering manager in January 2005. These whiteboards are now located on walls next to each cluster. Printable whiteboards are also provided in the meeting rooms. This agile team collectively made a decision to have discussions requiring long periods of time in meeting rooms away from the team work area, minimising the noise levels for

others. In January 2008, additional whiteboards were provided in their open workspace on request by engineers as the team grew in size. The engineers identify whiteboards as an important tool to collectively identify and communicate ideas, actions and messages to others in their workspace.

> Whiteboards are important … the whiteboard is typically used in meetings … for recording information … have a notice board … depending on what the current issue is … post trends … so that people could see whether we are improving or are falling back. (Engineering Manager)

> Very important tool … write down issues and the actions identified during sprint review meetings … post useful team notice to remain or inform [other] … engineers use it to communicate design ideas to others. (Principal Engineer)

4.4.5.3 Proximity of resource

Their meeting rooms are located in close proximity. This agile team has access to two meeting rooms next to their team work area and they also have other meeting rooms including a very large video conference room. The close proximity of meeting rooms is regarded as a contributing factor for on-going interactions and collaborations. Akldevelopment provides meeting rooms with all the necessary equipment and facilities, including phone conference capabilities. While these facilities are shared, they are available most of the time. However, for important meetings, the teams are encouraged to book in advance.

> People [to] use these facilities, make it convenient as possible … if they are easy to use, convenient to use then they will make use of it … to walk over to a different building, make less use of it. (Engineering Manager)

> From a manager's perspective it is important to provide a good environment. (Principal Engineer)

4.5 Covert factors

This section provides information on Akldevelopment's agile approach adaptation driven by the covert factors or political roles of software development methods based on Fitzgerald's adaptation framework (Figure 32). First, adaptation information on the comfort factor at Akldevelopment is provided.

4.5.1 Comfort factor

This agile team's development practices, documented in the earlier sections on the organisational and overt factors, enable them to handle challenging development situations such as dealing with tight schedules, meeting deadlines, identifying business

value stories, and meeting specific quality standards. Next, adaptation information is provided on another practice which helps to further provide a comfort factor for their engineers.

Figure 32 Comfort factor - Akldevelopment

4.5.1.1 Sustainable velocity: 40 hours per week

The agile team adapted their development approach to have maximum of 40 working hours per week. In January 2005, this decision was made by engineering manager and agreed upon by the agile team. While the team manager gave them some flexibility in the starting time in the mornings, the team collectively agreed to have daily scrum meetings at 9 a.m., which was compulsory for everyone to attend. Through this 40 working hours a week practice, they minimise individual burn-out and maintain a consistent development velocity. This practice helps to avoid their engineers working late hours and weekends to meet deadlines. It also helps them to avoid setting or accepting unreasonable sprint schedules and release deadlines. To help ensure a high productivity on projects, they also have adapted their pair programming practice to have a maximum of six hours for pair work per day only.

> Expected to be here between 8-4 p.m. … people are not expected to work in pairs all the 8 hours … more like 6 hours in pairs and 2 hours for other stuff, email, learning, or reading. (Engineering Manager)

> We come to work from 9 to 5 … work as professionals … a mass of different types of individuals, a lot of parents, all have their little social stuff. (Senior Engineer)

Next, information is provided on the legitimacy factor for their agile development (Figure 33).

4.5.2 Legitimacy factor

Figure 33 Legitimacy factor - Akldevelopment

At Akldevelopment, there is a mutual acceptance between the agile team and the product strategy group regarding the legitimacy of their agile approach for achieving successful feature development and releases. Their agile approach incorporates planning practices and sprints that have a significant influence on the product management function; the product strategy group is able to deliver business value features to the marketplace. Their one week sprint cycles provide implemented features to the product strategy group on a regular basis. The product managers now could request for a simulated version of a feature which they implement at a client site; this type of feedback was not possible with their previous development approach. Information on these aspects of their agile approach, including their adaptations, is provided in the earlier sections on organisational and overt Factors.

> Faster than how we were building before … the Product Analyst is writing tests … helps to clarify what I wanted and to make it simpler for the engineers to implement … very collaborative environment between the engineers and the product analysts … run software all the time, which is fantastic for me. (Senior Product Analyst)

Next, information is provided on the aura of professionalism with Akldevelopment's agile development (Figure 34).

4.5.3 Aura of professionalism

Figure 34 Aura of professionalism - Akldevelopment

An aura of professionalism is achieved through a wide range of engineer empowerment and trust. They have achieved a common ground between the team management and engineers for their development activities. The agile team is endorsed to make decisions on the adaptation of their agile process and tools. Such empowerment does not burden the engineers with practices and tools that they find to be counter-productive. Information on engineer empowerment (hiring responsibility), including how and

why it is adapted, is provided in the earlier section on the organisational overt factors.

> Increases efficiency … most decisions made in an extremely short timeframe … don't work on any thing that doesn't make sense … focus their effort on exactly what is required, but at the same time they take off the load from manager … a lot of things that a manager doesn't have to take care of anymore. (Engineering Manager)

Next, information is provided on the confidence factor with Akldevelopment's agile development (Figure 35).

4.5.4 Confidence factor

Figure 35 Confidence factor - Akldevelopment

At Akldevelopment, the agile approach provides an adaptive development process enabling the product strategy group to identify business value projects. Through their adaptive agile development practices they implement and release new features every three months, well ahead of their competitors. The agile team enabled Akldevelopment to reduce its new feature releases by almost 15 months. The executive management team has built up a significant level of confidence with the agile team's engineering ability since they continuously provide internal releases. This is achieved by building a cohesive and high performing agile team. Information on this is provided below, while how and why their agile teams are adapted is provided with the size of IS department factor.

> The executive team … extremely pleased about the process and what has been done so far … now they have seen what the outcome is … not only see how much money is burnt, but they also see what they get in return … seem to be extremely pleased (Engineering Manager).

4.5.4.1 Cohesive and high performing team

Akldevelopment delivers visionary and highly customized product features. Their agile teams quickly adapt to deliver such features, enabling the company to take up business opportunities in the marketplace. An important aspect of their product development is their continual support for feature enhancement after implementation at a client site. To be successful, they not only require having cohesive but also high performing teams to deliver quality products and provide timely support.

> The team is well functioning … has a range of expertise, the domain experts, technical experts, UI experts … people work together … on a daily basis, you just get to look at every possibility, than just thinking of one sort of thing. (Principal Engineer)

Akldevelopment ensure that they have a high performing team through encouragement and providing the means for participative leadership, shared responsibilities and a communicative environment where people interact and collaborate all the time. They also employ creative and talented individuals, those who were able to respond rapidly to deliver business needs.

> Have high performing teams … people want results … deliver those as efficiently as possible … always trying to improve the velocity… if low, not performing … the requirements aren't clear , aren't really communicating with the customers … problem with the technology, they're inefficient … always trying to improve, will increase velocity. (Principal Engineer)

Next, information is provided on the audit trail **(**Figure 36).

4.5.5 Audit trail

Figure 36 Audit trail - Akldevelopment

Akldevelopment no longer has a separate quality assurance team. The agile team engineers are trusted to follow test driven and refactoring practices to ensure that quality is achieved. The audit trail is embedded in tests, and enforced through their pair programming practice and the agile tester. When pairing, the navigator ensures that driver is implementing a unit test and refactoring code. Since adapting their agile setup to include the agile tester role, the agile tester helps pairs to identify ways to break their code. The pair implements these as unit tests. Adaptation relating to the agile tester role has been provided in the earlier section on size of IS department under the organisational factors.

> There are some techniques that are associated with pair programming as part of audit trail … like test driven development, refactoring, … when you work in pairs , always there's the partner … essentially pulls you back to do things. (Engineering Manager)

> The other pair that is with … so it should be working like a continuous code review … i expect them to point out if I have typed something wrong … expect them to challenge my decisions. (Principal Engineer)

Next, information is provided on the raise the profile of IS department factor **(**Figure 37).

4.5.6 Raise the profile of IS department

Figure 37 Raise the profile of IS department - Akldevelopment

The success of the agile team at Akldevelopment has raised the profile of their agile approach as a better approach for Akldevelopment. The agile approach enables the company to achieve shorter release cycles with improvements in the quality of their product. The development successes have led to the agile team doubling its size within two years of its creation. Adaptations relating to the practices for achieving quality and short development cycles are provided in the earlier section on organisational factors.

> Had major releases every 1 ½ or 2 years … now release a new version every 3 to 4 months … improved quality, able to do that over the last 2 years … makes it easier if we need more people, perceived as a team delivering good results … we always have a better position. (Engineering Manager)

This change was driven by the company executives, and a decision was to be made on agile methods' suitability as a common development process for the entire company. Since its creation, the agile team's performance is regularly measured against the other autonomous teams at Akldevelopment.

> Within there is a competition … lots of different kinds of teams … waterfall teams, kind of agile teams, and there's us [agile team] … mantra that we are providing with agile is that we are better for the customers … better for the company … more efficient, we enjoy it more as well. (Principal Engineer)

The successful adoption of the agile approach was driven by the engineering manager, who was initially hired for the purpose. He was empowered by the company executives to ensure a successful agile approach adoption. Some of the key actions that he took for his agile team were as follows: hiring a highly agile experienced engineer, providing consultants, delegating decision making, co-locating product manager and re-allocating engineers to other teams who did not fit with their agile culture.

> Don’t have executive support don’t even try … there was nothing I couldn’t do … whatever I deemed to be necessary I certainly would talk to the engineering managers … whatever we agreed upon , is what we did … there was all the support that was required to make this happen. (Engineering manager)
>
> For management … their level of relevance is different … they want timelines and project plans … from a planning and tracking perspective, the approach does suit senior executives. (Principal Engineer)

4.6 Summary

This chapter provided adaptation information of Akldevelopment’s agile approach for software development. Their agile approach practices and related adaptations have been captured, analysed and presented using Fitzgerald’s adaptation framework; original formalised methodology vs. Methodology-in-action; profile of the development environment, covert factors and overt factors. As such, it shows that these four major adaptation factors with their sub-factors drive the adaptation of the agile approach at Akldevelopment. Hence, the chosen framework appears relevant. However, analysis of Akldevelopment’s adaptation data suggests modification to this framework and its major components to account for the particular adapted processes within agile teams. The proposed changes are suggested in Chapter 6.

Chapter 5: Meldevelopment case study

This chapter presents the Meldevelopment case study. First, the company overview is provided, followed by discussion of Meldevelopment's agile method adaptation. The data used to construct the case study were collected through a series of interviews held at Meldevelopment's development centre in Melbourne. Other sources of information include the company website and online industry publications about Meldevelopment and its software products.

5.1 Overview

This section provides information on Meldevelopment's business background, its past product development environment and its agile adoption.

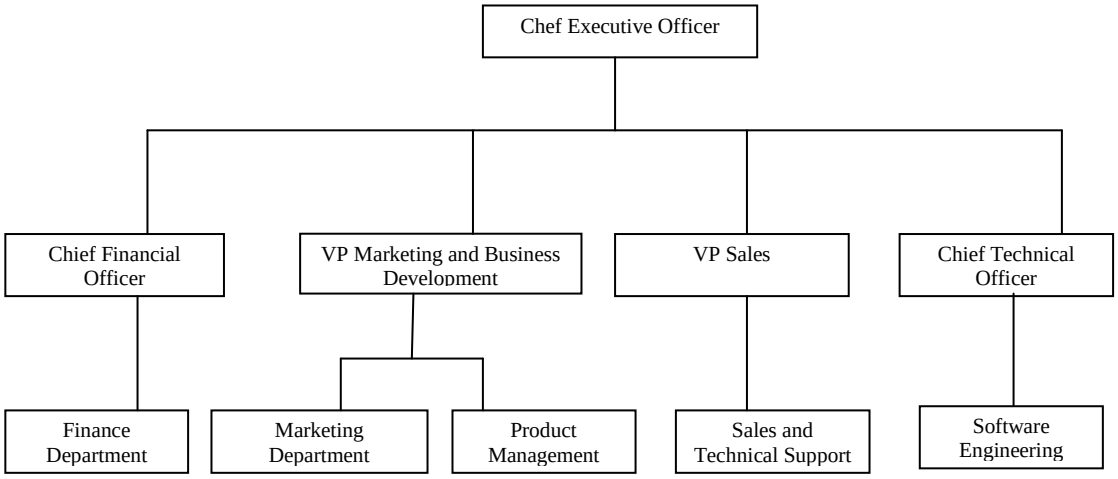
5.1.1 Company background

Meldevelopment is one of the fastest growing international software vendors in the world. The company is recognized in the software industry worldwide as a market leader and a provider of highly intelligent asset management software products. Founded in 1990, it has its head office in Boston and has offices across North America, Europe and Australia. Meldevelopment has a consistent annual growth rate of over one hundred percent (at the time of this study) and is aggressively expanding into new international markets.

Besides a software engineering department, the company has sales, marketing and business development, finance, and human resources departments. The company's international executive team is made up of a Chief Executive Officer, a Chief Financial Officer (leader of finance department), a Vice President of Marketing and Business Development (leader of marketing and product management), a Vice President of Sales (leader of sales and support), and a Chief Technical Officer (leader of software engineering department) (Figure 38).

The software engineering department is based at the Melbourne office. This department is headed by their Chief Technical Officer and one of the co-founders of the company. The Melbourne office has 60 staff members, most of whom are software engineers. The others are product, sales and marketing, finance or human resources staff.

Figure 38 the executive management structure of Meldevelopment



Meldevelopment's software products enable large and medium sized organisations and corporations to set up automated asset management and compliance systems. Meldevelopment has seven major product lines as their mainstream business. Their focus is on growing the potential of their products and on enhancing their quality and user experiences. Meldevelopment products are expected to grow for a number of years before reaching a maturity stage.

InfoWeek Online declared Meldevelopment a winner and awarded it a gold star rating in every product category in their software management review. Microsoft endorsed Meldevelopment as a certified partner, recognizing them as a market leader for their quality products and high-levels of customer service.

Meldevelopment was also named a winner in the Consensus Software Awards, Australia's most prestigious business-software competition. In addition, Meldevelopment was one of two companies to also earn the exclusive "Highly Commended" designation for their software products. The company won another prestigious Australian award, the Information Industry Association (AIIA) award, which recognised their innovation and excellence in providing visionary IT technologies. Meldevelopment was also acknowledged in the Gartner Visionary Quadrant Report for their reputation for providing complete software solutions and as a responsive vendor.

Meldevelopment works closely with industry-leading firms to ensure that they deliver the best possible software products. In their list of global partnership networks are some of the world's major hardware and software companies like Microsoft, IBM, and Siemens. Meldevelopment also has established partnerships with several other leading companies from all over the world.

Meldevelopment's clients are some of the world's largest corporations in different sectors having up to tens of thousands of IT users. Amongst their clients are: a dairy company with a $1 billion annual turnover; one of the largest oil and gas companies in the world, with 100,000 employees worldwide; a brewing company trading in 180 markets around the world; a research institute with 12,000 employees; a global ERP provider with 4000 users in 90 countries; a magazine company with 1,000 devices for a range of users; a major USA retailer with 230,000 employees worldwide; and a lottery company with 900 retail outlets and 270 gambling hotels and clubs.

5.1.2 Product development

The individuals who founded Meldevelopment discovered that there was a huge potential for a particular software product that they had expertise in. The engineering department was set up in Melbourne in 1990 with a small team of 3 developers.

> A small talented team …always hired very elite guys … our strength is our technical skills and ability to solve anything ... with a lot less skilled staff base, it would have been very hard to get a positive result. (Director Software Engineering)

Over time as the business grew, Meldevelopment boosted their development capacity. Additional software engineering personnel were recruited and the team rapidly grew in size. While exceptional calibre software development talent was sought from within the industry, Meldevelopment had also put a strong focus on recruiting the most talented IT graduates from universities around the Melbourne region.

> Always had a strong graduate program … hired as a graduate straight out of university … had a time when 80 or 90 percent of our engineers had come in like that. (Director Software Engineering)

Next, some of the key features of Meldevelopment's past product development environment are described.

5.1.2.1 Multi-skilled team

A small start up team resulted in developers doing all the development tasks. Hence, their job title was 'software engineer'. They were made responsible for the entire development process; analyzing, designing, coding, testing and writing documentation for the Meldevelopment products. From the onset the engineering department fostered multi-skilled developers. Meldevelopment developers were trained to be involved with the entire development process so that they could better understand the products they were building, why and who they were building for, and why someone would be interested in their products.

> Software engineers … able to analyse requirements, completely understand new features, analyse from a one paragraph description … negotiate, design, implement it, write test designs and test it … good at the entire life cycle. (Project Manager)

Meldevelopment aimed for collective developer accountability for their software products: the developers were given the ultimate responsibility for the delivery and ownership of their products. Amongst the necessary developer skills at Meldevelopment was domain knowledge. Meldevelopment developers were required to have an understanding of their product's implications on client businesses rather than just be able to come up with a technical solution.

> Understanding the domain … who is going to buy the product… how they are going to use it … put themselves in the shoes of users … the best technical answer is actually not the best business answer. (Director Software Engineering)

Some key business roles also emerged within Meldevelopment such as sales, support, marketing, and product management. Except for the marketing role, these roles required a fair level of technical expertise and in-depth knowledge of their software products. The business roles also identified new features and became the main source for product specifications since they were the ones working in the field with customers and clients.

> Our product is fairly technical … sales process is very technical, know enough to appreciate why certain technical decisions are made … why it would be a compelling product … product and sales engineers know technical directions of the market. (Director Software Engineering)

5.1.2.2 Small development teams

Realizing that the best development effort was achieved through a small number of software engineers working together, the engineering team was split into separate development teams as the number of development personnel grew.

> Team had up to 11 or 12 people … we don't tend to go above six or seven … much relies on internal communication with people … get above that it's hard to hear what is going on … we reduced team size. (Senior Engineer)

5.1.2.3 Quality assurance team

During the mid-1990s a dedicated quality assurance team was established for testing to ensure extremely high quality and completeness for their products. Meldevelopment had a list of items against which to test the quality of their product. Primarily, this was to ensure that thorough manual tests were done to certify the functionality, usability, reliability, performance, scalability, internationalization, security, and supportability of their products. The quality assurance team made certain that Meldevelopment's software products were compatible and integrated smoothly with a wide range of hardware and software platforms. This team raised the quality standards of their products enabling Meldevelopment to launch its product in international markets to meet specific client needs.

> Our products, very complicated system that needs to deal with 10 other systems in an organisation … dissatisfaction that occurs in our customers is trying to get everything configured in a certain way, it is complicated for them … not a straight functional defect … around the roundness and the completeness of a product. (Director Software Engineering)

The testing task was shifted from the software engineers. The quality assurance team became responsible for testing to certify features including the overall usability of their products. They tested for defects and provided a list of bugs and issues back to the engineering teams for fixing and improvements.

> Worked on a model where the software engineer would run and write test designs … it was just crazy. Run and fix all the bugs that came through. (Team Leader)

5.1.2.4 Documentation team

Similarly, a separate documentation team was established for the production of manuals and online support for their software products. For Meldevelopment, it was not practical to dedicate a technical writer to each team since the number of development teams was increasing.

> We had technical writers within the teams … used to be more closely associated with a particular team. (Team Leader)

Meldevelopment considered documentation an integral part of their software product that enhanced overall quality. They held the view that customer perspectives on quality were about the usability of software products. Having a separate documentation team

provided another viewpoint on the quality aspects of their software products. Meldevelopment was able to identify user problems and address specific user needs. Later, with agile adoption, the documentation team became the usability experts. They were the main influence for adapting their agile process with a User Centred Design (UCD) approach. They took the responsibility to ensure that the engineers developed usability skills.

> Our documentation team are like technical end-users. (Director Software Engineering)
>
> People typically need some documentation … to find out how something works. It should reduce support costs and find answers quickly … should improve the overall user experience. (Technical Communications Engineer)

5.1.2.5 Development challenges

Meldevelopment's engineering teams always had major development challenges to overcome. One of these was working with available resources to meet business demands.

> Resource head count … try and get the right people in the right areas to ensure that right communication was happening … taking resources away, have an impact on the project. (Director Software Engineering)

Task estimation was another constant challenge for the Meldevelopment development teams, which had an impact on their delivery schedules and deadlines.

> Estimation was a constant challenge … challenges with schedules … a number of projects where we had overrun the date. (Director Software Engineering)

At times, there were communication issues between Meldevelopment's functional teams within the engineering department impacting effective cooperation between them.

> The interaction [with] QA … you could picture it as a brick wall, and at the end of the project you would throw it over and they would throw it back to you. It wasn't very good. (Project Manager)
>
> Communication problems with engineering teams … specs and then as things get implemented there was variation … don't necessarily realise until you get the product … end up having to dig a lot of information out. (Technical Communications Engineer)

5.1.2.6 Method adoption

The engineering team had always adopted an appropriate method which best suited their development work. The method was important since it made their development work more visible and transparent, helped to determine the resource requirements at various points during their projects, and enabled them to meet their quality standards and

productivity requirements. However, it required a lot of upfront work from the team before they could start any implementation.

> Methodology has reporting aspects … give you visibility … give a heart beat to a project … ability to look at it at certain points … need more resources, need to tune the whole project, assess where you are … [had] a number of methodologies … some very rigid, slowed us right down … had project plans, lengthy requirements documents … have a review cycle around that … lots of feedback, tracking all the changes of everyone's comments. Keep cycling round quite a bit before we got round to building … get everything completely designed. (Director Software Engineering)

The engineering department had adopted different life-cycle based approaches in the early 1990s before taking up the Rational Unified Process (RUP) model prior to agile adoption in 2003. The adoption decision was entirely left to the engineering department, where senior engineers played a significant role in choosing a method. From the company's perspective, they had to deliver against an agreed upon set of features regardless of the method used. From their inception, the development teams were judged based on their delivery and the quality standards of their products.

> Used the RUP model … changed the names for the people in sales to understand … called inception phase requirements, elaboration phase design, had the construction phase, and transition phase completion. (Project Manager).

The Meldevelopment engineering team had adopted the approach to streamline their RUP development method to make it appropriate and relevant to their development projects. Meldevelopment had adapted some common rules associated with the RUP approach to ensure that product requirements were accurate and met the market needs.

> Rules [were] around the requirement specs … a list of people that had to review … get written feedback from each person, incorporated their comments … colour coded to the person … had to address each of the feedback points … very thorough [but] time consuming. (Director Software Engineering)

5.1.2.7 Development problems (RUP method)

While the Meldevelopment engineering team felt that they were reasonably good at product delivery, far too much time was spent sorting out what was to be built. Despite having a thorough process, they felt that some of their projects took more resources and a longer time to implement than they should have due to the issues described next.

5.1.2.7.1 Ineffective stakeholder partnership

The process to capture the requirements of their products was a bottleneck. This process was not that appropriate and the responses from the stakeholders were not as immediate as they would have liked. The senior engineers felt frustrated since they were spending a lot of time in meetings to finalise specifications instead of implementing features.

> Input based on a document … end up making superficial comments … can't grasp the whole picture in their head and give good feedback … the stakeholders don't have the time looking at the stuff … some requirements came afterwards. (Director Software Engineering)

5.1.2.7.2 Non-emergent architecture

The engineering team also felt that there was a lack of innovation. The principal engineer was spending a considerable amount of time designing the architecture upfront without any collaboration and feedback from the other engineers.

> Internal design got overly complicated … end to end prototypes of the architecture would have seen the pros and cons … something that was a lot more maintainable. (Director Software Engineering)

5.1.2.7.3 Unfeasible test coverage

While the engineering team would put in a huge effort to ensure bug free releases, at times (as their process required) writing a large number of test cases to cover all the scenarios were just not practical. A common sense approach was required for setting their test coverage goals.

> Test design … mapped out every equivalence class and every combination, had this massive test procedure … if we tested everything it would've taken days to run … was thinking, does this really feel right. (Director Software Engineering)

5.1.2.7.4 **Unsustainable pace for development**

The backend of Meldevelopment projects was very demanding on the entire engineering department. The engineering teams frequently spent long hours and weekends to get products released, causing burn-outs and impacting their motivation and productivity.

> Some of the days got into really tough situations it was really stressful … it takes its toll and you can get burnt out. (Director Software Engineering)

5.1.3 Culture for improvement

As a highly innovative company, the Meldevelopment engineering team employs a "learn-by-doing approach" and a culture of trying to improve. Through these practices, the engineering teams realised that their RUP process did not match their development conditions. They strongly felt that it required further improvements.

> Had a lot of people saying this doesn't feel right … engineers didn't had the same passion in their work … lots of ideas about how we can cut down … was a hard period … some people would feel that we were saying let's cut corners, not follow the process. (Director Software Engineering)

5.1.4 Move to agile approach

The engineering team felt that a better approach for requirements capture was to build prototypes and then let the various stakeholders provide feedback as opposed to reading large documents with lots of details. At Meldevelopment, the stakeholders have the concepts of the products in their mind, allowing for quick responses to queries and better feedback on how the products will be used in real situations. They needed to adapt their design phase to include stakeholder participation to quickly identify features for an immediate start to implementation. The engineers thought that this would also enable them to build and test features bit by bit and create a feel good factor that progress is being made from the start of the projects. Meldevelopment had already adopted version control systems (VCS), for automated management of their source code repository. An automated Test Driven Development (TDD) framework with a continuous integration approach was adopted with the VCS to build and test features incrementally. With adoption of an automated TDD approach, the functional (unit) tests are done by engineers during code implementation, and the systems (acceptance) test is done by the quality assurance team. The engineers quickly realised that they as a product company required effective cross-functional collaboration not only for development projects but also for product (business) planning. This would bring the needed interaction and collaboration between the business and engineering functions. In 2002, the Meldevelopment engineering team had discovered the agile approach for software development, which gave insights to development similar to those that they were considering would be a better approach for them.

> Came across the field of agile development … had Alistair's [Cockburn] book … talks about the cooperative game… different models of communication … enlightening for us … massive document is not ideal… losing out on face-to-face communication … start attending conferences, reading books, internal training sessions … getting everyone thinking about it, coming up with ideas … did project applying some of the techniques … learning and adapting. (Director Software Engineering)

In 2003, the agile approach was fully incorporated into their development process. At that stage, the engineering department at the Melbourne office had four types of cross-

functional engineering teams. These were three development teams, a quality assurance team, a documentation team and a technical support team. This structural setup remained the same with agile adoption.

5.2 Methodology Adaptation

This section provides comprehensive findings about agile method adaptation at Meldevelopment based on the components identified in Fitzgerald's adaptation framework: original formalised methodology vs. Methodology-in-action, profile of the development environment, overt factors and covert factors. Findings on each of these framework components on Meldevelopment are provided separately and in the above order.

5.2.1 Original formalised methodology vs. Methodology-in-action

This section presents findings in relation to original formalised methodology vs. Methodology-in-action at Meldevelopment. The findings in relation to this component are presented according to the themes that emerged from analysis of data collected from Meldevelopment; method-in-action, rationale for adaptation, adaptation process and adaptation responsibility.

5.2.1.1 Method-in-action

The Meldevelopment engineering team had a hybrid agile method adapted from several agile and structured methods. However, none of the methods are used by them in their entire form. The selected practices for their hybrid approach were based upon agile values and principles for software development. The agile values and principles place a very strong emphasis on collaboration which they require with their development practices. They have incorporated their hybrid method with immediate communication and interaction with their key stakeholders for projects. Their product success is dependent upon this type of development collaboration with their sales and marketing department including with their clients. They derived a hybrid agile approach by selecting relevant practices from the agile methods DSDM, Scrum, Feature Driven Development, and XP; the Rational Unified Process (RUP); and the usability engineering method, User Centred Design (UCD). At Meldevelopment they adapt their method according to development situations.

> The talent and commitment of the guys … for every situation they do what's necessary to make a thing successful … the principles of agile development are consistent with every project we do … communication, they're fundamentals… the stuff in scrum a lot of things there are similar to

> the way we work, … the feature-driven development works … prototype [DSDM] tackling feature after feature … XP, unit testing. (Director Software Engineering)

5.2.1.2 Rationale for adaptation

As Meldevelopment grew with its software products and market share, new development challenges emerged that impacted the quality and delivery schedules. They had to adopt new methods and tailor their approach to counter the issues. Their engineering team has learnt to adapt their method as they are aware that they cannot apply a particular end-to-end method for every project.

> Trusted to do the best we can with whatever techniques we want to use … judged on that (Director Software Engineering)

> Evolved over time as to what has been working for us and what hasn't … tuned things as we have gone along … have picked up different techniques … have learnt from other methodologies … never implement a complete top to down methodology … just tailored things. (Team Leader)

At Meldevelopment, the development projects are closely aligned with the company's business goals and objectives. Here, the engineering team ensures that these goals and objectives are achieved through successful project implementations. The business goals and objectives play a strong role in the vision of their products and the reasons why the products are developed. These reasons constantly change according to market demands and what the company wants to achieve. Therefore, they cannot depend on strictly applying their hybrid agile method to all the development projects. They have to adapt their method to achieve their business goals and objectives since they know that the practices that worked well previously may require tailoring.

> Have different requirements … focused on the scalability or correctness … different risk profiles, need to change the way you work … had goals of being the first on the market … strategy was focused on quick delivery … whereas [with some] products … establish more maturity, the focus is on quality and upgrade to a large customer. (Director Software Engineering)

> Engineer needs to adapt to the business … the business and market requirements change … within a 6 month release cycle, the market moves … features that they have started working on may no longer be needed … something else is more important … unable to adapt you will not cope in a software development house. (Product Manager)

5.2.1.3 Adaptation process

At Meldevelopment, adaptation is regarded as taking suitable practices and standards of a published development method, modifying them for their environment and changing them to suit the needs for their projects. Here, agile adaptation is low ceremony, having an informal process and done on an as needs basis during projects. Some development

teams adapt to have two week iteration reflection meetings due to the complexity of their projects. At times, the project manager or team leader may call up a short meeting to discuss an issue raised by an engineer during an iteration. However, Meldevelopment has a major reflection meeting at the end of each project which is compulsory for all team members, quality assurance engineers, technical writers, project manager and product manager to attend. This reflection meeting is in two parts where the first meeting identifies all the issues faced during the project. A second, follow up meeting is called after a few days to collectively decide the actions for going forward based on the issues identified with the previous project. The few days gap taken between the meetings is for individuals to think of likely actions for the team to take. At Meldevelopment, method adaptation is the responsibility of all individuals who are involved with software development.

> Don't have a formal process … [don't] record adaptation … think of it on a case by case basis … look at what's not working, what result is not as optimal and how can we actually improve … come from people doing the stuff day to day. (Director Software Engineering)

> Processes we use always change … emphasis on don't get bogged down in the documentation … team gets together to discuss things … reviews on how things are going, reflection meetings … has even been a few just five minute meeting, where the manager will say "someone has raised this issue". (Senior Engineer)

5.2.1.4 Adaptation responsibility

Here, the engineering teams are responsible for adapting their agile approach. The decision to adapt is delegated to the development teams. Through extensive development experience with their products, the engineers have developed strong views on method adaptation. They feel that it is much easier and more valuable to adopt individual practices and to tune them to what they need to do on projects.

> Adopt the idea of the process … don't get hung up on all the bits … capture the essence of what they mean to do … we do that. (Project Manager)

> Easier to start applying practices where you've got problems than to just take a complete methodology … flexibility of able to take bits and pieces and to adapt is a real advantage. (Director Software Engineering)

The adoption decision for a practice, technique or a tool is made by the engineering team based on trying it out with their agile environment. Even though the project manager has the final authority on adoption, he generally implements what the team agrees upon.

> Very open to trying something out … if it doesn't work it goes … it has to work before you adopt it … happy to give it a go for a couple of weeks. (Project Manager)

5.3 Profile of the development environment

This section provides adaptation information on Meldevelopment's product development environment (Profile of development environment) based on Fitzgerald's adaptation framework. Figure 39 provides its adaptation factors as identified in the framework. First, information on adaptation of Meldevelopment's in-house development is provided as shown in Figure 39.

Figure 39 Profile of development environment-Meldevelopment

Profile of development environment
In-house development
Size of IS department
Project duration
Legacy systems development
Responsible autonomy
Productivity-rigor trade-off

5.3.1 In-house development

Meldevelopment's in-house development is a planned option. The adoption of an agile approach has significantly contributed to the growth of their software products. Meldevelopment's in-house agile development capability is built to meet their business needs. Their projects are mostly short term, one month to six months in duration and they regularly meet their target of four to six major releases annually. Their agile approach allows the flexibility to select or add features to be implemented without being limited to what is initially planned.

> Realised that something needed to change … whether the market moved or needed something [different] … have to do it … just work hard to do that type of thing … change requests, have to take those on board. (Director Software Engineering)

> You're making a guess at what people want … sort of learning and evolving that as you go through … everyone has a different input and it's a real sort of balancing act. (Senior Engineer)

Meldevelopment has once outsourced some of its development work. Using this external resource to leverage development efforts did not limit Meldevelopment's own innovative capabilities. The company only relying on suppliers and contractors would have limited the growth of their technical capabilities and knowledge of their own products. The complexity and technical nature of Meldevelopment products requires engineers to have past development experience with their products. In-house development also plays a significant role in their product innovation. The following information provided by the product manager indicates why in-house development is the preferred option for Meldevelopment.

> Time to market is critical for us … come to market late … impacts the entire company … put marketing plans in place … beta programs have been running prior to the release … have sales opportunities lined up … new product lines … look at how we can leverage some of the technology … to take leadership within a different market … brought to market a product, now bringing that up to another level … features are driven from sales opportunities, have customers requesting features … that is one of the big challenges, prioritising those value add-ons on top of the existing products. (Product Manager)

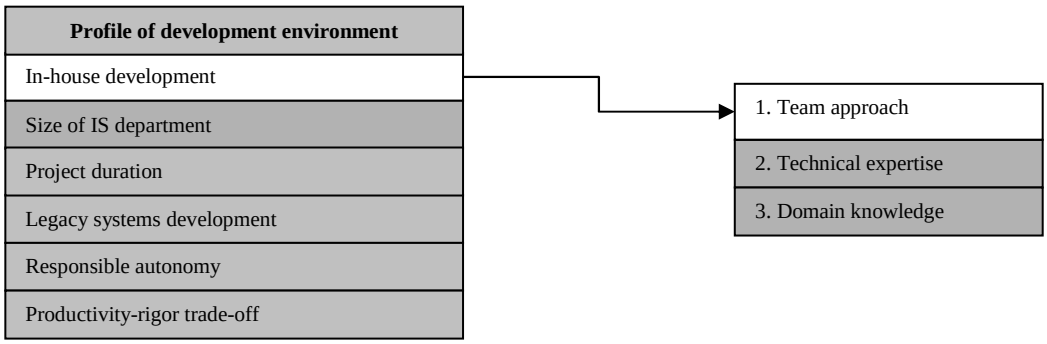
Through a solid in-house development, Meldevelopment has never been in danger of exposing its product technical knowledge outside the company. Meldevelopment with its agile approach has its business and development functions working together daily and has an effective control over its in-house development effort providing the stakeholders with first-hand information on project status.

Meldevelopment has focused on a team approach, technical expertise, and domain knowledge to build highly skilled and innovative in-house development capabilities. These three sub-factors associated with the in-house development function emerged from the analysis of data collected from Meldevelopment. First, information on the team approach is provided. Figure 40 illustrates the adaptation factors associated with in-house development.

5.3.1.1 Team approach

Meldevelopment's products are built with a lot of imagination, rather than by direct inputs from their user organisations. The engineering department has wide consultations to get plenty of insights before decisions are made on the likely product features. Meldevelopment's agile product development environment has a cross-functional effort.

Figure 40 In-house development – Meldevelopment



Meldevelopment's field staff (who work closely with the customers) are consulted for feedback and ideas on product requirements. They are mostly consultants and sales and

marketing personnel. Conversely, the sales and marketing department works closely with the engineering department so that they know when certain features will be available. This helps with starting marketing campaigns or sales processes with prospective customers. For this reason, a team approach for their in-house development is important. The team approach is particularly important for the growth of their products. Their sales and marketing department has vital product information since they frequently liaise with the customers, which is critical for engineers to decide exactly what is to be implemented.

> Things are constantly evolving … we don't necessarily know, what the customer wants … it could all change. (Senior Engineer)

> Developing some new things that no one else has yet … don't have someone giving us the definitive answer … there's a range of people, sales and marketing and consultants. (Director Software Engineering)

To achieve a cross-functional approach, the management promotes team spirit, and encourages and provides a fun work environment. Meldevelopment provides an environment where individuals can feel part of the team making them non-hesitant and freely contribute, help others, and seek assistance. The management has also bought a corporate gym membership, provides a darts area and a sofa in the kitchen, hosts lunches and barbecues for major events, buys movie tickets and DVDs, hosts off-site events, and runs Friday morning teas and one hour individual presentations. These team building activities are considered to be critical by Meldevelopment to promote cross-functional collaboration to have workforce agility for product development.

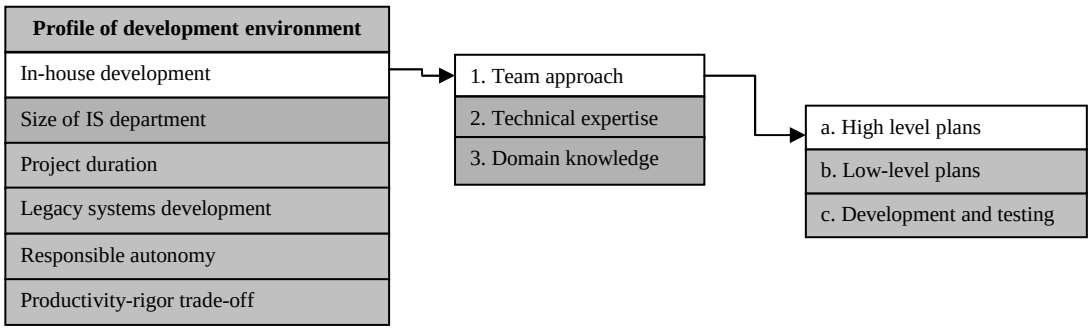
> Swap and meet … get insight into what someone else is working on, establishes a deep relationship and builds a stronger team … show and tell, a forum for communicating … sofa in the kitchen, to promote communication (Director Software Engineering)
>
> Spend a lot of time with team, trust and rely on each other … learn more about how to do things better … you share knowledge a lot more. (Software Engineer)

Meldevelopment's in-house agile product development has a three phase development cycle; high-level planning, low-level planning, and development and testing. Next, adaptation information on each of these phases to achieve a better team approach is provided.

5.3.1.1.1 High-level plan

This section provides information on Meldevelopment's high-level planning task and its adaptation to achieve an efficient team approach. Figure 41 illustrates the three phases of Meldevelopment's agile product development cycle for its in-house development.

Figure 41 Team approach in-relation to high-level planning – Meldevelopment



Meldevelopment with its agile development now has a cross-functional approach to establish product plans which also involves their senior management. At Meldevelopment, their high-level agile product planning process involves creating the product vision and roadmap plans. These two artefacts show their initiative to produce a strategic product identifying a prioritised list of highly innovative features including a set of short and long-term release goals of these features. However, priority of features and release plans at Meldevelopment may change due to the changes happening at the marketplace.

> Have positive agile culture through the whole company … roadmap is on what dates you would deliver things … the vision is what the roadmap is trying to achieve with new features in the market … have a roadmap for next 12 to 18 months of projects. (Director Software Engineering)
>
> Always have changes … people accept it … requirements change, the needs of the business change … more challenging than it sounds when the business comes back and says we really need this on this date … its not always realistic. (Team Leader)

At Meldevelopment, product planning is a cross-departmental function requiring everyone to contribute ideas and provide views for a proposed product. With their agile approach, it is the responsibility of their product manager to communicate, coordinate and compile the vision plan. This plan is then presented to the product planning team consisting of senior executives and managers of various departments of the company including the engineering and product managers for further discussions and approval for implementation. Based on the vision plan, the product planning team collectively formulates a roadmap plan for market releases for a period up to eighteen months. The

individual engineers at Meldevelopment are not directly involved in the final decision making during product planning, but have an engineering manager representing them.

> The product manager works with engineering, sales and marketing [to] propose the plan … vision plan, get creative spark from people with insight into how it will be the best product … roadmap, you can list features, how big they are, the release they are in. (Director Software Engineering)
>
> This aspect of our work is not agile … non-collaborative thing ... developers aren't really involved. (Senior Engineer)

This team approach model consisting of senior management (representing different functional teams) for product planning has been consistently applied at Meldevelopment since adopting the agile approach in January, 2003. Ultimately, the CEO of the company has the final say, but he is part of the product planning process rather than just at the end to make the decision to accept, reject or request a change in a plan. With the adoption of an agile approach, their product planning now involves collaborative decision making. Once an innovation is approved for development, the high-level and roadmap plans are then given to the engineering team through their product manager for its implementation.

> Spend two or three days doing product planning, … looking at the market and the roadmap … bouncing some things around, feel what's realistic, what we can do as a complete team … have consensus and agreement. (Director Software Engineering)
>
> Get some very high-level requirements from the product planning committee … the project manager, documentation and QA … all sit on meetings just to get the product vision together … get a rough sort of skeleton … get a rough agreement on what the system is going to look like … whole team is involved with that. (Senior Engineer)

5.3.1.1.1.1 Adapting planning artefact

In June 2003, Meldevelopment adapted the format in which the roadmap plan is captured to effectively communicate the product release targets within the organisation. The senior engineers realised that format of the roadmap plan did not match the feature driven development which they are undertaking. The engineers collectively agreed that this artefact required additional columns for them to track and reflect use case analysis, user centred design, and unit test coverage for each high-level requirement. The tracking and reflecting activities are critical since with their agile development engineers are also assigned the responsibility to ensure the quality of features. During a project review meeting, the engineers collectively agreed with the product manager on a new format for the roadmap artefact to effectively identify, communicate and reflect each high-level

requirement. A major benefit through this adaptation has been that the development teams at Meldevelopment now easily manage the risks and scope associated with each high-level requirement.

> The roadmap … it has adapted into a format to get good communication between the product manager and us. (Direct Software Engineering)

5.3.1.1.1.2 Adapting product manager responsibility

At Meldevelopment, there was a significant change in the requirement elicitation process with their agile approach adoption in January 2003. This change ensures an effective team approach that involves the engineering and the sales and marketing departments to establish high-level requirements. This model is based upon face-to-face interaction, collaboration and working together as a group. In March 2003, the product manager (part of sales and marketing group) adapted with the responsibility to elicit and set priorities to high-level requirements through a series of meetings with various stakeholders. This role adaptation was suggested by the senior engineers and agreed upon by the sales and marketing group. With their previous approach, senior engineers had faced serious challenges to elicit low-level requirements often causing delays in their implementation. With this adaptation, the product manager now is also part of their design phase helping engineers to swiftly elicit low-level requirements and providing engineers and project managers with further information to help them understand high-level requirements.

> Product manager, responsible for defining the requirements … works with the marketing & sales, consultants, documentation , quality assurance and support team … sees customer issues day in day out … does all the trade-offs and priorities. (Director Software Engineering)
>
> As an engineer I have access to the product manager directly… so my communication can be straight to the product manager. (Engineer)

5.3.1.1.1.3 Adapting product manager role

In June 2003, the product manager role was adapted to have someone with a reliable technical background. Previously individuals with a "traditional" product management background were appointed. Prior to agile adoption, the senior engineers at Meldevelopment had realised that they required a product manager with technical understanding to determine what can be implemented from a large number of ideas and concepts put forward from various stakeholders. This suggestion was made by the senior engineers and agreed with the sales and marketing department on adoption of an agile approach. Previously, engineers experienced requests coming from product

management which technically were not possible or extremely costly to implement. The product manager role now requires sound technical expertise to effectively communicate the proposed features with user organisations, understand specific and market needs, and to understand and communicate the limitations of what can be achieved. With a technical background, the product manager quickly and effectively identifies the most appropriate high-level requirements. This product manager has also facilitated effective collaboration with development teams during the design phase. Adapting their product manager role has now made their team approach more effective and efficient for requirement elicitation.

> Our product is technical … had product managers just looked at externals … current product manager drives a lot of our projects … seeing what the market wants … puts together [what] we would be able to do … does culling. (Director Software Engineering)
>
> Has to have a technical understanding …wouldn't need to have technical skills down to a code level … primary role is to almost simulate the user, looking at the system … the technical knowledge should be more of the customer domain. (Senior Engineer)

5.3.1.1.1.4 Adapting with customer input

In November 2003, Meldevelopment adapted their technique for high-level requirements elicitation with customer input. The decision to incorporate potential customer input was made collectively by the product manager, project managers and senior engineers due to the fuzziness of proposed features when providing high-level estimates. On occasions, engineers found it difficult to provide high-level estimates for features due to lack of sufficient understanding of them. During a project review meeting, the engineering unit collectively agreed to develop prototypes to get feedback from clients for engineers to develop understanding to provide high-level estimates for complex requirements. Using prototypes to determine high-level requirements also includes checking if a feature is worth implementing due to a high estimated cost. Adapting the practice to identify high-level requirements by involving customer input and feedback helps Meldevelopment to swiftly understand and determine the business value of a proposed feature. Here, customer involvement has become more common now, making the team approach more effective for requirement elicitation.

> Had a few customers … is a positive thing …we collaborate well … we understand the requirements … and they get a good experience out of it. (Director Software Engineering)
>
> Have a list of features prioritised … customer based stuff fits in best … just constantly talking to the customer … showing them lots of stuff along the way. (Team Leader)

5.3.1.1.2 Low-level plan

This section provides information on Meldevelopment's low-level planning task and its adaptation to achieve an efficient team approach.

The design phase is used to analyse the high-level requirements to derive low-level requirements, creating a product backlog for implementation. The project manager drives the design phase and is responsible for the backlog and its implementation. This design phase is one of the most important phases for the Meldevelopment's development teams as it enables them to get a better understanding of implementation tasks. For the quality assurance team, it provides insights into test cases. For the documentation team, it provides insights into the documentation requirements for products. The product backlog is also important to the marketing department to know which and when features will be ready to show to the potential customers or to include them in marketing campaigns.

> Get very high-level requirements from product planning … the Project Manager calls meetings to start fleshing things out … whole team is involved. (Principal Engineer)

> Project managers ensure that implementing is based on the priority, we're in session constantly … project managers engage with the field … our sales teams go directly to the project managers. (Product Manager)

Adapting low-level planning

Described below are some of key adaptations of Meldevelopment's low-level planning phase to achieve a better team approach.

5.3.1.1.2.1 Adapting with QA and documentation collaboration

Since adoption of an agile approach in January 2003, their design phase requires a team approach involving the development, quality assurance and documentation teams. The decision to involve these two functional teams with the design phase was made collectively by the engineering department to enhance cross-functional collaboration and to help each other to deliver features in short development cycles. Prior to agile adoption, the engineering department at Meldevelopment had a dysfunctional working relationship with each other. In April 2003, the team collectively agreed to adapt the approach used to elicit low-level requirements based on the clarity of the high-level requirements. The senior engineers suggested adapting the approach to elicit requirements on projects, which was agreed by the engineering team during a project reflection meeting. This adaptation decision was made since they continuously experience complex high-level requirements which require different approaches to beak

them into smaller tasks for implementation. Based on the nature of a project, the software engineers, project managers and product manager collectively decided the approach to take to elicit low-level requirements. If they agree to take the test driven approach, then quality assurance engineers are required to be more prominent and provide the guidance to identify the low-level requirements. If the project is agreed to be interface driven, then the approach is adapted to make the documentation team provide the guidance through user interfaces to elicit low-level requirements. A major benefit of this adaptation is that a product backlog is swiftly created for immediate implementation within a few days of a design phase.

> Have a brainstorm, all the different ways of testing it [QA driven] … an adaptation of using test driven technique to come up with a requirement … decide the best way to understand requirements … who needs to talk … try something different. (Director of Software Engineering)
>
> Now we have QA and documentation team sitting with us during design session … generally we have a separate QA team and documentation team. (Senior Engineer)

5.3.1.1.2.2 Adapting use case

How the engineering team captures the low-level requirements is also adapted, from using use cases for a feature with a lot of end user interaction to capturing a technical feature with a written description. This adapting decision was proposed by senior engineers since they experienced that not all their requirements can be captured through use case diagrams. In June 2003, the engineering department collectively agreed to adapt on projects to employ use case, state requirements or both depending on the nature of the projects and the high-level requirements. This adaptation enables Meldevelopment to accurately capture and verify different functionalities in relation to their products. Later, the engineering team discovered that developing features based on just the functional requirements compromised the user experiences of their products. They ended up building products that were very functional but lacked the flow.

> A lot of end user interaction, tend to employ use case analysis … when more algorithmic, drive that through more stated requirements … if it is mixture of everything will have feature type of requirements [stated] and employ use cases. (Senior Engineer)
>
> Building with an engineer's mentality as opposed to the end user … just figured that one person was using the system … weren't thinking the company had administrators and managers, with different needs. (Director Software Engineering)

5.3.1.1.2.3 Adapting with user centred design (UCD) approach

The development teams started to use more use cases but it did not enable them to identify the different roles and their objectives in relation to the end users of their products. By middle of 2004, the engineering team experienced an increase in defects being reported by the customers and support team. Analysis of the defects by the engineering department identified that most of the issues related to the integration of their product with other systems at client sites, requiring Meldevelopment to provide help features for users for getting out of such situations. The engineering manager suggested adopting a user centred design (UCD) approach to identify the specific users, features that they are likely to use and the likely issues they may encounter. In December 2004, the engineering department collectively agreed to adapt their design phase with UCD. The UCD approach enables them to put user experiences at the forefront by learning from the actual or proxy users on how they are going to use the product and how it will fit into their working environment. Here, UCD involves inviting individuals who can provide the best insights into the likely users and creating prototypes to get feedback on the features and usability requirements.

> Defects reported by customers, the user centred design would've solved the problem … got to think how to help users overcome issues, even if they are not to do with our product. (Director Software Engineering)

> It starts with a face-to-face meeting with main experts … try to understand and define personas … brainstorming, whiteboard sketching, mock-up prototypes … send that around, get feedback and have a few iterations. (Team Leader)

5.3.1.1.2.4 Adapting with prototyping practice

Meldevelopment's development teams regard prototyping as an essential tool to implement quality products. On adoption of a UCD approach in December 2004, the engineering unit collectively agreed to adapt its design phase by developing different capability prototypes to identify and understand the features they were developing through a time-boxed approach. Senior engineers identified prototypes as an effective tool to get better and instant feedback on product features and architecture. They create throwaway prototypes and prototypes that become part of their products.

> Put a bit of software in front of someone … people just understand so much more clearly … some prototypes are for the UI, throw them away … some are architectural prototypes, become an evolutionary prototype, turns into a real product … just a quick way to mitigate risk, time box it. (Team Leader)

5.3.1.1.2.5 Adapting participants for user centred design

The user centred design approach is a key technique in Meldevelopment's design phase for getting different key stakeholders to collaborate with each other through brainstorming, whiteboard sketching, mock-ups, and prototypes to identify the different roles associated with the key features. The engineers realised that in some cases the product manager does not have sufficient knowledge to provide the real context of the user environments. In April 2005, the engineering unit collectively agreed to make team effort more effective by adapting the participants according to the nature of a project. The product manager, consultant, sales engineer or a customer is invited to be part of a design session or to provide feedback on artefacts produced through a design session. This adaptation enables the engineering team to accurately capture specific user requirements to provide a better user experience with their products.

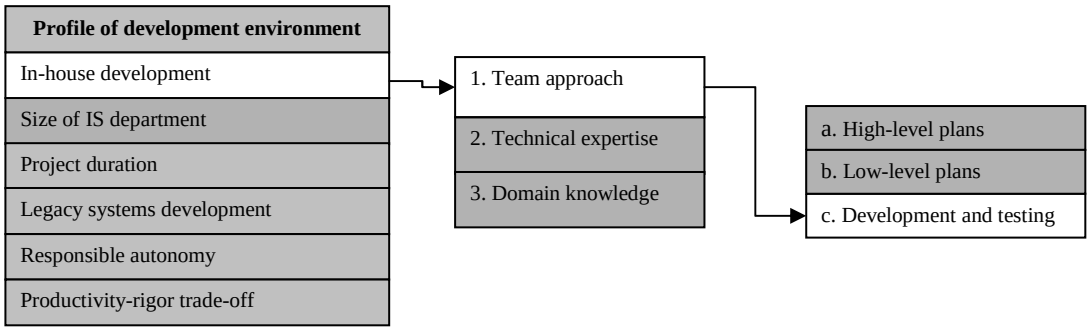
> Sometimes is very customer centric … meet customers and get their feedback … sometimes our product management team get involved … adapt how we get to the end requirement … product manager jumps into design meetings, give feedback over email when he's on road. (Project Manager)

> Adapted to identify a series of different personas … give them a name … written up a little paragraph on who they are and what their motivation is and what they're trying to do … work through as if they were using the product … then you come from the other person's angle … run them together. (Director Software Engineering)

5.3.1.1.3 Development and testing phase

This section provides information on Meldevelopment's development and testing phase and its adaptation to achieve an efficient team approach. Figure 42 illustrates this phase of Meldevelopment's agile development cycle in relation to its in-house development

Figure 42 Team approach in-relation to development and testing – Meldevelopment



Meldevelopment's team effort also extends into the development and testing phase. Through team effort, the engineering department continuously performs at the optimal level and avoids individual burnt-outs that would impact the productivity of individuals.

The following comments highlight the importance of Meldevelopment's engineering team effort.

> Issues with meeting deadlines … somebody thinks they have got a good design but haven't talked to anyone … without other people's input … end up building that aren't quite right … suffer that as a quality loss … end up spending a lot more time trying to bring it back. (Project Manager)

The development engineers work as a team adopting a team attitude, responsibility and ownership of the work that they commit to deliver. Software engineers' development efforts are supported by the quality assurance engineers and the technical writers working in their own teams but having significant cross-team collaboration with them. For the development engineers, collaborations are on-going and face-to-face with own team engineers including engineers from other teams to implement features. Quality assurance engineers ensure the compatibility of the products with other systems, while technical writers ensure product usability.

> Team effort improves quality … hard to think of different scenarios … bringing QA people, a different way of approaching the problem … UI people, who are thinking more like a user … end up opening issues that you overlook … reduces the issues later on, in terms of quality … productive in that you achieve better quality in same amount of time (Software Engineer)

Described below is the key adaptation of Meldevelopment's development and testing phase to achieve a better team approach**.**

5.3.1.1.3.1 Solo vs. pair programming

In January 2003, the engineers, project managers and management of software engineering unit through consensus adopted a solo rather than pair programming practice for code development. At the time of agile adoption, they had recognised that they had a larger number of senior engineers who were highly experienced with their product development. On the other hand, the engineering unit collectively agreed to hire only those engineers who are extremely talent for their agile development as they did with their previous approach. With a pool of extremely talented and experienced engineers, this engineering unit agreed that a solo programming practice is a better option for them, ensuring high productivity where more features are being implemented in parallel.

> Discussed pair programming with the engineers … have a very highly experienced team … that is why we don't want … heavily weighted to senior engineers. (Project Manager)
>
> Gives us a more productive outcome … working on all those things in parallel … main benefits are productivity. (Director Software Engineering)

Although the development effort at Meldevelopment is a team effort, they regard solo programming as a natural motivating factor for their engineers due to their very high levels of skills and talents. At Meldevelopment, engineers are also given the responsibility to carry out research to enhance and expand a feature. Hence, solo effort is seen as motivating factor for the engineer's ownership of sub-features or a main feature. At Meldevelopment, they regard the solo programming practice as an encouragement for pride-in-accomplishment and pride-in-contribution for implementing and enhancing the product features.

> Got some very talented guys, write very high quality code … they feel like it is something that they own … go that extra mile to make it absolutely simple. (Director Software Engineering)
>
> Engineers here are very good technically … a lot of talent … intimate knowledge of the technologies … the ability to pick up new things very fast … produce much higher quality work. (Senior Engineer)

5.3.1.1.3.2 Adapting solo effort

In August 2006, the engineering team adapted their solo programming practice to achieve high quality standards of their products. They incorporated code reviews with their solo programming practice where code developed by an engineer is reviewed by another engineer. This adaptation was suggested by a senior engineer and collectively agreed by the engineering team as a result of a high rate of bugs identified by the quality assurance engineers when testing iteration builds. On analysis of these bugs, the engineers agreed that most of the bugs reported by the quality assurance engineers would easily be picked up by another engineer reviewing the code. The code review now is a light weight practice, an over-the-shoulder informal process in their agile approach to identify product vulnerabilities and to uncover bugs before any code is submitted for a build. This adaptation at Meldevelopment not only enables them to minimise bugs with their iteration builds but has also enables engineers to learn from one another to implement better code.

> Adapted to get higher quality and better results … code reviews was one of the things that we did … people are working on code … before check it in give it to another person and explain it to them … come up with things that they hadn't thought. (Director Software Engineering)
>
> Get a code review for my own code or doing the review of someone else's code … quite happy to ask questions or take any advice on board that they might have … ask different people quite often … get some different opinions. (Engineer)

In May 2007, the development teams at Meldevelopment adapted their solo programming practice with another practice, a pair discussion involving a quality assurance engineer. With the engineers writing the unit tests now, the previous group effort by the quality assurance team to test the product using different angles no longer applied. This issue was highlighted and discussed with software engineers by the quality assurance team during project review meetings. The company had also decided to offshore all the repetitive manual tests of their product making QA engineers available to work more closely and to be integrated with the development teams. The adaptation decision to have pair discussion was collectively made by the senior engineers to involve an informal whiteboard discussion between a development engineer and a quality assurance engineer before any code development to identify vulnerability issues associated with the task being implemented. The insights now provided by quality assurance engineers during pair discussion make available useful information allowing engineers to implement better unit tests.

> Adapted our test thinking recently … think all the ways this product could fall apart … get someone of the QA team … go around a whiteboard , if this was already built, how would I break it … gives a developer series of things to think about. (Director Software Engineering)
>
> A QA member and developer brain storming how they would break a enhancement before the enhancement is built … some of that turns into writing in unit test cases … the real kind of insight comes in to helping the developers. (Senior Engineer)

Having two software engineers implementing the different parts of a task through collaboration was another adaptation of their solo programming effort in September 2007. Their development teams frequently discover that some of the tasks require more than a single engineer's coding effort to be able to deliver in a single iteration cycle. Hence, team leaders and project managers proposed sharing of a large task by engineers, which was agreed upon by the engineers on a project. This adaptation enhanced their solo programming practice with task sharing practice, helping team to deliver a high priority task in a single iteration and fostering a better team effort and understanding between engineers working closely with each other.

> A couple of people will be paired up, working on the same thing … each day have a little chat … divide [task] up … do their bits and then share it. (Director Software Engineering)
>
> Working with a few people, particularly when you're working closely … often you are sort of coming at it from slightly different angles … end up covering a lot more … thinking of different scenarios … therefore building it in, and don't have quite so many issues later. (Engineer)

5.3.1.2 Technical expertise

Information on adaptation of Meldevelopment's in-house development regarding its technical expertise is now provided (Figure 43).

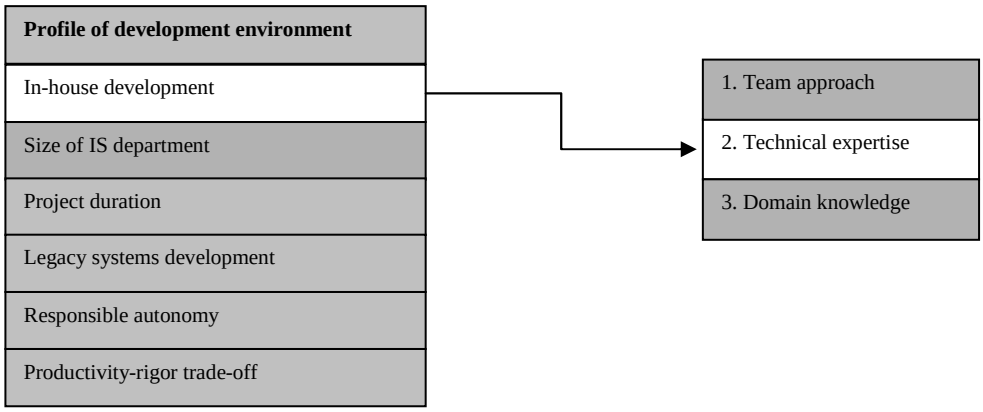
5.3.1.2.1 Engineering roles

At Meldevelopment, their agile approach is adapted from the past development roles they had such as graduate engineer, engineer, principal engineer, senior engineer and project manager. A new senior role, the director of software engineering was created when an agile approach was adopted in January, 2003. At Meldevelopment, this leadership role provides the support and coaching for the engineering unit to successfully adopt an agile approach.

> Kept the same job titles … our engineers had responsibility to design and build … just software engineers … had the complete life cycle responsibilities. (Director Software Engineering)

> Software engineers do everything in the entire life cycle … analyse requirements …negotiate with the stakeholders … design , implementing it, write all the test designs and test it as well. (Engineer)

Figure 43 In-house development in relation to technical expertise – Meldevelopment



5.3.1.2.2 Engineering responsibilities

Graduate engineers contribute to the design, development and testing of Meldevelopment products, while engineers are responsible for these activities and also provide support and consulting services for their clients. Senior engineers provide leadership in the design, development and testing of their products including providing support and consulting services to Meldevelopment's clients. They also investigate requirements for new products. Senior engineers undertake team, technical or project leadership responsibilities under the supervision of project managers. They also provide training and mentoring for graduate engineers in the relevant areas of technology, products, processes and quality requirements. Principal software engineers at Meldevelopment provide architectural vision for the design and development of their

products, including providing the architectural consulting for their clients. Principal engineers need to have up-to-date specialist knowledge in a broad range of technical areas which are applied to the evolution of their products and to identify emerging requirements. The project managers are in charge of the projects while their director of software engineering is in charge of three functional teams within the engineering department.

5.3.1.2.3 Engineering career path

Meldevelopment provides a well structured career path for its software engineers. They have three levels of technical expertise: graduate engineer, engineer, and senior engineer; there is a well established career path for progression. For a graduate engineer it takes between four and five years to acquire sufficient technical expertise and experience to become a senior engineer. Beyond these three levels, an engineer can become a field consultant, team leader, project manager or product manager. An engineer who chooses to stay with development becomes a principal engineer requiring ten years of development or systems consulting experience and two years experience at Meldevelopment as a senior engineer.

> Giving them different opportunities for them to grow … having an environment that people enjoy working in is a big thing. (Director of Software Engineering)

> Performance plans … meet with a developer, discuss their ambition … incorporate that into the work that they do … set them goals for next six months, at the end evaluate and make more goals going forward … we are looking after them in terms of their career and progression. (Project Manager)

5.3.1.2.4 Engineering talent

From the outset, the company focused on recruiting extremely talented engineers. Meldevelopment requires individuals who are highly intelligent and have an extremely competent technical background. Most importantly, they are able to communicate effectively in all situations during development and have empathy for others. The team leader provides information on the company view of extremely talented engineers.

> Quick thinking, grasp concepts … thrown into situation, understand details, assess and come up with some model … make informed decisions … empathy for others … will be disagreements, understand where they're coming from … it is all about hiring … hiring the best graduates, fresh, eager to do things and learn … hired some really experienced people … have to let people go who don't have the talent. (Team Leader)

5.3.1.2.5 Hiring criteria

At Meldevelopment, excellent problem solving ability, interaction skills and the ability to contribute effectively to their collaborative product development effort became

extremely important criteria for hiring new engineers at all levels. In their agile environment, the interpersonal skills of developers such as interaction, collaboration and co-operative skill sets are as important as their technical skills to successfully implement their highly innovative features by continuously working with others.

> Communication skills are really important … got talent and have no way to express it to others then it's a tough situation … not performing anywhere near as well with someone who can communicate and express it well. (Director Software Engineering)

> Have good communication skills … seems to be less document put in front of you … more flexible to get a good understanding … there is a lot more creativity and input that has to come from you as a person. (Engineer)

5.3.1.2.6 Adapting the hiring process

To ensure that new recruits have effective interpersonal skills, Meldevelopment's hiring process was adapted from giving programming tests and conducting interviews to conducting interviews and giving design tests to the potential candidates in April 2003. This approach was suggested by a senior engineer during their project review meeting since close interaction and working together as a group became the most important work practice in their engineering unit. This engineering unit collectively agreed upon this approach for hiring new individuals and empowered engineers to be part of their new hiring process. During another project review meeting the director of software engineering also suggested to adapting how the job advertisements are written to attract good candidates to select the required talent. This suggestion was as a result of some engineers leaving the company as they could not adapt to work in their new approach for software development. In June 2003, the engineering unit collectively agreed to adapt to further enhance their hiring process. The new recruitment methodology at Meldevelopment has helped the engineering unit to identify and recruit capable individuals by engineers who fit well with their agile approach for software development.

> Human factors are more important now … switched to new interview methodology … do a design interview … get the candidate, and engineers around a whiteboard collaboration … lead the person down a path and challenge by changing the problem … see how they handle negative feedback about their idea … whether get really possessive or feel insulted … our ads are different to a normal job ad … talking about this is your chance to do something amazing … is going to be challenging … attracts a lot of people. (Director Software Engineering)

> Get to choose who we get in our company … design interview … can this person analyse a problem, accept criticism and capable of changing their minds and taking on a better option … have good communication skills. (Senior Engineer)

5.3.1.2.7 Adapting the selection criteria

In January 2007, Meldevelopment adapted their selection criteria for new recruits. This adaptation involved hiring graduates than hiring industry experienced engineers to have a right balance of technical expertise within the organisation. Since their agile adoption, they were mostly hiring experienced engineers and their development teams became heavily oriented with senior engineers. The future implication of only hiring at senior level was realised by the engineering unit, where in future they may not have a sufficient number of experienced individuals with high technical knowledge to enhance and expand their own products. The director of engineering made the decision to hire new recruits at graduate level to grow them so that the company in future will have the necessary expertise and experience on a continuous basis.

> A success factor … got a lot of guys that have stayed here … had some turnover, replaced with people that are experienced … get different insights … back into a lot more of the graduate recruitment … the spirit of graduate is very motivating. (Project Manager)

> Last couple of years with turn over … replaced with people at senior levels … back into a lot of the graduate recruitment … fresh young guys that are coming in to be the leaders of tomorrow … having a good balance is important … graduates coming in is always very strong and motivating … re-energises the group. (Senior Engineer)

5.3.1.2.8 Adapting senior and principal engineering roles

In December 2003, the senior and principal engineer roles were adapted to also focus on the product architecture. This is a new responsibility for a senior or principal engineer in their agile set up when sub-teams are formed. However, this informal role collaborates and works with engineers in sub-teams to make architectural decisions. This new informal role was proposed by the director of software engineering and agreed by engineers and project managers during a project reflection meeting so that architectural discussion does not get carried away for long periods of time due to disagreements. Early in their agile adoption, long drawn discussions often occurred with little progress on achieving architectural vision, impacting their ability to quickly create a product backlog during the design phase. With this adaptation, sub teams formed on projects are now able to make swift architectural decisions during the design phase.

> Dedicated to the architecture and design … talking to the guys, helping create architectural vision … still senior or principal engineer … achieving consensus … not trying to say how

> things will be … mediating and get all the right people involved to make decision. (Director Software Engineering)
>
> They're more experienced … they have a greater weight in the final decision … might come forward with a general design for other people to comment … sometimes we will design it together … other times someone will come forward with a proposal and other people can look, comment and change it … have those sort of technical lead. (Engineer)

5.3.1.2.9 Adapting project manager role

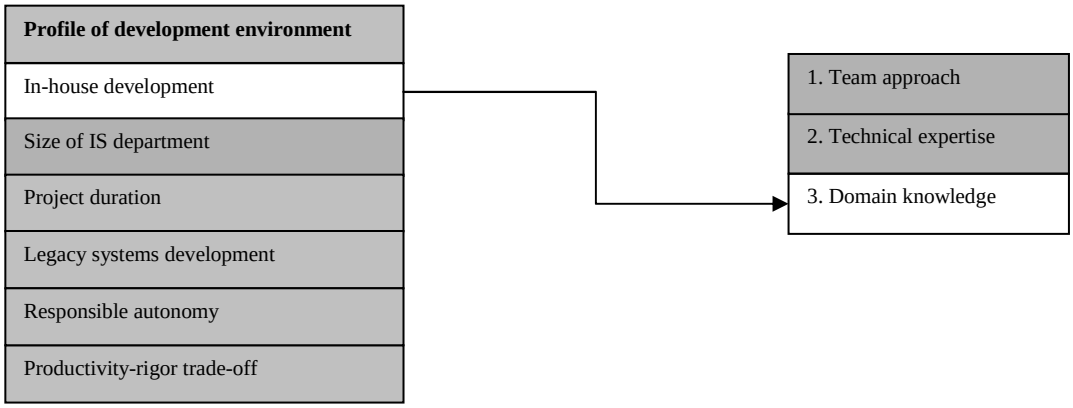
In January 2007, the project manager role was adapted to a product development manager role by incorporating product management functions. The reason for adapting the role was due to the product manager leaving the company, who was based at the Melbourne office. Frequently, the engineers had to liaise with the product manager to get contextual information in regards to the task they were implementing. At Meldevelopment, the project managers did not work in the field and lacked the product information, impacting the engineers' ability to swiftly access vital product information during implementation. The engineering unit collectively made the decision to request senior company executives for project manager roles to be adapted to incorporate product management functions. These roles now provide them with better domain support as a result of them doing product management tasks. At Meldevelopment, the product manager and project managers worked closely. The project managers were the proxy product managers for the engineers providing them with domain support.

> Product manager moved onto a different company… adapted to manage product planning with the managers in US … adapted is that project managers got more responsibility for product direction rather than just managing the team. (Director Software Engineering)
>
> Sometimes project managers don't have the context to make some decisions. You need to be able to know more about what the customer is thinking and the product manager has that context. …. representing our customer in a way. (Senior Engineer)

Next, information on adaptation of Meldevelopment's in-house development in regards to its domain knowledge is provided, as shown in Figure 44.

5.3.1.3 Domain knowledge

Figure 44 In-house development in relation to domain knowledge – Meldevelopment



To stay ahead of the competition, Meldevelopment makes a significant effort in trying to understand the environment in which their products will be used. For Meldevelopment, domain knowledge is critical for their entire product development process. This knowledge enables them to contribute ideas for high-level product requirements. For creating and prioritizing the product backlog, it enables them to identify and break tasks (low-level requirements) into their smallest form to fit into an iteration cycle. Domain knowledge helps the engineers to write appropriate unit tests and to develop features that meet specific quality requirements. For quality assurance and documentation teams, it helps them to test the products and to write manuals from user perspectives.

> Business understanding … get faster turnaround on all the day to day decisions that need to be made … if didn't have any domain knowledge, not able to take it to the next level of detail … talking to people they get an idea, then go through and implement. (Project Manger)

Over the years the engineering department has significantly built the domain knowledge of its products, usually learnt from a variety of sources. With the adoption of the agile approach, their main source for domain knowledge is their project manager (now product development managers) including the sales & marketing team and the product manager who is co-located with the engineering team.

> Have people in the domain for 10 years , product managers [and] engineers … need domain expertise … people can always ask questions at any time and get a quick answer … project managers play that role. (Director Software Engineering)

> Our team leader is a project manager … generally has very good domain knowledge and he does all the liaising with the internal stakeholders. (Senior Engineer)

At Meldevelopment the project managers work closely with the product manager. Therefore, they have sufficient domain knowledge and become the main source for product related information for the engineers when the product manager is in the field. They play the role of on-site customers. The Meldevelopment project managers have both technical and business backgrounds. Both of them started as graduate engineers with the company and moved up the ranks to become project mangers. One of the project managers worked as a consultant implementing products at customer sites. Their engineering department has a permanent presence of project mangers with substantial domain knowledge.

> Project manager is more readily available … if it is critical … talk to the product manager … sometimes project managers don't have the context to make some decisions … need to know more about what the customer is thinking … Product Manager has the context. (Senior Engineer)

Next information is provided on adaptation of their approach by which their engineers enhance their domain knowledge.

5.3.1.3.1 Hiring domain experts

Meldevelopment hired an individual at principal engineer level with significant domain expertise. Most recently they have hired a leading industry expert as a product marketing manager, based in the United States. He regularly spends time at the Melbourne office and also makes himself available through phone conferences and emails.

> An engineer worked in a similar area, that really helps … our domain expert isn't in the building … sending emails and waiting for the responses overnight … we have him here 6 months out of the year. (Team Leader)

5.3.1.3.2 Adapting domain knowledge source

Since the adoption of user centred design approach (UCD) in December 2004 clients have presented to the engineering unit about their businesses, and discussed issues that they have and how can they be solved. The decision to invite or work with an existing client is usually made by the team during the design phase to learn about fuzzy innovations, client requested enhancements or client request features. After the adoption of UCD approach, the engineers also agreed during a project retrospective meeting to invite sales consultants to present to the engineering unit in regards to clients' views and issues with their product. This additional approach to adapt the source for domain knowledge was made collectively by the engineering unit based on a senior engineer raising concerns about long discussion and arguments during the design phase. Also, the

engineering unit with the product manager has collectively agreed to encourage engineers to visit customer sites with consultants to learn more about client environments. Engineers find these adaptations to be very useful in enhancing their domain knowledge and avoiding lengthy arguments in coming to a team agreement on what exactly is to be built.

> [Clients] present to our engineers on what they do and some of the issues they face … consultants come back and present, where the customer like and did not like the product … engineers go on site with consultants. (Director Software Engineering)
>
> One of our biggest problems is our decision-making ability … we tend to argue, we try and agree on a solution … that is probably one of the hardest things. (Senior Engineer)
>
> We've talked about trying to get that to happen more … I don't think it happens as much as it should … we've had the sales guys give a lunchtime presentation … has been very useful. (Senior Engineer)

Next, information on adaptation of Meldevelopment's IS department (engineering function) is provided (Figure 45).

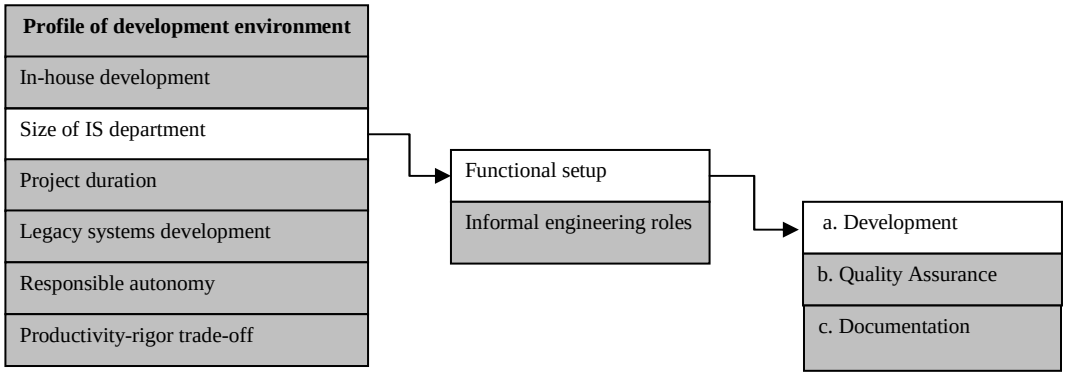
Figure 45 Size of IS department –Meldevelopment

Profile of development environment
In-house development
Size of IS department
Project duration
Legacy systems development
Responsible autonomy
Productivity-rigor trade-off

5.3.2 Size of IS department

The findings in relation to this adaptation factor are presented according to the themes that emerged from the analysis of data collected from Meldevelopment; these are functional setup and informal engineering role adaptations, as illustrated in Figure 46. First, information is provided on Meldevelopment's functional setup.

Figure 46 Development setup in relation to size of IS department – Meldevelopment



5.3.2.1 Functional setup

When Meldevelopment adopted the agile process the setup remained the same with three functional areas: development, quality assurance, and documentation. Meldevelopment has 31 technical staff members: 24 software engineers, 5 quality assurance engineers (testers), and 2 communications engineers (technical writers). The management team consists of 2 project managers, a quality assurance manager and a documentation manager, under the director of software engineering. First, adaptation information is provided on Meldevelopment's development function, as illustrated in Figure 46.

5.3.2.1.1 Development

With their software development function, work is allocated into small teams. The teams are organised based on the development projects. The management and engineers have an understanding that each development team must not have more than 7 engineers. Prior to agile development, Meldevelopment had experienced that the best development efforts were achieved through small teams. In their agile set up, small teams ensure swift decision making, effective interaction and excellent support to understand features they are implementing.

> More than seven … hard for communication, coordination and collaboration … easier when a pod of 4 people … all on the same wavelength … can overhear , pick up on things and contribute … in design meetings … 3 people that are collaborating will come to consensus fairly quickly … up to 8 people it is so hard to get closure. (Director Software Engineering)

> The main thing is the communication … to interact and talk to people and discuss ideas … people generally know what they're doing … are good at communicating … some people are a bit quieter than others, if you start the communication they're really good at communicating. (Engineer)

The project managers have also learned that team management is most effective when they have small teams. As managers, they cannot give every engineer enough attention when the number of individuals in a team is higher.

> A single manager can't effectively manage more than 8 people … break it up; give autonomy to do something independently of the other teams. (Director Software Engineering)

> Team lead … you can go to him for guidance or help … catch up one on one … struggling with anything … might even have a personal issue, individual people can raise any concerns … gives us a bit of confidence and trust that he cares about us. (Engineer)

5.3.2.1.1.1 Adapting number and team size

Normally, a Meldevelopment development team will undertake development of a product throughout its entire lifetime. However, the number of teams is adapted from one project to another. For a project, it is determined by how important the product is for Meldevelopment at the time when the project is undertaken. The number of teams on a project varies from 1 to 3 and the number of engineers from 3 to 7 for each team.

> Look at the size and how much we need to deliver … more than 6 people, break into a couple of teams … put number of people based on how important the product is to us at the time … if something bit further out … have a smaller team of 2 to 3 people … if it is a key technology, have 6 or 7 people. (Director Software Engineering)

> Changes every 6 months… focus can change quite a bit … when that happens the actual teams are broken up … usually a team just for a project and then the team disbands. (Engineer)

5.3.2.1.1.2 Adapting team size during projects

At times, Meldevelopment increases a team size by one or two engineers to boost their development capacity. This adaptation for a team size is an effort to meet delivery commitments when a project falls behind. This team adaptation has been carried out since Meldevelopment adopted an agile approach in January 2003. The need to adapt on a project is identified by the project manager, team leader or technical leader. Their practice is to be as open as possible when moving engineers into another team. A discussion takes place, as soon as possible when the problem is identified, between the project managers, technical and team leaders, and director of software engineering to decide which particular skills are needed. An engineering unit meeting is called explaining that engineers' help is needed and they are identified from the other development teams to boost the capacity of the team which is lagging behind on a project. From the management perspective, they facilitate engineers volunteering rather than management having to make decisions about who to move. One of the major benefits of this adaptation is that it helps to build a team culture among the engineers.

> Pull everyone together to say we are in trouble on this project, looking for people to come across and help … this person has got this skill for this, so they're going to come across … if anyone else wants to chip in, let us know. (Director Software Engineering)

> Kind of like who can we squeeze off the other teams to put onto this project … had to kind of build around that … not sure if there is a magic formula. (Engineer)

5.3.2.1.1.3 Adapting number of teams during projects

Some of the Meldevelopment projects require additional teams to meet fast approaching deadlines, especially when the release dates have been brought forward because a project has become high value. At Meldevelopment, this business decision has been made by involving the engineering unit since their agile adoption in January 2003. In such cases, the adaptation decision at development level is collectively made by having a meeting between the director of software engineering, project managers, and team and technical leaders. The development effort is adapted into two sub-teams; one sub-team with 6 or 7 engineers and another sub-team with 4 or 5 engineers. The project is headed by the project manager, who will also be the team leader for one sub-team. The other sub-team is headed by a senior engineer. This adaptation helps to assign senior engineers with leadership roles to enhance their career path.

> A project need more people … changing in priorities, a big customer thing comes up … gone up to 11 people, effectively made a sub team … someone in-charge of a group of 4 or 5. (Director Software Engineering)

> In our team we will be working on two components at once … so there will be basically two teams working on that sort of stuff … decisions will get made on those developers including the team lead. (Engineer)

5.3.2.1.1.4 Adapting teams to provide complete solution

The team's ownership of products is adapted based on Meldevelopment's focus for the next business quarter. The engineering unit has being adapting its development effort based on such business decisions since agile adoption in January 2003. This decision is collectively made with the engineering unit during product planning meetings. In such cases, the adaptation decision on team structure at development level is made collectively by having a meeting between the director of software engineering, project managers, and team and technical leaders. This adaptation helps to spread the knowledge base of individual products to other teams, without having one team focused on just one product. Hence, a major engineer benefit is that they are able to enhance their technical knowledge and skill set by working on different products. A Meldevelopment team own a product; they have all the technical knowledge of that

product. For the next release cycle of a product, they share the project with another team. Such adaptation is required for Meldevelopment to fulfil delivery commitments when customers request a complete Meldevelopment solution.

> Have functionally oriented teams … other times, to have the whole solution to the customer … have a team that covers bits of product a and bits of product b … look at the highest priorities for the business … rank them, can't change team structures on project basis, wasting your time. (Director Software Engineering)

> Our core products … that team remain as a team … the other two, is a bit more transferable. So there will be people going between those two teams … that would be more often … just kind of the way it seems to work well here. (Engineer)

5.3.2.1.1.5 Rules for teams on same projects

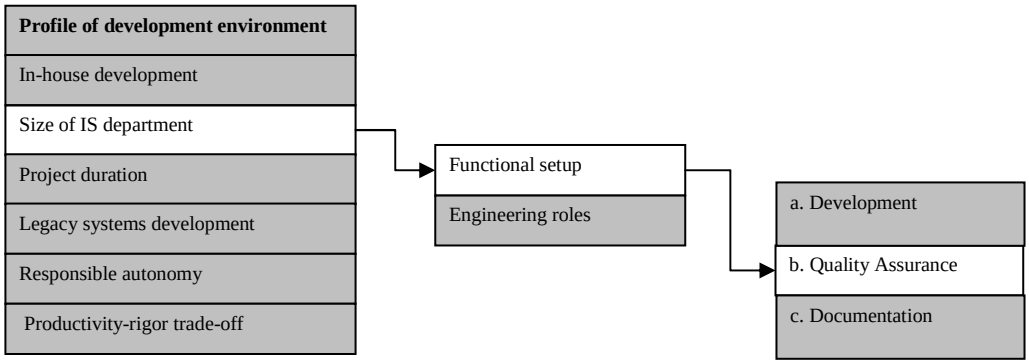
They also adapt how teams on the same project are allocated development tasks. Task allocation is done on a component basis, i.e. one team takes one part of the project and another team the other part. At other times, task allocation is based on an end-to-end scenario where teams work on the same code; each team owns one big use case from end to end, and another team owns another big use case. In such cases, there are no clear lines to separate the work between the teams. When changes are made, it also impacts the other teams. These issues need to be clearly communicated with the other teams. For this reason, Meldevelopment's agile development teams have some informal rules (explained below) that allow teams to work together, smartly.

> Make sure that they are not breaking the builds … cause the other team to move off line … have regular meetings between the teams … if doing changes in an area owned by the other teams, get their input and review … have a rule for communication in the common infrastructure. (Project Manager)

Next, adaptation information is provided on Meldevelopment's quality assurance function, as illustrated in Figure 47.

5.3.2.1.2 Quality assurance (QA) function

Figure 47 Quality Assurance setup in relation to size of IS department – Meldevelopment



Prior to agile adoption, the QA team had two key tasks before Meldevelopment products were released: core reviews and manual testing. The core reviews involved doing audits to verify that all Meldevelopment quality processes were followed by the development engineers. Such audits ensured consistent interfaces and also the traceability for any changes to their product. The QA engineers also carried out manual tests at the systems level to certify the quality requirements of their products. This included the compatibility, functionality, usability, reliability, performance, and scalability of products as well as their supportability in terms of their ability to support migration from one system to another. The QA team functioned independently of the development teams where the effort between the two was on a formal basis. Templates were developed as official documents and forms, which required rigorous adherence by the development teams.

> A lot of templates came out … what requirements [and] design should be like … quality team was telling what we should be doing … every release they would audit, spec doesn't have the right version number or haven't updated the history of what you have changed, they can't sign it off … that's just crazy. (Director Software Engineering)

> To ensure that all the [development] processes and quality processes are followed … there is also traceability, there's paper work. (Quality Assurance Manager)

5.3.2.1.2.1 Adapting with interactive and collaborative approach

On adoption of an agile approach in January 2003, the QA team adapted their approach to work very closely with development engineers. This adaptation decision was made by the engineering manager to foster a team effort in their engineering unit. The QA team is now proactive in the design phase helping engineers to elicit low-level requirements or tasks to create product backlogs. In addition, the QA and software engineers

collaborate one to one in each others' work environment to identify, explain and fix bugs during their iterations. As a result, core reviews carried out by their QA team were abandoned in December 2005 requiring QA engineers to pair with development engineers to have face-to-face collaborations. Despite these upfront collaborations, the QA team discovered many bugs with their builds on some of their projects. Highlighted by the QA team during a project review meeting, the engineering unit collectively agreed to the director of software engineering's proposal to co-locate a QA engineer with each development team. Since February 2006, a QA engineer is co-located with each development team providing engineers with additional information and carrying out tests during development to help minimise bugs with builds, which require three times the engineering effort to fix when discovered later, during build tests. This adaptation creates a more productive development environment at Meldevelopment and provides the QA engineers with the opportunity to discuss the coding and quality standards one-to-one with the development engineers, while providing input on the development tasks.

> Adapted into a QA team that is separate but very integrated with developers … core reviews dropped because we are into market-driven development … previous projects we used to get the product that would have issues … quality starts with the developers … catching defects and fixing it is 3 times the work … we keep reinforcing that. (Quality Assurance Manager)

> I have got a team of six developers at the moment … we have got one QA engineer who works very closely with them, but reports through the QA manager. (Team Leader)

5.3.2.1.2.2 Adapting with core reviews

Despite working closely with development engineers, QA engineers felt that quality standards of the Meldevelopment products were not being met. Highlighted by the QA team during a project review meeting, the engineering unit collectively agreed to adapt with core reviews because of them hiring new software engineers at graduate level. In May 2007, some core reviews have been incorporated back into their testing activities on request by the QA team. This audit activity is based around the test driven development practice to check that development engineers are using adequate test procedures for unit tests and are following their coding standards. This core review also ensures that the development engineers have tested a feature through unit tests before it is delivered to the QA team. On the other hand, core reviews help graduate engineers to learn and develop their testing skills based on discussions with QA engineers on issues identified during this activity.

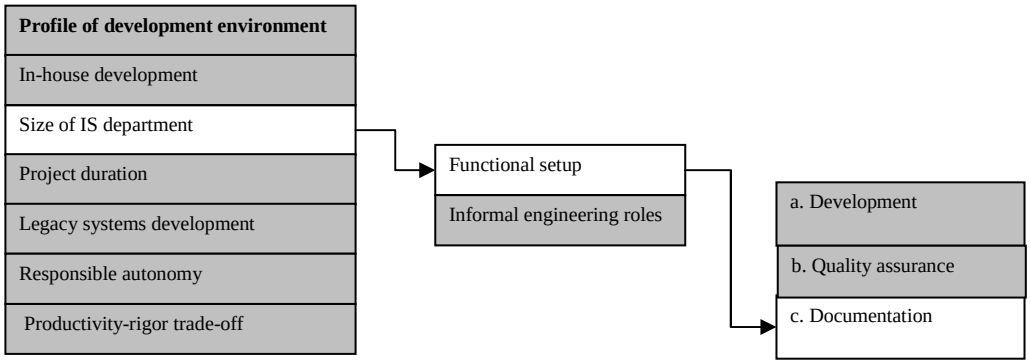
> Back to some audits … core reviews to ensure that developers have tested the product at a unit level … there's traceability, to ensure coding standards … a lot of new people give any [inconsistent] window. (Quality Assurance Manager)
>
> We have got a couple of quite new guys in the team… similarly the other team have had a couple of new people join their team in like the last 6 months or so. (Engineer)

Next, adaptation information is provided on Meldevelopment's documentation function, as illustrated in Figure 48.

5.3.2.1.3 Documentation function

At Meldevelopment, the documentation role is formalised as 'technical communications engineer'. The documentation team aligns Meldevelopment's product development with the user perspective. They also work closely with the product manager to learn about user experiences.

Figure 48 Documentation setup in relation to size of IS department – Meldevelopment



They have two technical communications engineers at the senior level. Besides their writing skills, they are required to have a technical background; one worked as a programmer and the other had 15 years of documentation experience in a software development environment.

> Need to be quick on the uptake … can't understand what the engineers are talking about, will struggle to produce documentation that is helpful to the user … not knowing the implications of migrating from SQL Server 2000 to 2003, [will] not able to explain … technical understanding are a must. (Senior Technical Communication Engineer)
>
> The documentation team are really looking at the product from quite a high-level just trying to understand how it works, if I was a user. (Engineer)

One of the key challenges with agile development for Meldevelopment's technical communication engineers is to get the documentation done before the release date. Their agile development has put increasing demands on the technical writers since the

development engineers no longer produce upfront a complete requirements specification. Previously, a requirements specification document was a good indicator of features allowing the documentation work to start early in their projects. This is not the case with their agile approach.

> Harder to get the information about what's being built … a lot more fluid, things change more frequently … a mad scramble at the end, to get the documentation finished in time for release. (Senior Technical Communication Engineer)

> They expect me to let them know particularly of any UI changes that might go in … keep them up to speed with what's going on, particularly as the date gets closer, they start taking some screen shots if any of those screens change. (Engineer)

5.3.2.1.3.1 Adapting with collaborative approach

On adoption of an agile approach in January, 2003 the documentation team also adapted their approach to work very closely with development engineers. This adaptation decision too was made by the engineering manager to foster a team effort in their engineering unit. The documentation team now contributes in the design phase helping engineers to elicit low-level requirements. This contribution early in projects gives the technical writers an insight into their documentation tasks. However, their involvement at the design phase only is not enough as changes happen during iterations. In their agile environment, the documentation team continuously interacts with the development and QA teams to learn what is being implemented in iterations. The following statement made by the senior technical communication engineer highlights the reason why at Meldevelopment it is important for them to be interacting continuously with software and QA engineers.

> Vigilant about communication … have to jump up and down and say, didn't know that you were building that … thought we said last week that you were going to do this … then they say but the QA said this and we refactored. (Senior Technical Communication Engineer)

Despite these challenges, the passion and commitment of the communication engineers enables them to get the documentation written, reviewed, and produced to fulfil Meldevelopment's release requirements. There is tool and other software support for the documentation team so that they can be as productive as possible, and to provide the mechanism for effective and on-going communication with others.

> Use Frame Maker and Web Works Publisher, single source for print documentation and online help … keep project pages up-to-date … people writing what-we-have-built, engineers recording signed decisions, BUGZILLA- managing bugs and issues.
> (Senior Technical Communication Engineer)

5.3.2.1.3.2 Adapting documentation with release notes

The technical communication engineers have adapted how they document iteration releases. At the beginning of a project they do little or no documentation because they expect changes during the project. Hence, documentation for early iterations is pushed towards the middle to the end of a project. In June 2003, they adapted to a more flexible approach of doing most of the documentation towards the middle to the end of a project, incorporating changes that have happened while the rest are packaged as the release notes with products. The documentation team made this adaptation decision themselves after a experiencing a couple projects in their first year of agile adoption. However, using release notes to provide documentation support for late implementation was proposed by the documentation team during a project review meeting and supported by the engineering unit.

> Iterative development, more time-consuming … end up doing a bigger percentage later in the project … gives you a time management problem … if they are still changing things - release notes. (Senior Technical Communication Engineer)

5.3.2.1.3.3 Adapting the size of documentation team

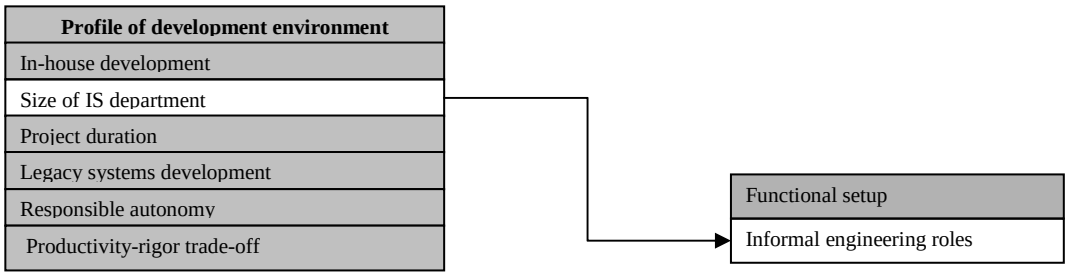
Meldevelopment adapted the size of the documentation team to have a highly productive and capable documentation team supporting their agile development. This adaptation was influenced by the talent of their communication engineers. Adapting the size of the documentation team was a joint decision between the director of software engineering and the documentation team to have a more productive documentation function by having the best talent. The current size of 2 in 2006 was adapted from 5 communication engineers they had in 2002 by not hiring to replace the technical writers leaving the company. The third individual in the documentation team is the team manager. Currently, it is a separate functional team working with all the development teams. On agile adoption, this was adapted from an individual technical communication engineer being part of a development team producing user documents for that team. Despite a small documentation team, engineers have support from technical writers to deal with usability issues with the features they implement.

> Had 4 or 5 people, a person in each team … now two writers, they're like a separate team … so knowledgeable in the domain. (Director Software Engineering)
>
> There are three [includes the manager] of us in the group, and we are servicing all the product teams. (Senior Technical Communication Engineer)
>
> Often go to them on UI ideas … before putting the text on the screen just to make sure that the documentation person is happy. (Senior Engineer)

5.3.2.2 Informal engineering roles

This section provides information on adaptation of engineering roles in relation to the size of the IS department at Meldevelopment (Figure 49).

Figure 49 Engineering role adaptation – Meldevelopment



5.3.2.2.1 Leadership roles

Meldevelopment has two key management roles within their development teams: project manager, and technical leader/architect. The two Meldevelopment project managers undertake team leader roles while being in charge of the projects. Here, 'team leader' is an informal role. A senior engineer, who had an aspiration to become a project or product manger, is given the chance to lead a small sub-team. The project managers do not get involved with the architecture or technical issues. The architect provides technical and architectural vision of products. As a product company, the technical architecture of their products is critical, making this specialist role within their agile teams extremely important.

> The project managers lead … someone is getting up into the management ranks, make them a team leader under a project manager … architect, is the technical lead to close off on architectural decisions … the catalyst … mediate and has the technical credibility. (Director Software Engineering)

> No official team lead job title … most experienced engineer in the team can be team lead … they've got as much work load as any other engineer. I am the principal engineer, comes with a few different hats … help other engineers out … they hit obstacles, technical ones … just help them with that … a role in the overall architecture, high-level design … often involved with the architecture of a product (Principle Engineer)

5.3.2.2.2 Role expectation: team leader

At Meldevelopment, team leaders are expected to have people management attributes that facilitate, motivate, and encourage engineers to work towards team goals. Here, the team leaders are required to create a positive work environment and to meet the needs of the individuals. Their team leaders also ensure that the work environment is driven by trust. Meldevelopment's product development requires good judgement by team

leaders where they allocate tasks which are significant and have trust in their team to deliver them. Meldevelopment's senior management expects team leaders to accurately and honestly report the state of the engineers' feelings, the work status and any other information that they might request.

> Trusting these guys … have a huge responsibility … worked with them for 6 or 7 years so know what they're like … how they deal with pressure. (Director Software Engineering)

> Help you in giving you some direction on that … passes down useful information, and relevant information, without bothering you with the politics … passes on both ways … helps each of the team members feel part of the team.… helps foster a lot of team things. (Engineer)

5.3.2.2.3 Role expectation: technical leader

A Meldevelopment, a technical leader is required to be an experienced development engineer with a high-level of technical competence. This role at Meldevelopment requires an individual to have excellent practical judgement. If individuals get obsessed with engineering purity, it significantly hampers their delivery schedules and business goals. Technical leaders make judgement calls to ensure that code is readable and maintainable, especially when code is complicated or refactored to a point where it was not readable.

> Architect [must] know extremely elegant code and business realities … keep refactoring something till it is so elegant, no one can actually understand how it works … a good practical judgement of when enough is enough. (Director Software Engineering)

> Got a responsibility of making sure that the code is healthy and to watch out for problems that can be fixed. (Principal Engineer)

5.3.2.2.4 Adapting to create leadership roles

At Meldevelopment, the leadership roles are created as needed. At times a development project is too large, so it is split appropriately requiring additional individuals to provide leadership. The decision to split a project is usually made during the design phase involving the engineering manager, project manager and engineers who are part of the project. Prior to agile adoption the engineering unit had realised the significant benefits of having work located in small teams. Since December 2003, the team structure is adapted for larger projects to create sub-teams with informal leadership roles. The complex nature of Meldevelopment projects requires an individual to manage one major aspect while another manages the other major aspect of a project. Breaking things up in a project with separate backlogs has ensured that project managers are not overloaded. The leadership experiences allow senior engineers to prepare themselves for career

progression for higher positions at Meldevelopment such as a principal engineer, sales consultant, project manager or product manager role.

> A big project, a small area … identify someone who can be a team leader … done that on an as need basis … manage the whole thing as if it's a small project … try to adapt … career progression. (Director Software Engineering)

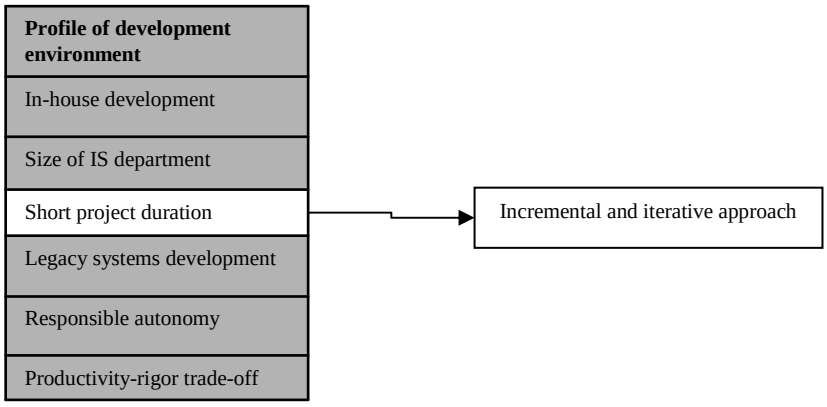
> Even with the same people … couldn't imagine by the time you get up to about 15 people … stretch the boundaries of what you can do in one team … have to break it … so usually decisions will get made…including the team lead (Engineer).

Next, information is provided about the short project duration at Meldevelopment. This is identified as another adaptation factor under the profile of the development environment in Fitzgerald's adaptation framework (Figure 50).

5.3.3 Short project duration

The findings in relation to this adaptation factor are presented according to the theme that emerged from the analysis of data collected from Meldevelopment; the incremental and iterative approach.

Figure 50 Short project duration – Meldevelopment



5.3.3.1 Incremental and iterative approach

Meldevelopment projects are between one to six months in duration. They have three major releases per year, one for each of their main products. They also deliver 4 to 5 patches of their products including a feature pack before a major release. Meldevelopment's target is at least one major product release for every quarter in a financial year. To achieve this target, their agile development is driven by incremental and iterative practices. Thus, all of their projects are feature driven with short development cycles, known as iteration releases (IR). The short iteration cycles enable

them to package those new features of Meldevelopment products that are implemented up until a major release date.

> Series of projects emerging at the same time … 3 major ones, one each for three development teams … minor releases and patches, some feature packs … probably 8 different deliverables. (Director Software Engineering)

> A lot of time, the features are driven from sales opportunities or customer demand … really need to assess the market … using an iterative approach ... ends up with a much better result. (Product Manager)

5.3.3.1.1 Two week iteration release (IR) cycles

The Meldevelopment projects have two week iteration cycles as part of their agile approach. For each iteration release cycle, the Meldevelopment development teams negotiate and commit to deliver the prioritized features from the product backlog. The implemented features from an iteration release cycle are delivered as a build to the QA teams for various quality assurance checks, and then to the documentation team to produce manuals and on-line help materials. The QA and documentation teams have a combined two week iteration cycle within which they carry out their respective tasks and get any bugs or issues fixed by the development engineers.

> 2 week iteration release … generally it is about a month … 2 weeks in development, then test for 2 weeks… fixed any bugs that they have found … got a feature completed. (Senior Engineer)

The two week iteration is regarded by their development teams to be enough for their development and testing cycle. The engineers feel that it not only gives them enough time to write code, but also to do other important activities that go with implementation, such as pair design, writing unit tests, fixing bugs, and doing code reviews. On the other hand, the two week duration for an iteration cycle does not overwhelm the QA and documentation teams with builds for their respective tasks.

> Long enough to get development done … too much functionality to be tested that swamps [QA] … not so many bugs that you are swamped … not just writing code, unit tests, design reviews etc. (Senior Engineer)

5.3.3.1.2 Adapting iteration release (IR) cycle

While a two week iteration cycle works well for most Meldevelopment projects, at times it is adapted to extend it to three weeks to ensure that a product feature is complete. A short team meeting is called by the project manager, team leader or senior engineer upon realising that most of the engineers will not be able to implement their task appropriately. Meldevelopment has an understanding that such situations arise due

to the highly technical and complex nature of their innovations limiting engineers understanding to provide accurate estimates. The decision to extend the duration of a single iteration is made by the project manager based on a team discussion and agreement. This iteration adaptation has prevented engineer burnouts due to them not working weekends and long hours to appropriately deliver iteration commitment.

> Extend them if a feature isn't ready … make iteration a week later and deliver it properly … did 1 week iterations to bring out a lot of stuff very early … 1 week was too short … 3 weeks was getting too long … 2 just felt about right. (Senior Engineer)

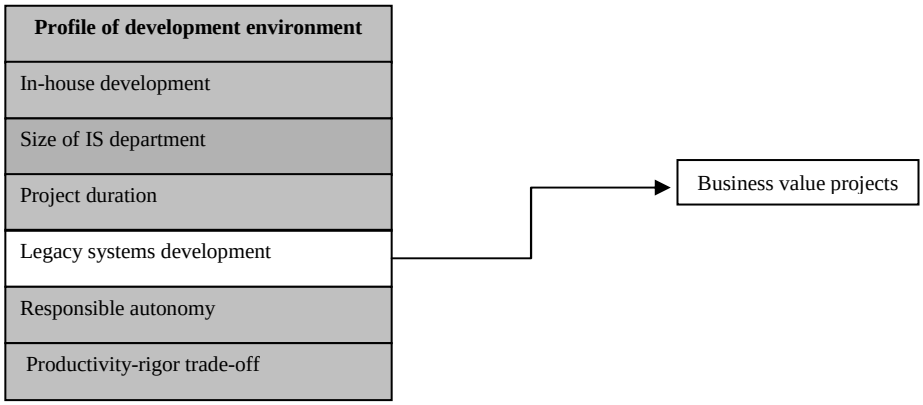
Next, information is provided on legacy systems development at Meldevelopment. This is identified as an adaptation factor under the profile of the development environment in Fitzgerald's adaptation framework (Figure 51).

5.3.4 Legacy systems development

The findings in relation to this adaptation factor are presented according to the theme that emerged from the analysis of data collected from Meldevelopment; the business value projects.

5.3.4.1 Business value projects

Figure 51 Legacy systems development – Meldevelopment



Since agile adoption in January 2003, Meldevelopment undertakes business value projects only. They use agile development to determine and enhance existing (legacy) products just as for their new developments using short cycles for development, quality assurance and documentation activities. However, such projects must provide business value for the company. They are undertaken if the current priority is the highest in their project portfolio. Meldevelopment project priorities are determined by customer and

marketing needs. They have five year product plans and detailed roadmaps for product releases for the next twelve to eighteen months.

> Project, look at the business case … a neat technical idea but can't see anyone spending money on it, wouldn't assign priority … existing customers [and marketing] come up with things that they need. (Director Software Engineering)

> [Priori to agile development] ended up with long drawn out projects … took a very long time to deliver … then when they were delivered we didn't always hit the requirements what the market was asking for. (Product Manager)

5.3.4.1.2 Product backlog

Just like for any new products, Meldevelopment creates and keeps a product backlog for all the likely features that could be improved upon for all their products. These improvements are mostly in the form of Low-level requirements at the functional level of a product. While the product backlog includes features that can be improved upon, it also includes new features that could be implemented to add value to the product. At Meldevelopment, all their developments are now managed using product backlogs.

> Kept a big backlog of enhancements… everyone is always want more things done … there's so many requests that come in. (Director Software Engineering)

> Get so many requirements coming in from different parts of the company … don't have the power to add every feature into the product. (Product Manager)

5.3.4.1.3 Adapting product backlog

In June 2003, the engineering department adapted how it prioritises implementing enhancements from their backlog. Because of the larger number of enhancement being requested, the engineers felt that they would never get to implement them all. The engineering unit collectively agreed during project review meeting that all high-level requirements relating to product enhancement must also be prioritised enabling them to swiftly implement them to reduce their enhancement backlogs. The priority decision for implementing the backlog is made by the project manager during the design phase based on the priorities set on the high-level requirements relating to feature enhancements. At Meldevelopment, high-level priorities setting now is a business decision done collectively during the product planning meetings based on the expected business value a enhanced feature will bring to the company. High-level priorities setting practice is adapted based on customer request, strategic feature or when enhancement was submitted. This adaptation now enables the engineering unit to implement enhancements, new innovations or both in parallel based on their business value to the organisation.

> Keep adapting the way we handle … tried strategies of just get this customer right… start working with other customers, everything in the backlog related to them … other times, go through the backlog that give a strategic product … sometimes, when they were submitted. (Director Software Engineering)
>
> Big challenges is prioritising … if one customer is willing to pay you a lot of money to do one feature … what is it going to cost me to maintain and support it, what's the opportunity within the market … add any value strategically to your product … a difficult decision to make. (Product Manager)

5.3.4.1.4 Adapting tool support

In October 2003, Meldevelopment adapted their tool support for creating and maintaining the product backlogs. This was done to have a better and more efficient tool support while minimising the overheads involved when using it. The project manager at Meldevelopment highlighted during a project review meeting that it was taking them unacceptable time to record and update tasks in their product backlog using their current software tool. An unproductive development environment was being experienced during their design phase. As a result, the engineering unit collectively agreed to purchase an effective tool support to manage their product backlogs. With their new tool Bugzilla, the engineering unit now experience an improvement in their planning, monitoring and the control of their backlogs.

> Had [a software tool] where we kept a backlog of enhancements … then have this tool called Bugzilla, more efficient to use … was causing too much wasted time. (Director Software Engineering)
>
> In support for the iterations, we've got Bugzilla, so people can keep an eye on it and see what's changing … central place that recorded design decisions … a sort of central repository. (Senior Technical Communication Engineer)

Next, information is provided on responsible autonomy at Meldevelopment. This is identified as an adaptation factor under the profile of the development environment in Fitzgerald's adaptation framework (Figure 52).

5.3.5 Responsible autonomy

The findings relating to this adaptation factor are presented for Meldevelopment.

5.3.5.1 Engineering autonomy

The engineers are empowered to adapt their agile product development and planning process to ensure that Meldevelopment's business objectives are continuously being

met. The engineers are entrusted with the responsibility for and the ownership of Meldevelopment's products and the development process

Figure 52 Developer autonomy – Meldevelopment

Profile of development environment
In-house development
Size of IS department
Short project duration
Legacy systems development
Responsible autonomy
Productivity-rigor trade-off

Meldevelopment's belief is that when engineers contribute ideas for development practices and help to implement them, then they are more passionate to use those practices as opposed to being told what to do by someone who is a step away from the actual work. The senior management of the company focus on product, sales, marketing, and other business functions. The engineering department is autonomous and empowered to do what they need to do to develop products and meet deadlines without being micro-managed by senior management.

> Senior management … trusting us as a team … not scrutinising every part of what we do … allowing us to do what we need to do with the development process … rather than being told by someone who doesn't do coding or testing, what you can do. (Director Software Engineering)

> The CEO is interested in what we're delivering to market … what sort of timeframes and making sure we have high quality … he trusts us to do the job … not bogged down in all the details of it. (Senior Engineer)

> We can respond to what the market needs, very quickly … that is part of the agile development … are extremely good at that. (Marketing Manager)

As an agile unit, engineers together with the individuals from their business function work as a team. Engineers are empowered and responsible for all the aspects of their product development, from the requirements analysis to taking part in product implementation at the customer site. The director of software engineering explains the outcome of providing engineer autonomy at Meldevelopment in the following quote.

> In a less autonomous role, won't necessarily grow your leadership skills … to interact with different parts of the company … speak to technical and non-technical people, marketing people, field consultants … gives you the greatest sense of maturity. (Director Software Engineering)

> Have an engineer talking to someone in support or in marketing, that you wouldn't necessarily talk … just to try and get some bonds and communication going. (Engineer)

5.3.5.2 Adapting engineer delegation

At Meldevelopment, engineer autonomy was adapted to empower them with a wide authority. On adoption of an agile approach in January 2003, the engineering roles were adapted with the authority to make architectural decisions and a role in project planning. This decision was made by the director of software engineering (and agreed by engineers) to make the development teams collectively be responsible and take ownership for development and delivery of software. Engineers in their sub-teams now collaborate with the other engineers and principal engineer to make architectural decisions of features they implement. They also take part in the design phase to create backlogs for release cycles and collectively plan each iteration cycle with their project manager or team leader. In the design phase, engineers are empowered to provide estimates and, within iteration planning, they are also empowered to re-negotiate those estimates. In July 2003, engineers were delegated with the ownership of their product enhancements and undertaking technical investigations of products. This decision was also made by the director of software engineer in recognition of their engineering talent and as a motivating factor for producing high quality work. With these adaptations of the engineering roles, engineers feel they are accountable for feature implementation.

> Get more autonomy, take on a different type of role … it is one of those things that tend to grow … do your own architecture, set your own estimates and all those things … but it makes you more accountable. (Senior Engineer)

However, individual delegation is based on the experience of individual engineers. Engineers are empowered to take part in product planning. On occasions, it is not practical to involve the entire development team. In January 2004, participation in product planning activities was adapted to include only a few senior engineers. These senior engineers are responsible for communicating design decisions made during the meeting to their teams. This adaptation decision was suggested by the project manager and a few senior engineers during their project review meetings. They experienced impact on their delivery commitments when the entire team took part in product planning. The engineering unit collectively agreed to this adaptation. In September 2004, the engineer empowerment to provide estimates was adapted with a re-negotiation practice with the team leader or project manager. The project managers noticed that some engineers were consistently providing low estimates for tasks. This was raised in their project review meeting by the project managers and the engineering unit agreed to a re-negotiation practice. At Meldevelopment, even as the engineers are delegated to

provide estimates and inputs for schedules, at times they are reassessed to ensure that they are based on the right information and that the engineers are heading on the right path. With such situations, the approach undertaken to make and accept estimates is adapted. The benefit from this practice is that the engineers get extra information and help on the task which the project manager or team leader thinks has been under estimated.

> Sometimes planning can be disruptive … do discussions, get consensus and communicate to everyone … estimation skills vary from person to person … some notoriously under estimate … some are uncertain and hesitant and over estimate to make sure they've got enough room … have to challenge them. (Director Software Engineering)

> Planning is continuous thing … keep on liaising with people … you have to take on a bigger responsibility … actually a lot harder and time consuming … selected individuals take part (Senior Engineer)

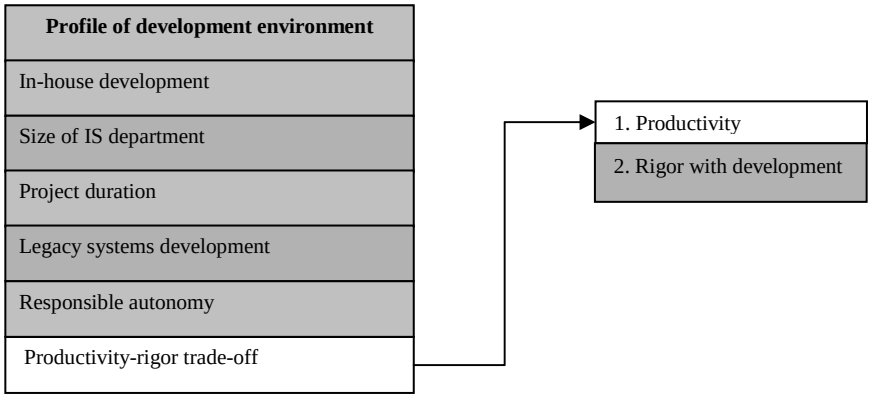
> Our weakest areas … it is your responsibility to estimate your own area … give examples as why come up with those estimates … breaking it down, present those to the team leader … re-negotiate if under estimated … always flawed … certainly haven't got to the point where our estimations are very accurate. (Senior Engineer)

Next, information is provided on the productivity and rigour trade-off at Meldevelopment. This is identified as an adaptation factor under the profile of the development environment in Fitzgerald's adaptation framework, as illustrated in Figure 53.

5.3.6 Productivity and rigour trade-off

The findings in relation to this adaptation factor are presented according to the themes that emerged from the analysis of data collected from Meldevelopment; productivity and rigor with development process. Productivity and rigour are equally important at Meldevelopment.

Figure 53 Productivity in relation to productivity-rigor trade-off – Meldevelopment



5.3.6.1 Productivity

Productivity at Meldevelopment means having the ability to get the most out of the available time for product development. Having a productive engineering department allows quick product releases in the market, ahead of the competition. A highly productive environment also enables them to deliver as much as possible with their limited resources.

> Compete with other companies and get to market early, a big advantage … quicker release, more competitive you are going to be … more productive, tackle more things that need to do. (Director Software Engineering)

> Be late to market you need to adapt the organisation to cope with that … cancel launches of the product, seminars and web casts … can realign and release like a Beta or an earlier release … still go ahead with the marketing launch to cope with the engineering slip. (Product Manager)

The following are some key practices for ensuring a productive agile development environment at Meldevelopment:

5.3.6.1.1 Effective iteration plan

Meldevelopment teams are required to have a good iteration plan, which must clearly indicate what needs to be done within one iteration release cycle before the teams can move on to the next one. This avoids engineers going overboard and gold-plating features that they are implementing.

> Never finish things off … getting this thing better and better and keep reworking it, when to draw the line … having a good plan. (Director Software Engineering)

> Spending a lot of time making requirements and the actual feature clear of what you actually have to do. (Senior Engineer)

5.3.6.1.2 Good project vision

Meldevelopment project managers are required to provide project direction by communicating the product plan thoroughly to ensure that a good project vision is established. This enables the team members to be on same page, helping to avoid uncertainty where engineers find it hard to do anything and also avoiding engineers heading off in the wrong direction.

> A lack of direction … people are procrastinating … do something but it is often in the wrong direction … project managers, work with them over an overall plan ... saying this is where we're going and this is what we're trying to achieve. (Director Software Engineering)

> Project managers have enough on their plate … managing a team, where they don't really have time to talk to as many customers as they would like and think about the vision as much. (Senior Engineer)

5.3.6.1.3 Good tool support

Good tool support is provided for engineers to make sure that they have everything needed to develop and test software in an intelligent manner. They have a strong belief that good tool support removes the overheads and barriers that impact the efficiency to carry out development tasks. For this reason, their development environment includes a powerful version control system, a multiple build system, printable whiteboards, and computers and projectors in the meeting rooms. A version control system is important to track code changes as a product comes together, and multiple build systems are important so that engineers know quickly of any breakage once code is committed to the version control system. With printable whiteboards, everyone is given printouts of what is discussed and agreed upon in meetings. They have access to computers and projectors in the meeting rooms, this enables them to make instant technical decisions.

> Version control is crucial … waiting for builds to happen, cost you … the printable whiteboards, a big productivity boost … have a session, at the end you print, a copy for everyone … computers and the projectors in the meeting rooms, a big productivity thing … make a decision on the spot. (Director Software Engineering)

> Build box is configured to do a build and run the tests and that's typically 20 minutes … which is really a bit long … really need to at least halve that … when I first got here it was a three-hour build. (Senior Engineer)

While these practices have been consistently applied at Meldevelopment, they have recently adapted a few other practices to make their agile development more productive. These are discussed next.

5.3.6.1.4 Adapting automated testing environment

The development engineers no longer write or update a comprehensive manual test procedure for every iteration release. Some of it took 20 to 40 pages, with exact instructions so that an individual who has never seen a feature before would be able to test it. The testing procedure required a complete set of steps, with frequent updates as a feature was implemented, adding a considerable overhead. The development teams felt the pressure towards the end of each iteration cycle to write a detailed test procedure. The senior engineers raised this issue in their project review meeting and they collectively agreed to provide just a brief test outline rather than detailed steps for executing a feature. In June 2004, this practice was adapted to provide a test outline only by the development engineers. This adaptation enabled Meldevelopment to continue implementing features in two week iteration cycles rather than increasing the duration of iteration cycle time to maintain a sustainable development pace.

> Have a summarised test design rather than a big procedure … it would just sink us … the overhead in it was just too much. (Director Software Engineering)
>
> For acceptance test our main role is to provide support material [test outline] … gives them more clarity. (Team Leader)

5.3.6.1.5 Adapting specification documentation practice

Meldevelopment maintains a history of all project specifications and documents all the changes that products go through. The specification highlights the different aspects of the product. When a product is enhanced with new features, the specification with all its history attached to it is reflected upon. However, reflecting all the past history of improvements is regarded by the engineers as not providing any substantial benefit in understanding and identifying new features to be implemented. In addition to this issue, it is a time consuming activity since most of Meldevelopment products have a history of enhancements since the early 1990s. Highlighted by the senior engineers the engineering department collectively agreed to adapt this practice to have a more effective product design phase. This product history reflection practice added the overhead to their design phase and this practice was adapted in December 2004 where the engineers no longer add new changes to the list of all the other changes with the original specification to maintain a product history. Rather, the specification is only updated to reflect all the current product features. As a result of this adaptation, engineers during the design phase immediately focus on new requirements after a brief reflection on the current features of the product.

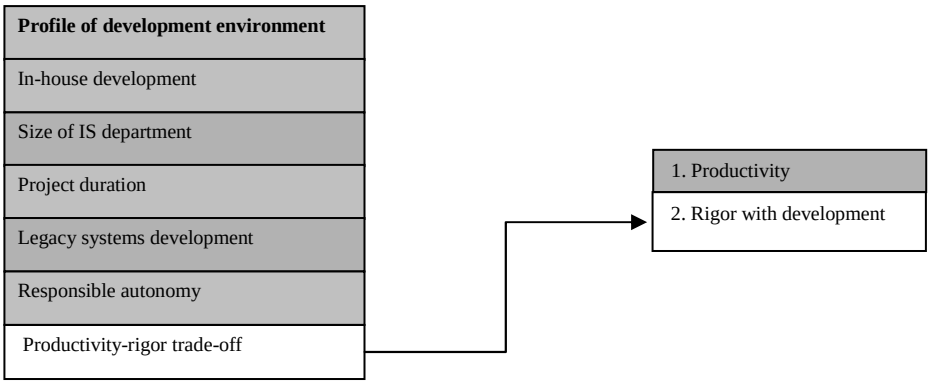
> Have a big archive for every spec that was written … change anything, had to go back and fully reflect it … dropped that a bit … reflect the product as it is today. (Director Software Engineering)
>
> All sit on meetings just to get the vision together … get a rough sort of skeleton. (Senior Engineer)

Next, information is provided on rigour with Meldevelopment's development process (Figure 54).

5.3.6.2 Rigour with development process

At Meldevelopment, rigour with their agile development is not compromised to achieve development team productivity. They have a rigorous agile process to ensure high quality products.

Figure 54 Rigor in relation to productivity-rigor trade off – Meldevelopment



Meldevelopment has a rigorous testing process with different types of tests in their agile development. This is achieved through adopting test driven development (TDD) within their automated testing environment providing speed, efficiency and accuracy.

Next, adaptation information is provided on the various types of test practices which they have adopted.

5.3.6.2.1 Automated testing environment

Prior to agile adoption, Meldevelopment had adopted TDD. However, they had created a limited automated testing framework using C++ for writing test cases, mostly for testing base classes. At Meldevelopment, base classes give a lot of common functionalities to their products. They also created a Perl infrastructure where some automated system test cases were written where a function call ran the test cases against the product. On adoption of agile approach in January 2003, the development team

decided to deploy NUnit for writing test cases for their unit tests with their TDD practice.

> Everything we do is really in NUnit… haven't discarded the Perl infrastructure … still runs … we don't really add a lot of test cases to it. (Director Software Engineering)

> Work very heavily at unit testing … do my code and try to considerably define completeness. (Senior Engineer)

5.3.6.2.2 Unit test

At Meldevelopment, the development engineers write unit tests for features that they implement. The normal procedure is to write a unit test before writing any code. Writing unit tests makes engineers think about how a feature will be used by their customers and how to eliminate a larger number of defects early. At Meldevelopment, implementing unit tests provide engineers with immediate feedback giving them the confidence when they lack expertise on a feature and encourage engineers to refactor code during implementation (making the code as efficient as possible).

> Avoids regression in the quality of code … encourages refactoring … a lot more confident when that's an area you haven't seen before … helps establishing the interfaces to your classes. (Project Manager)

Their TDD environment is implemented with a complete automated and continuous integration system. At Meldevelopment, the continuous integration system helps to detect and fix integration problems, gives an early warning of broken code and instantly evaluates unit test changes. This system also provides builds for quality assurance activities, product releases and marketing purposes. An automated development and testing environment has helped Meldevelopment to enhance their productivity by providing tools that make development much easier where implemented features are delivered in two week iteration cycles.

> Automated, continuous integration system … everything is completely automated from a build, array of build machines always building and end result is an actual CD that you could shift to a customer. (Director Software Engineering)

> The goal is that by the next iteration the feature that you delivered last time is in a good state … whatever is delivered into QA in one cycle, by the next cycle, which is 2 weeks late should have been tested [and] addressed all the issues … move on to the next iteration with zero defect policy. (Senior Engineer)

5.3.6.2.3 Adapting TDD

Meldevelopment's development environment has been adapted from heavyweight manual testing to an automated TDD environment. Prior to agile adoption a product release required weeks of manual developer level testing and over a period of time it got even worse. To overcome this problem, there was some limited TDD. However, the two systems that they had adopted were hampered by technological issues, which were experienced by their engineers. A collective decision was made by engineers to investigate for a more productive tool where they came up with the idea to adopt an open source testing framework. Then, some of the Meldevelopment new recruits had experience in using an open source framework. This played a big part in influencing the decision to adopt an open source framework rather than to purchase a commercial package. In February 2003, Meldevelopment adapted with NUnit, an open source unit testing framework for their Microsoft .NET development environment. This adaptation now makes it much easier for their engineers to implement unit tests to ensure high quality in making new features available in the shortest possible time.

> Hard to maintain and keep up-to-date (automated testing framework in C++ and Perl infrastructure) … there were issues … it is a lot easier with NUnit. (Director Software Engineering)

> Challenge to have tests in place to know that there are no defects … just the practicality of having those tests in place. (Team Leader)

In February 2006, Meldevelopment further adapted their agile TDD practice by incorporating brainstorming meetings between the development and the quality assurance teams about how a feature can be broken before its implementation. This TDD adaptation was achieved by co-locating a QA engineer in a sub-team's work environment (details of this adaptation is provided on page 218). This adaptation to co-locate a QA engineer with each sub-team for swift collaboration on tasks helps development engineers to understand the features to be implemented much better and faster, resulting in superior unit tests being implemented and higher quality features being delivered.

> Adapted test driven development … have QA, project manager and developers brain storming how they would break enhancement before [it] is built … that turns into unit test cases. (Director Software Engineering)

> Always think of all the different scenarios … having meetings not only with our actual development team but bringing testers in. (Engineer)

5.3.6.2.4 Adapting with manual testing

Despite having an automated TDD environment, manual testing is still important for their products. Some of the tests identified during the brainstorm meetings between the software and the quality assurance engineers cannot be implemented as unit tests. For such test cases, software engineers proposed that they must be manually tested by QA engineers with the respective builds. The engineering unit collectively agreed to this adaptation of their unit test to ensure that the high quality standards are being maintained with their implemented features. Since March 2006, test cases that warrant manual test are carried out by QA engineers using the iteration build. This is a further adaptation of their automated TDD environment.

> Some stuff you can't code well in test cases, but still got to check … short list of things that once you have built them, test manually. (Director Software Engineering)

> Automated tests just covers basic functionality and most of the time you might cover [most] of the issues … some functionality problems are performed with the manual testing. (Quality Assurance Engineer)

5.3.6.2.5 Adapting builds

Meldevelopment adapted their build system in July 2006 to have fast builds providing quick and more frequent responses for broken builds during project implementation. This build adaptation facilitates engineers to quickly find out a broken test to fix it immediately to deliver zero defect iteration builds. A sub-team highlighted that waiting for two hours for them to know of broken builds was a risk factor for the timely delivery of their high value project which they where undertaking. Team discussion involving their project manager and director of engineering led to increasing the hardware capacity providing a separate build system and adapting to involve only the relevant part of a product for build tests rather than the entire Meldevelopment product suite. With this adaptation, they have a faster build process where the engineers get a quick response when they commit their code to the version control system.

> Our XXX product sits on top of one core technology …takes two hours to build on an extremely fast machine … we adapted in XXX team, need to build a little bit of XXX … have a little 5 minute build machine … get more rapid feedback. (Director Software Engineering)

> Getting immediate responses when something is not working … every 5 minutes if possible we trigger a new build of specific areas … within 5 minutes, know if you have broken a test. (Team Leader)

5.3.6.2.6 Adapting acceptance test

In September 2006, Meldevelopment adapt their acceptance test to involve someone who works closely with their clients to carry out this test. Normally, acceptance test is done by QA team. Since their agile adoption in January 2003, they have very effective cross-functional collaboration at their design phase involving field staff to help them identify specific Low-level tasks associated with features. A decision was made by the director of software engineering and agreed by the engineering unit in a project review meeting to get a real world perspective to improve their "just" an engineering perspective on acceptance testing of the implemented features. Getting views from outside the engineering department enables them to enhance the quality of their highly strategic products.

> Have adapted, get customer interface people into the acceptance testing environment … independent of QA to try and give us a real independent perspective. (Director Software Engineering)
>
> Cutting the version of the product about a month before release, give it our field consultants to have a play with it. (Project Manager)

In November 2006, they have also adapted their acceptance tests to have the ability to handle changes halfway through an acceptance test. Normally, any change requires running all the tests again through the builds where developers reset the tests and re-run the tests from scratch. At Meldevelopment, the build process takes a long time due to an increasing number of automated test cases and a very large code base of their products. At times, these changes affect the dates for the market release of their products, impacting their sales and marketing activities. Highlighted by a product manager during a project review meeting, a decision was made by the director of software engineering to adapt the re-run of their automated tests after the changes are made during acceptance testing. This decision was collectively agreed by the engineering unit. They adapted it so that when changes are made, tests are re-run around that area only when builds are triggered. This adaptation enables engineers to swiftly fix and test changes, signing off releases on time.

> Adapting acceptance tests … have a week long acceptance test for a release … got a change half way through the test, retest around that area and continue through … time to market is a challenge, find quickest way of getting the release out. (Director Software Engineering)

5.3.6.2.7 Adapting systems testing

At Meldevelopment, some of the manual systems testing was adapted to automated testing in May 2004. Initially, the QA team did all the systems testing manually to

ensure that the products are compatible and integrate well with a list of different hardware and software platforms. Despite this automation of systems testing, QA engineers found it challenging to test the iteration builds on a larger number of different platforms in two week iteration cycles. As suggested by QA engineers and agreed by the engineering unit during a project review meeting, they adapted the systems test in November 2006 by separating the different configurations from the actual testing procedures. This enables testers to pick which ones to run on which configurations and to avoid duplicating it on all the configurations. This also allows quality assurance engineers to cover as much testing ground as possible in two week iteration cycles. Now, their releases are a lot more efficient.

> Had this massive configuration matrix to support Meldevelopment products … by simplifying the procedures, can run one in this configuration and another in a different configuration … give us more [coverage] and reduce the manual testing time. (Director Software Engineering)

> Lots of different configurations … a big distribution network of machines … there is quite a lot of complexity and just different combinations of that … quite a lot of effort that has to go into [systems testing]. (Quality Assurance Engineer)

5.4. Overt factors

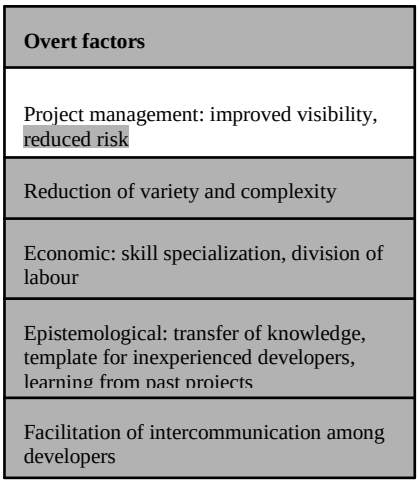
This section provides detailed information on Meldevelopment's agile approach adaptation driven by the overt factors or intellectual roles of the software development method based on Fitzgerald's adaptation framework (Figure 55). First, information on project management adaptation at Meldevelopment is provided.

5.4.1 Project management

This section provides adaptation information to improve visibility of the development projects at Meldevelopment, as illustrated in Figure 55. Improved visibility is part of project management as an adaptation (overt) factor in Fitzgerald's adaptation framework.

5.4.1.1 Improved visibility

Figure 55 Improved visibility (project management)-Meldevelopment



At Meldevelopment, both planning and executing projects are regarded as equally important. Their agile product planning task maximises the visibility of upcoming features; discussed are the product vision and iteration plans. Information on adaptation of these plans to improve visibility is also provided.

5.4.1.1.1 Product vision (high-level product plan)

Considerable effort goes into developing a high-level product plan at Meldevelopment. It is an important artefact used for establishing a product's vision company-wide before developments are given the nod. As a result, related development projects are identified. The product manager develops and communicates the vision plan company-wide by investigating the marketplace, and also by collaborating and deliberating with clients, sales and marketing department and engineering department to determine the features that will sell well. This strategy at Meldevelopment creates the visibility and awareness for upcoming features. At an engineering level, the focus changes to the execution of high-level plans by getting to understand what is to be built. This is achieved through their design phase. The engineering unit uses the design phase to identify low-level requirements from the high-level plan creating a release backlog.

> The product managers envision what we can build … very high-level … driven at a marketing level … the time it comes down to us (Engineering) … get the right people in front of the whiteboard … brainstorm and get an idea of what it is to be built … something that looks like a product. (Director Software Engineering)

> Product management says we want this feature for this release … engineers are involved in fleshing out those features … how big they are and how many things on that list we can actually get into the release … some bottom up request for features from the engineers to product management. (Team Leader)

5.4.1.1.2 Adapting product vision plans with prototypes

In November 2003, Meldevelopment adapted their product vision plans with prototypes. During a project review meeting, the engineering unit collectively agreed to develop prototypes to get feedback on proposed innovations. At Meldevelopment, it is difficult

to communicate the product vision based just on documents because their products are highly technical. Here, the vision plan was adapted to include prototypes to try to transform the high-level vision into something that looks like a product that individuals can interact with to gain a better understanding of the features. At the engineering level, prototypes create an awareness of the likely technical hurdles allowing the architects to think of potential architectural problems before the commencement of projects.

> Articulate the vision … prototype to make sure everybody understands … any technical hurdles, start to resolve … senior management can look at and understand what we are trying to communicate. (Director Software Engineering)
>
> Input that we have to get from all the stakeholders … he [product manager] will work with us early on to get prototypes of things that people need. (Senior engineer)

5.4.1.1.3 Iteration plan

At Meldevelopment, an iteration plan identifies the schedule and individual ownership of the tasks that are to be implemented in the next iteration. Iterations start after a design phase, once a sufficient prioritized product backlog is established. The priorities are assigned by the product manager, which are market-driven. At Meldevelopment, iteration planning meetings last no more than an hour and take place a few days before the start of an iteration cycle. Here, the iteration plan is an outcome based on agreement by development engineers on the individual ownership of tasks and their estimates with their project managers. Their QA function, documentation function and the marketing and sales department are all dependent upon the build from iterations. Therefore, the visibility of the iterations is extremely important. This visibility is dependent upon the reliability of the task estimates provided by the engineers in an iteration plan.

> Established a feature list [product backlog] …work out rough estimates … put together an iteration plan … based on the numbers that the engineers have come up with for their estimates … work out what the schedule will look like. (Project Manager)
>
> Sometimes it can be the Project Manager … sometimes it will be the engineers [putting the iteration schedule] … you want the developers to be involved in that. (Senior Engineer)

5.4.1.1.4 Adapting task estimating practice

At Meldevelopment, there are three points for estimation. One is with the roadmap, estimating on a large scale for the next 6 to 12 months for the releases Meldevelopment will do. This is a high-level estimation, providing an indication of the likely duration of feature implementation. At this level of estimation only a few senior engineers are involved. Next, estimating at the design phase determines the number of iterations needed to implement a feature. The design phase requires all the engineers on a project

to participate in analyzing the high-level features into individual independent tasks and providing estimates for each. Then at a micro-level for iteration planning, the engineers evaluate and confirm estimates for tasks against which they will work. This re-estimating practice enables the team to have a detailed understanding of design decisions and implementation strategies.

> Engineers are focused on finalisation of the product [so] a small number of people involved in the high-level [estimation]… doing design, we have to make the commitment … need every person … knowing that they can deliver against it. (Project Manager)
>
> Sometimes things will be slower than you expect … we will adjust a lot of the other estimates to say that it is not accurate … then adjust the schedule to compensate for that. (Senior Engineer)

Despite having three different points for estimation, it is a constant challenge for their development teams to deliver against the schedule. On occasions, to deliver on a certain date requires crowding out some of the tasks. Other times, iterations have more than the accepted level of bugs due to insufficient time spent properly implementing tasks. These are mostly due to estimation errors resulting from what the engineers thought of at that time. As a result, they adapted their agile estimation practice in January 2006. This adapted practice now involves not revealing estimates until all engineers have their own estimates ready to reveal to the team. As a result, the engineers have experienced more team discussion on tasks to understand design decisions and implementation strategies. The previous practice was based on someone suggesting an estimate and everyone mostly agreeing to it. In September 2006, the engineering unit collectively agreed in a project review meeting to adapt their agile estimation practice by introducing a process around iterations to capture, track and evaluate iteration performance. This was a result of the team trying to improve their task estimation skills since some engineers were consistently providing estimates which were lower than what it took them to implement those tasks.

> Estimating in group, not showing until they are all done … reveal their number at the same time … not influenced by other person … a lot of differences, it prompts discussion … adapted formal levels … record what we do, how our estimates compare … how many bugs we end up with. (Project Manager)
>
> Working on the estimation process … one of our weakest areas … hopefully one of the technique we use will [enable] to get the best estimates … most developers and I am included, tend to take on more than can handle … want the project to be done as quickly as possible. (Senior Engineer)

Next, adaptation information to reduce risks in relation to project management at Meldevelopment is provided (Figure 56). Reduced risk is part of project management as an adaptation (overt) factor in Fitzgerald's adaptation framework.

5.4.1.2 Reduced risk

Figure 56 Reduced risks (project management) – Meldevelopment

Overt factors
Project management: improved visibility, reduced risk
Reduction of variety and complexity
Economic: skill specialization, division of labour
Epistemological: transfer of knowledge, template for inexperienced developers, learning from past projects
Facilitation of intercommunication among developers

Discussed next are some of the key risk factors that may impact Meldevelopment's new development. Information is provided on adaptation of their agile approach with new practices to minimise the impact of these risk factors.

5.4.1.2.1 Unclear project scope

In September 2003, Meldevelopment adapted their product backlog planning practice with "sufficient enough" planning to establish a stable product vision and product release dates. This adaptation decision was collectively made by the engineering unit during a project review meeting. Due to unclear project scope, engineers in a team implemented product features that were not exactly what was expected by the product management. When this was discovered later in the project time line by the product manager, the project had incurred cost and schedule impacts. At Meldevelopment, due to the highly technical nature of their innovations this risk factor emerges with projects despite having the flexibility with their agile approach to embrace change as the market requirements change. As a result of this adaptation, engineers are able to plan a reliable product backlog for implementation.

> Planning, to understand the scope … starting to run before you know where you're running … happened in the past … senior management didn't have a idea of where they wanted to go … start building, couple of months later … actually meant to do other stuff … vision hasn't been stable. (Project Manager)

> There was a period a while ago when we were going backwards and forwards on a couple of issues on [a project]. (Senior Engineer)

5.4.1.2.2 Product conceptualization: quality Issues

Since November 2003, Meldevelopment made the involvement of external stakeholders such as the product manager imperative during development. Initially, they were only involved with creating, prioritising and scheduling the product backlog for

implementation. However, there was always a risk that some aspect of a new innovation they were implementing could become a major quality issue. Engineers discovered that the product manager's direct involvement during iteration cycles in a project helped them to deliver much higher quality with the new features. The engineering unit collectively agreed to involve the product manager as much as possible during iterations when he is not working in the field. Now, there is a mutual understanding that the external stakeholder contribution extends beyond planning to bring their unique perspectives into development which engineers often do not have.

> The most successful projects, not just in planning but the whole projects, is when we've got stakeholder influence. Product manager had a little play around with product … had so many bugs, it wasn't functionally broken but it wasn't telling a story. (Project Manager)
>
> Think from the customer point of view … product manager often does sit down at various points in the project and have a play with things … sometimes it happens a bit late for our liking. (Senior Engineer)

5.4.1.2.3 Unreliable schedule: inaccurate estimates

Meldevelopment employ an "estimation based on knowledge" practice for task estimation to plan and schedule development work. Their belief is that the engineers have the right to tell how long it will take to build a feature. Trusting and working with engineers are the two most important aspects of Meldevelopment's agile culture, which drive project management activities. However, project managers discovered that engineers were under-estimating tasks. When tasks are not implemented as estimated it impacts their development and business schedules. Since September 2004, the project managers employ a re-negotiation practice to add a buffer to the estimates when they suspect under-estimates. The engineer empowerment to provide estimates was adapted with re-negotiation with the team leader or project manager. This was raised in their project review meeting by the project managers and the engineering unit agreed to a re-negotiation practice. The benefit from this adaptation is that the engineers get help on the task which the project manager or team leader thinks has been underestimated.

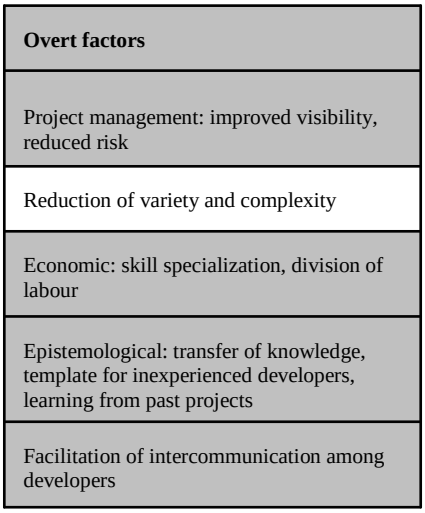
> They have the right to tell us how long it will take to build … have to filter; some people always underestimate … just a personality thing … double his numbers. (Project Manager)
>
> Different opinions on how long something is going to take … can't agree on something … break them down … present those to the team Leader. (Senior Engineer)

Next, adaptation information is provided on reduction of variety and complexity with Meldevelopment's agile approach. It is identified as an adaptation factor in Fitzgerald's adaptation framework (Figure 57).

5.4.2 Reduction of variety and complexity

This section provides adaptation information on release and daily plans at Meldevelopment.

Figure 57 Reduction of variety and complexity – Meldevelopment



5.4.2.1 Roadmap plan; release and daily plans

In conjunction with Meldevelopment's vision planning, their product roadmap planning is also undertaken. Here, the roadmap plan involves identifying the main features, their sizes, and the quarter releases in which the features will be included. This plan usually covers a roadmap for a product for up to eighteen months. Reduction of variety and complexity in Meldevelopment's agile product development is achieved through their short development cycles. Their release dates and iterations are examples of short development cycles that have a set of targets to meet. Based on a roadmap, a release plan is done for a quarter. This plan includes the creation of a backlog through a design phase with low-level features (tasks) which are implemented on their own. At Meldevelopment, the design phase provides the reduction in variety and complexity of the high-level features.

> Roadmap … define the steps, what we need to do in order to get to where we want to be … have a product requirements document … a snap shot of next release … a very high-level document … looking at delivery dates … engineers flesh out, put it together [release backlog] … do some prototyping and design work … at a point engineers give us numbers [estimates]. (Project Manger)

The other important aspect of a release plan is that it identifies all the iteration cycles leading to a specific release date. At Meldevelopment, the reduction in variety and complexity of a backlog is achieved through iteration cycles. At a micro-level, iteration cycles provide a further reduction in variety and complexity through daily plans.

> Define what we are going to deliver, break into iterations. Each tiny bit is complete and useful functionality, as a story to describe project. The whole team [participates] … numbers go into the schedule … put together the iteration plan of what's going to be deliver in each iteration. (Project Manger)

> I think as a developer … say this is how long something is going to take to do … it is breaking it down into some database work … there is some business logic we need to work on … need to put best and worst likely times … we break them down. (Senior Engineer)

5.4.2.2 Adapting release plans

Early in agile adoption, Meldevelopment had experienced delays in releases at the end of each quarter when several products were involved. Each release has to be certified with all the required Meldevelopment quality assurance activities and to do these tasks for several products all at the same time often overwhelmed the QA team. The director of software engineering in discussion with the QA team proposed at a project review meeting to adapt their roadmap plan to have release dates of different products with sufficient time gap between them so that each product is properly tested without overwhelming the QA team with different products at the same time. This suggestion was accepted by the engineering team, requiring them to adapt how they plan their release dates with the roadmap plan. In September 2003, release plans were adapted to synchronize between the releases to achieve better productivity and predictability in their schedules.

> Release plan had to adapt … building 5 different products … make sure that QA wasn't getting swamped at any particular time. (Director Software Engineering)

> Very dynamic … sometimes it happens where we will have 3 products come into QA concurrently … we target, they give us 1. (Quality Assurance Engineer)

5.4.2.3 Adapting daily plans

Since January 2003, the daily team meetings were held to provide for development task status updates by engineers. Their stand-up meetings have an informal structure. These meetings are short but without being time-boxed and are not held at the same time each day. However, the engineers experienced not much value when they just provided task status as they are frequently aware of one another's progress since they all are co-located in an open team space. During a project review meeting, the senior engineers requested to adapt the stand-up meetings to identify and have quick discussions around the development issues engineers faced during iterations. In September 2004, the engineering unit agreed to this adaptation of their daily stand-up meetings. However, some teams realised that not much value was achieved in having separate daily stand-up

meetings since engineers discussed issues in their sub-teams as soon as they encountered them. This was raised in their project review meeting by the engineers and the engineering unit agreed to adapt the stand-up meeting practice to an as-need basis. Since November 2005, the daily stand-up meeting practice is adapted based on the nature of the projects. Now at Meldevelopment, some projects have daily stand-ups, while other teams adapt it to weekly stand-ups, depending upon the stage and complexity of their project.

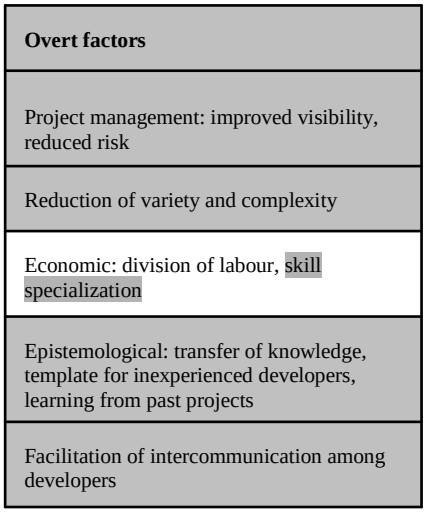
> Done it informally in teams, without having a set time … everyone say what they've been working on … didn't seem to be effective … then tried, what issues do we have … have a quick discussion, how to solve it … at the moment, sometimes we don't depending on how well the project is tracking. (Director Software Engineering)
>
> We did for a time had dailies … found good … know what everyone was up to every day … areas were really segmented and not generally conflicting, it didn't matter what anyone else was saying … so we stopped it after a while. (Senior Engineer)

Next, adaptation information is provided on division of labour and skill specialisation with Meldevelopment's agile approach (Figure 58).

5.4.3 Economic: division of labour and skill specialisation

First, adaptation information is provided on Meldevelopment's division of labour with their agile approach.

Figure 58 Division of labour – Meldevelopment



5.4.3.1 Division of labour

Meldevelopment's engineering department is split into three functional areas; development, quality assurance and documentation. Each of these functional units has a separate role in their agile development process. At Meldevelopment, this functional setup has continuously brought development success. While Meldevelopment maintains a clear division of labour within its engineering department, they work as a team to deliver products for their market releases. With their agile development, they do not have specialist roles within their functional teams, expecting individuals to be able to do all the tasks in their teams. Hence, there is no division of labour within each functional team.

> Development teams … separate QA team … technical writers (documentations team)… structure hasn't changed … interaction within the teams has changed significantly. (Director Software Engineering)
>
> Development teams will often go to a documentation person before they put the text on the screen just make sure that the documentation person is happy … likewise, ask the QA person for their advice, more from the user perspective. (Engineer)

5.4.3.1.1 Adapting with business contribution

Since January 2003, their agile development included a contribution from their product manager. However, this business role contribution was based on mutual understanding to help the development function with the vital customer or business related information on where and how the proposed features are likely to be used. Hence, the product manager was the most critical source for the engineers to understand the high-level requirements (market and specific customer needs). The product manager's involvement in the design phase helped the engineering unit to swiftly establish backlogs for implementation. The product manager testing during iterations provided engineers with quick feedback on the quality of the implemented features. In January 2007, project manager roles were adapted to product development manager roles (now under the sales and marketing department but co-located with the engineering unit), making business contribution permanent at development level. The engineering unit collectively made the decision to request senior company executives for project manager roles to be adapted to incorporate product management functions since their product manager was leaving the company. Due to their highly technical products, they required company experienced individuals to provide them with the domain support. At Meldevelopment, the product manager worked closely with project managers, making project managers most suitable for this new role. This product development manager role now provides engineers with an effective domain support, since project managers in this role also work in the field and constantly liaise with other field staff to carry out product management tasks. With this adaptation, their agile development now has business and engineering functions working together on a daily basis. At the development level, there is a division of labour whilst working together as a team. Their projects require a full commitment from their engineers and the product development managers.

> [Project managers] adapted to manage product planning … responsibility for product direction. (Director Software Engineering)
>
> The project manager's ability to understand the requirement roles of product management … need to ensure that their engineers are implementing based on the priority. (Product Manager)

Next, adaptation information is provided on skill specialisation (Figure 59).

5.4.3.2 Skill specialisation

Figure 59 Skills specialisation – Meldevelopment

Overt factors
Project management: improved visibility, reduced risk
Reduction of variety and complexity
Economic: division of labour, skill specialization
Epistemological: transfer of knowledge, template for inexperienced developers, learning from past projects
Facilitation of intercommunication among developers

Meldevelopment engineers were always required to acquire broad engineering skills along with their specialist skill sets. The engineers now are also expected to undertake all of the related tasks that their functional teams are responsible for in product development. For development teams, this ability to undertake all development tasks by the software engineers is most critical to foster a team effort and commitment for development and delivery of features in short development cycles. Hence, this generalist engineer skill set is critical to be able to share tasks with other engineers to enable team effort for regular delivery of features in short cycles. Even prior to agile adoption, they had particularly focused on software engineers acquiring a broad development skill set rather than specializing only in a certain role because of their frequent market release of products.

> Mostly generalist … don't have specialist in different areas … have a bunch of like minded, broadly skilled but also quite specialised in several areas … a team structure that has worked for us. (Director Software Engineering)

> Have highly skilled people … have extremely good judgement … it is about making the right decisions for the business … shouldn't have divisions … everyone should be involved in all work. (Team Leader)

5.4.3.2.1 Adapting development engineers generalist skill sets

Meldevelopment engineers adapt their skills to remain on the leading edge of technology development. One of the main reasons for adopting an agile approach was to foster a team approach involving the field staff with software engineers to swiftly identify what exactly was to be built. Another key reason was that software engineers were required to have regular discussions among themselves, showing and getting feedback on what each of them was implementing. Such interaction was critical for them to determine that they were on the right track because of highly technical innovations they were implementing. Prior to their agile adoption, the senior engineers

together with the director of software engineering identified interaction as a critical skill for software engineers at Meldevelopment. On agile adoption, engineers adapted with interpersonal skills to achieve effective and efficient interaction and collaboration with the stakeholders. This decision to adapt with interpersonal skills was made by the by the director of software engineering. However, a few months into their agile adoption a couple of the engineers left the company since this work style did not suit them. Therefore, the hiring process was adapted in April 2003 to determine a prospective engineer's ability to work in a team setup. The software engineers also adapted their technical skills to include design, usability engineering, and planning skills. These skills are required for the design phase of their agile process. For implementing product features, they also adapted their technical ability to include refactoring and test driven development skills. This skill adaptation was collectively agreed by the engineers on adoption of the agile approach. These new skills for software engineers were determined to be critical since the engineering unit had made a decision to work in short development cycles on projects. With this adaptation, engineers now have the ability to implement most of the tasks in a product backlog including tasks associated with their agile development.

> Adapt requirements and design [skills], do everything on the fly with white board discussions … coding, adapted with refactoring … test driven development, adaptation was because of quality … something that hangs together rather than long defect cycles. (Director Software Engineering)

> Besides coding … responsible for the requirements review designing… unit testing … system testing … technical investigations … responsible for support … sometimes they will have to go on site for implementation. (Senior Engineer)

5.4.3.2.2 Adapting the QA role

In February 2006, Meldevelopment focused on making the development and quality assurance functions even more productive by adapting the QA role into an agile tester role (details on this role adaptation are provided on page 232). This is achieved by outsourcing some of the repetitive manual systems testing functions and freeing the QA engineers to work more closely with the development engineers. The QA engineers now contribute more to unit tests and eliciting the low-level requirements. With this adaptation, the QA engineers now make an end-to-end contribution to develop better quality products. Software engineers also get help in implementing better unit tests.

> Offshoring manual testing … get done some of the mundane work and free up the current QA people … provide insights to developers to make the code better. (Director Software Engineering)

> From an engineer's point of view, we would like to have dedicated QA member … for the first time ever, we have a QA member actually embedded in our team. (Senior Engineer)

5.4.3.2.3 Adapting documentation role

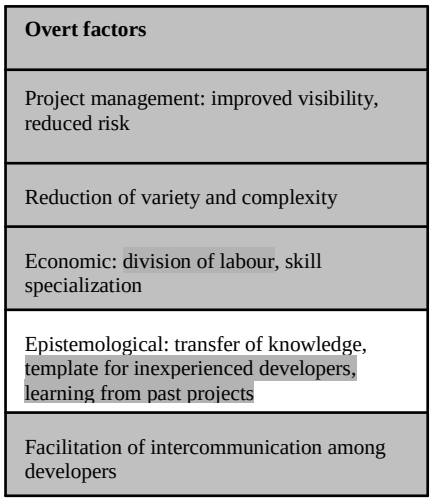
In December 2004, the engineering department collectively agreed to adapt their design phase with the User Centred Design (UCD) approach. This adaptation required the technical communication engineers to adapt their documentation skill sets to include usability engineering skills, especially knowledge of the UCD approach. The usability skills of documentation team now enable them to run usability tests to learn directly from their customers on how they are going to use their product and the likely difficulties that they may face. This learning helps them to determine the appropriate manual support to provide for their products. This learning also enables the documentation team to provide data to their engineering unit to enhance their products with improvements in usability and user experience. The following information demonstrates the usability engineering work that the documentation team performs in their agile development environment.

> Usability session, guy from a bank … [gave] a set of scenarios to work through … facilitated videotaping, did screen capturing … analysis of what worked and what he found difficult … fed that into the product design. (Senior Technical Communication Engineer)

Next, information is provided on the epistemological role of methodology with Meldevelopment's agile process (Figure 60)

5.4.4 Epistemological role of methodology

Figure 60 Transfer of knowledge – Meldevelopment



Meldevelopment has two organisational practices that the company has held since its inception as a software vendor; mentoring and post-mortem practices. The mentoring of graduate recruits by senior engineers for transfer of development knowledge and the conducting of post-mortems for major releases has always been a part of the development process. The epistemological role of Meldevelopment's agile method is presented using its sub-factors; the template for inexperienced developers, transfer of knowledge, and learning from past projects. First, adaptation information relating to transfer of knowledge is provided (Figure 60).

5.4.4.1 Adapting with mentoring practice for transfer of knowledge

With agile adoption in January 2003, the team philosophy was also expected to bring benefits for engineers where they learn and develop new skills and experiences through task sharing. To enhance the agile learning experience, the engineering unit collectively agreed to adapt the learning practice through task sharing with a mentoring practice. This decision was made immediately after agile adoption, since they had graduate engineers who had been mentored by their senior engineers. This suggestion was proposed by the senior engineers based on their mentoring experiences with their previous approach. With this adaptation in February 2003, their graduate recruitment programme for agile development was backed by a strong one-to-one mentoring and coaching policy to up-skill their graduate engineers by providing them with useful development experiences. For the mentoring roles at Meldevelopment, an engineer is required to have substantial software development experience with the company. Experienced engineers continuously transfer knowledge, nurture and provide insights into different development situations for inexperienced engineers.

> Graduates have mentoring … shaped to do development … our best guys have been mentored by some of the previous best guys … the better your mentor, better is coaching. (Director Software Engineering)

> A new guy comes in … a senior engineer will sit next to him … go through the process, how we do things, who to approach, and what you should do ... 6 months or a year, they will sit next to each other. (Engineer)

Mentoring and coaching is a strategy to ensure that the levels of necessary development skills and experience are maintained, minimising impact on development capabilities as a result of the departure of engineers. This practice helps Meldevelopment to maintain its quality and to ensure distinction in its delivery of software products.

> Have two people for each product … after six months everyone has to move onto something else … we have good hand over … can leave yourself exposed. (Director Software Engineering)

Next, adaptation information relating to the template for inexperience developers is provided (Figure 61).

5.4.4.2 Adapting the template for inexperience developers

Figure 61 Template for inexperienced developers – Meldevelopment

Overt factors
Project management: improved visibility, reduced risk
Reduction of variety and complexity
Economic: division of labour, skill specialization
Epistemological: transfer of knowledge, template for inexperienced developers, learning from past projects
Facilitation of intercommunication among developers

The new recruits are immediately assigned to development teams and are included in all aspects of the work assigned to teams. Meldevelopment has adopted on-the-job training as a key approach to provide development training for new recruits to become proficient engineers. Here, it takes up to two years for a graduate engineer to become a skilled engineer. However, they adapted their agile approach with formal training sessions for all new recruits in August 2003, involving delivering product training sessions. This suggestion was made by director of software engineering in a project review meeting and agreed by the engineering unit. At that stage, Meldevelopment was hiring new recruits and to help them ease into development teams this formal training was proposed. This adaptation minimised the impact on senior engineers own work when providing support and helped the new recruits to get familiar with their products and development environment during the first month of their employment at Meldevelopment. Through this training, new recruits become familiar with their products, making their learning much easier when they are assigned to development teams.

> Put them on a formal training course that we deliver to customers about how to use the products … understand the products from the outside and what we're about. (Director Software Engineering)
>
> Learning was much easier … training provided some understanding of the products the company was selling. (Engineer)

Next, adaptation information relating to post mortems is provided (Figure 62).

5.4.4.3 Adapting to learn from past projects

On their agile adoption in January 2003, the engineering unit collectively agreed to adapt their agile approach to include a reflection meeting after a release to learn about the development issues encountered during a release cycle and to drive forward for the next release cycle armed with lessons learned from the past. This was Meldevelopment's adaptation of the agile approach requiring a regular reflection meeting for each iteration cycle. Having a regular reflection meeting every two or three weeks was seen as not feasible by the senior engineers at Meldevelopment. Hence, as suggested by senior engineers they collectively agreed to adapt to have a reflection meeting after every release just as they had with their previous development approach. However, the engineering unit had also agreed to have reflection sessions every two weeks or weekly on highly complex projects. Usually, their engineering teams have two reflection meetings for a release, with the second one being a follow-up meeting to decide the actions to take to resolve the issues that arise in the first one.

Figure 62 Learning from past projects – Meldevelopment

> Always have one after release… encourage people to brainstorm issues beforehand … people raise their items, put them on a whiteboard, go through them … weigh them … collate and look at the results … bring everyone back together … decide actions to go forward. (Team Leader)

Next, adaptation information is provided on facilitation of intercommunication among developers. It is identified as the final overt adaptation factor in Fitzgerald's adaptation framework (Figure 63).

5.4.5 Facilitation of intercommunication among developers

Figure 63 Facilitation of intercommunication among developers-Meldevelopment

Overt factors
Project management: improved visibility, reduced risk
Reduction of variety and complexity
Economic: division of labour, skill specialization
Epistemological: transfer of knowledge, template for inexperienced developers, learning from past projects
Facilitation of intercommunication among developers

Meldevelopment provides an open workspace for their development teams. Since adopting the agile approach, the engineers have all been co-located in a single open workspace. Their management team, the product development managers and the director of software engineering are co-located. At Meldevelopment, this co-location practice was adopted to encourage spontaneous collaboration between individuals. The quality of their team work is dependant upon this collaboration within their development function and also critical to deliver their strategic products. For this reason, an open workspace practice is adopted as part of their agile approach to break down barriers to effective communication and to encourage the engineers to continuously collaborate with one another during development.

> Open space is very important … teams are facing each other … the communication, little things you overhear, as you work day-to-day, seems crucial … through collaboration, we get a product right the first time rather than having to take a second bite at it. (Director Software Engineering)
>
> Open team space … feel more like you are part of a team … communication a lot easier … interact and talk to people and discuss ideas … turn sideways and you talk to someone about issues. (Senior Engineer)

5.4.5.1 Adapting with partitions

In their open workspace each team is separated with a partition and has own workspace or team work area. The proposal to partition the open workspace was made by the senior engineers immediately after their agile adoption and agreed by the engineering unit. In February 2003, their open workspace was adapted with partitions. High noise levels were experienced by engineers because of the constant collaboration happening in sub-teams, since they were co-located next to each other. Close co-location caused disruption for engineers in focusing on what they were doing collectively or

individually in their sub-teams. The partitions minimise noise reaching the other teams. Teams on the same project are placed next to each other so that they can communicate or hear each other over the partition, and easily walk into another's workspace for a discussion.

> Partition, some of the noise of the other areas is reduced … discussion can get too loud, to the point that people are getting distracted. (Director Software Engineering)
>
> If there are some other things going on, it can be a little bit distracting … but most of the time I think it is a positive thing. (Engineer)

5.4.5.2 Adapting with whiteboards

At Meldevelopment, whiteboards are regarded as an important tool for facilitating effective interaction amongst the engineers. Since February 2003, each engineer is provided with a whiteboard in his team workspace, where he is located. This decision to provide engineers with a whiteboard was made by the director of software engineering after the partition decision to encourage interaction between engineers. Whiteboards are also provided in the corridor, meeting rooms and in the kitchen. On suggestion by the director of software engineering, the engineers collectively agreed to have meetings and discussions requiring a long period of time away from the team workspace, minimising the noise levels for others. With these adaptations, their open workspace has become a highly effective tool for software engineers, enabling them efficient interaction and productive team effort.

> Feel comfortable … enough whiteboard space to be able to draw things up … you spend a bit of time in there … want it to be as good as you can make it. (Director Software Engineering)
>
> People are generally quite considerate … going to be having a big discussion they will take it into a meeting room … most people have a set of headphones have music to drown them out. (Engineer)

5.4.5.3 Proximity of resource

The resources such as meeting rooms are all located in close proximity and on the same level as their team workspace. The close proximity of meeting rooms is regarded as a contributing factor for on-going interactions, which is very strongly encouraged in their development environment. Meldevelopment provides three such meeting rooms with all the necessary equipment and facilities, including phone conference capabilities. These facilities are shared and are readily available for engineers to use but need to be booked for large team or customer meetings.

> No meeting room within a distance, people will go okay maybe it is not worth it or they just don't really meet … make a lot harder to make that happen. (Director Software Engineering)

> I can't be bothered walking all the way over to the other side of the building to have a meeting… it [meeting rooms] really improves the communication. (Engineer)

5.5 Covert factors

This section provides detailed information on covert factors or political roles of software development methods (Figure 64). First, adaptation information on the comfort factor is provided.

5.5.1 Comfort factor

Meldevelopment's agile approach has established practices to undertake projects (documented with organisational factors and overt roles). These agile practices are also embraced to ease the difficulties and pressures of their market-driven product development. They enable engineers to handle stressful situations such as dealing with tight schedules, meeting fast approaching deadlines, identifying business value features, and meeting specific quality standards. Next, adaptation information is provided on another practice which helps to further provide a comfort factor for their engineers.

Figure 64 Comfort factor – Meldevelopment

Covert factors
Comfort factor
Legitimacy factor
Aura of professionalism
Confidence factor
Audit trail
Raise profile of IS department

(i) Sustainable velocity: 40 hours per week

Meldevelopment adapted their agile development approach to include a 40 hour week per engineer on adoption of an agile approach in January 2003. This was proposed by the senior engineers and collectively agreed by the engineering unit based on their experience of working long hours and weekends to meet the release commitments. At Meldevelopment, this practice minimises burn-outs and maintains a consistent development velocity to ensure high productivity for projects. A 40 hour working week has helped them to avoid engineers regularly working late hours and weekends to meet deadlines. In trying to keep ahead of the competition, such practices help to minimise the chance of setting unreasonable project schedules and deadlines. However, Meldevelopment engineers have a mutual understanding to stay back as a team if any extra bit of effort is required to meet deadlines.

> Guideline is 40 hours a week … never tell someone to do extra work … do it as a team … work overtime, counteract to get a quality result … get burnt out, keep it in business hours … work at a sustained pace… knowing that its under control. (Director Software Engineering)

> Stay back late on Wednesday and Thursday, and you are going to work Saturday … that sort of thing doesn't work. (Senior Engineer)

Next, information is provided on the legitimacy factor (Figure 65).

5.5.2 Legitimacy factor

Figure 65 Legitimacy factor – Meldevelopment

There is a mutual acceptance between the engineering and business functions at Meldevelopment that their agile approach provides them with a legitimate method for achieving successful product development. Iterations and demonstration releases as part of their agile approach have a significant impact on Meldevelopment's business function. The product management team formulates a better product strategy. (Information on these aspects of their agile approach, including their adaptations is provided earlier in the section on organisational factors and overt factors). The iterative approach enables the engineering team to provide implemented features to the product management team on a regular basis to check for the business value. The development teams are also able to build demonstration releases for the product management team. A demonstration release of their features helps them to get effective feedback from potential customers. Quality feedback was not possible with their previous approach, which as based just on written descriptions of the product features. From the company management perspective, they are meeting their business objectives through agile development.

> With RUP … projects took a very long time to deliver … didn't always hit the requirements of what people liked or the market was asking … didn't get to see anything until it was too late to make any changes … using an iterative approach… demonstration releases, get feedback, something that they could use … whether they would buy product, how much they would be willing to pay. (Product Manager)

Next, information is provided on aura of professionalism (Figure 66).

5.5.3 Aura of professionalism

An aura of professionalism surrounding development work at Meldevelopment is achieved through engineer empowerment and trust. Through these two beliefs at

Meldevelopment, a common ground between the management and development team is achieved for all the activities associated with product development. Engineers are empowered to make decisions on the adoption and adaptation of their development process and tools. Such empowerment does not burden their development teams with process and tools that engineers find to be counterproductive for achieving the development goals.

Figure 66 Aura of professionalism – Meldevelopment

Covert factors
Comfort factor
Legitimacy factor
Aura of professionalism
Confidence factor
Audit trail
Raise profile of IS department

Being able to negotiate features and accept the responsibilities for planning projects enables the Meldevelopment engineers to avoid unreasonable requests in relation to product development. (Information on engineering empowerment and on its adaptation is provided with the profile of development environment factor).

> See things in shades of grey rather than black and white, compromises can be made … developing units have got a lot of power … within the company you feel like that they all rely on you … it is great to be empowered … comes great responsibility. (Principal Engineer)

Next, information is provided on the confidence factor (Figure 67).

5.5.4 Confidence factor

Figure 67 Confidence factor – Meldevelopment.

Covert factors
Comfort factor
Legitimacy factor
Aura of professionalism
Confidence factor
Audit trail
Raise profile of IS department

Meldevelopment's agile approach provides a systematic product development process which enables the product management and engineering teams to justify a development project. Their agile approach allow them to identify business value products by formulating vision plans to release, support and grow their product capabilities in the marketplace. Executive management have a significant level of confidence in their engineering ability since they continuously implement products that are very successful. This is achieved through building cohesive and high performing development teams. (Adaptation information on development teams is provided with in the size of IS department factor).

> Whatever is thrown at us we will find a way through …we will adapt … gives me the confidence in a lot of our projects … I trust the team because I know that we will find a way … as a manager that gives you a lot of confidence that these people are going to succeed regardless of exactly how we do it. (Director Software Engineering)

> You spend a lot of time with the team … you form relationships … learned to trust and rely on each other and you know that as you go through each project and you have successes and you can build on those … it gives you confidence. (Engineer)

5.5.4.1 Cohesive and high performing teams

Meldevelopment delivers visionary products, which are also driven by short term customer needs or a large customer that has particular needs. The engineering teams quickly adapt to grab such business opportunities. Another aspect is that Meldevelopment continually broadens its product features. To be successful in such business environment, Meldevelopment requires not only cohesive but also high performing teams to deliver quality products on time.

> Cohesive teams … fundamental to the success … not able to work well, issues be meeting deadlines and things blowing out … somebody is struggling on an area… important is being able to communicate ideas. (Senior Engineer)

Meldevelopment ensures that they have high performing development teams through encouraging and providing the means for participative leadership, shared responsibilities and a communicative environment where people interact and collaborate all the time. They also achieve this by employing creative and talented individuals, those who are able to respond rapidly to meet changing business needs.

> Have well rounded people … very talented, high performing … very focused on team work … group of young, energetic, passionate people who have done whatever it takes to meet goals and objectives … seen as achievers and successful engineers. (Director Software Engineering)

> Have a very good team here … engineers are very friendly … willing to do what it takes to get the job done … we can always do with engineers having a better understanding of the requirements in the market and what customers are asking for … a skill that doesn't exist. (Product Manager)

Next, information is provided on the audit trail **(**Figure 68).

5.5.5 Audit trail

Figure 68 Audit trail - Meldevelopment

Covert factors
Comfort factor
Legitimacy factor
Aura of professionalism
Confidence factor
Audit trail
Raise profile of IS department

The audit trails at Meldevelopment enforced quality in their previous development approach. In order to achieve a quick and regular delivery of product features, their audit trail practice of core review was discarded. It was compensated for by their agile practices such as code reviews, whiteboard discussions between development and quality assurance engineers, and TDD. In addition, Management trusts that the development engineers will follow their expected agile development practices to ensure that product quality is maintained all the time. However, most recently, with hiring of new engineers their agile approach is adapted with some core reviews. This has been documented in the quality assurance function section under organisation factors.

Next, information is provided on the adaptation factor, raise the profile of IS department **(**Figure 69).

5.5.6 Raise the profile of IS department

Figure 69 Raise profile of IS department - Meldevelopment

Covert factors
Comfort factor
Legitimacy factor
Aura of professionalism
Confidence factor
Audit trail
Raise profile of IS department

Adopting the agile approach has raised the engineering team’s profile, resulting in the entire company adopting an agile philosophy. At Meldevelopment, a group of senior engineers (in-house champions) believed that agile was a better approach for product development, which become a key factor in their successful agile adoption. These engineers recognised that their RUP methodology was too structured, causing frustration through delays to the actual development work. They devised an agile approach and trialled it extremely successfully on a very high value development project that met all of their engineering (quality) and business (sales) expectations. The development success of this project was attributed to adopting the

agile philosophies of creating a shared vision, assigning extremely talented and motivated engineers, and co-locating the team in one room for constant communication for the special project.

> Raw results … the team got a lot of accolades for the product … can demonstrate the results of agile development … much easier to get support … promoted to the position of director of software Engineering. (Direct Software Engineering)

The buy-in for the agile approach was achieved by a series of activities (change management process). This involved presenting the approach and the success factors regarding the trial project to senior management. This was followed by promoting the agile approach within the development teams by making agile textbooks available and doing presentations to the entire engineering department. To further promote the agile approach, regular engineering reflection workshops were held during projects. Finally, for Meldevelopment to embrace the agile approach, an offsite two day event was held to create and establish a strong vision of a team approach for product development using agile methods.

> CEO gave support to hold an offsite event…theme was breaking all barriers …set engineering vision … team building and bottom-up brainstorming … most rewarding activity … ignited spark in the whole team. (Director Software Engineering)

5.6 Summary

This chapter provided adaptation information of Meldevelopment's agile approach for software development. Their agile approach practices and related adaptations have been captured, analysed and presented using Fitzgerald's adaptation framework; original formalised methodology vs. Methodology-in-action; profile of the development environment, covert factors and overt factors. It shows that these four major adaptation factors with their sub-factors drive the adaptation of the agile approach at Meldevelopment. Hence, the chosen framework appears relevant. However, analysis of Meldevelopment's adaptation data suggests modification to this framework and its major components to account for the particular adapted processes within agile teams. The proposed changes are suggested in Chapter 6.

Chapter six: Case study analysis

This thesis has used two case studies to investigate agile method adaptation factors. The previous two chapters have provided in-depth descriptions of the two case studies, including the backgrounds that led to the adoption of agile methods and their adaptation. This has made it possible to answer the two sub-questions of this research:

a. How and why do organisational factors influence the adaptation of an agile approach?
b. How and why do intellectual (overt) and political (covert) factors influence the adaptation of an agile approach?

This chapter provides a cross-case analysis of the two case studies to answer the main research question:

1. How does adaptation work in an agile approach?

This chapter is organised as follows:
First, the two case study backgrounds are compared. These are discussed in the context of their development structures and the related agile method adaptation.

Second, cross- case analysis of the two case studies is provided. This involves an analysis relating to the methodology-in-action and to three key adaptation factors; the organisational, overt and covert factors. The discussion relates the case study data back to the key literature on method adaptation (static and dynamic adaptations, method fragments) and on behavioural adaptation (adaptive work performance behaviour). For each case study, the adaptation factors are identified and explanations are provided.

6.1 Background comparison of the two cases

The two case study organisations have product managers (who write project specifications) and project managers (with Akldevelopment, the title is Engineering Manager). The two also are international software vendors and market leaders.
Table 11 provides a summary of their similarities. However, the two organisations have differences with their products, customers, engineering departments (sizes), adopted methods, and development problems and challenges. Table 13 provides a summary of the differences between the two organisations.
The following analysis shows how roles were adapted with their agile approaches.

Table 11 Case study background (similarities) prior to agile adoption

Similarities	Meldevelopment	Akldevelopment
Business	International software vendor.	International software vendor.
Business Function	Product manager provides project specification	Product manager provides project specification
Team Management	Project manager	Engineering manager (project manager)

6.1.1 Similarities in development structures and related adaptation

6.1.1.1 Product manager role

As highlighted in Table 11, both organisations had product manager roles. This business role wrote project specifications but had limited face-to-face collaboration with their development teams in their previous development approaches.

In accordance with an agile approach for software development both organisations adapted to have on-site customers to enable their business functions to become proactive at development. With Akldevelopment, the product manager adapted to this role, while at Meldevelopment it was the project managers (proxy-product managers). This role adaptation distinguishes agile software development from the other development approaches. Hence, the developer-embodied factors component in Fitzgerald's (1998) adaptation framework is adapted to include this role and other roles that are part of the agile team rather than just the developer role (Figure 70).

With this adaptation, both organisations' product development became a *cross-functional* effort (Gunasekaran & Yusuf, 2002) giving them development *agility* (Kettunen, 2009). Their on-site customers drive development activities which makes it easier to communicate the product vision (high-level requirements) at the development

level. They provide in-person interaction to their development teams allowing for *spontaneous* collaboration (Dyer & Shafer, 2003).

The on-site customers at both organisations perform *multiple roles (*project manager, tester and domain support) (Dyer & Shafer, 2003). Clearly, this role at both organisations has individuals who have *adaptive performance* capabilities (Pulakos et al., 2000).

Both organisations have a successful adaptation to have on-site customer roles because the individuals in this role have *swiftly* adapted with new *skills* (Breu, Hemingway, Strathern, & Bridger, 2001) and accepted the *additional responsibilities* (Plonka, 1997). The individuals in this role also had product development experience with their respective organisations.

Both organisations had to further adapt to enhance their on-site customer roles. Akldevelopment adapted with formal product analyst roles to be the permanent on-site customers allowing the product manager role to focus on the fieldwork. At Meldevelopment, the project manager roles were adapted to be product development managers enhancing the contribution at development level by bringing in first-hand perspectives from the field.

At Akldevelopment and Meldevelopment, this adaptation is a result of their *responsive behaviour* (Sharifi & Zhang, 1999); on-site customer roles allow for rapid problem solving at development level. This adaptation was possible because of their *strategic commitment* (executive support for implementing adaptation decisions) (Meredith & Francis, 2000). This role adaptation was *mutually accepted* (Bynjolfson, van Alstyne, Bernstein, & Renshaw, 1997).

The individuals appointed to the on-site customer roles at both organisations came from within their agile environment. Their agile development *cultivated skills agility* that enabled the individuals to be appointed in this (higher) role (Parkinson, 1999).

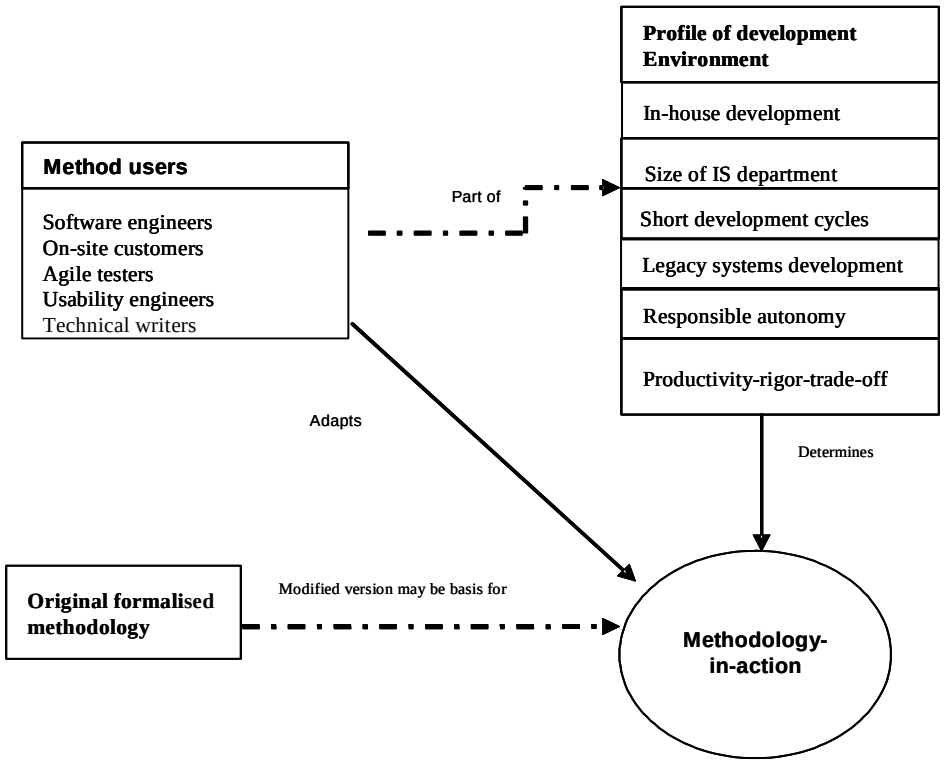
At Akldevelopment, their agile practices such as creating product backlogs, daily stand-up meetings, and iteration planning meetings provided the *cross-functional training* (Yao & Carlson, 2003) for engineers to take up this role. At Meldevelopment, the product manager worked directly with the project managers providing the on-job

training. This on-job training enabled project managers to learn product management tasks.

At both organisations, the on-site customers have *adaptive behaviours* where they have adapted to be *proactive, reactive and tolerant* to successfully implement the product vision at development level (Griffin & Hesketh, 2003; Sherehiy, Karwowski, & Layer, 2007). *Proactive behaviour* enables them to solve new and unfamiliar problems relating to the product vision during implementation. *Reactive behaviour* enables them to make contributions during their product backlog planning (low-level requirement definition), cope with changing business objectives and to work effectively with teams to achieve development goals. *Tolerant behaviour* enables them to cope with stressful situations and uncertainty during projects.

Finally, the on-site customer role at both organisations is as important as the development roles since they provide the needed business contribution at development level. They, together with software engineers, quality assurance engineers, usability professionals and technical writers determine the adaptation of their agile approach to meet the business needs. Hence, Fitzgerald's (1998) adaptation framework is modified to reflect these findings. The developer-embodied factors component identified in the original framework is removed; engineers as method users are now part of the Developer/methodology user component, which is labelled as 'method users' in Figure 70. The component (method users) identifies other roles as part of the agile teams at Akldevelopment and Meldevelopment (organisational factor), whereas in the initial framework only developers are recognized as the method users. (Figure 70 shows only part of the adaptation framework).

Figure 70 Enhanced method users component of the adaptation framework



6.1.1.2. Project manager

With both organisations the development teams and the projects they undertook were managed by project managers, (at Akldevelopment the engineering manager is the project manager) as shown in Table 11. Prior to agile adoption, all aspects of their development work were planned and micro-managed by them.

On adoption of an agile approach at Akldevelopment and Meldevelopment, the project management is *delegated* to the individuals in their agile team (Sherehiy et al., 2007). The engineers and on-site customers collaborate to do project plans, creating product and release backlogs, estimating tasks or stories and planning iterations. They also monitor their own schedules through plotting burn-down charts. This *proactive behaviour* at both organisations enables engineers to have a hands-on approach to plan and manage their development efforts (Dyer & Shafer, 2003). This adaptation enables more reliable schedules and engineer commitment for timely implementation of features, making engineers responsible for product delivery.

With project management delegation at Akldevelopment and Meldevelopment, the team managers adapt to *macro-manage* the projects and team performance (Sharp, Irani, &

Desai, 1999). These managers adapt to become outward or customer focussed and no longer direct, control or monitor the day-to-day tasks and work performance. They bring field experience into the development environment. This is the required *adaptive behaviour* for team managers to be *proactive* in the agile teams. The project manager as a specialist role is made redundant.

This has an impact on the adaptation framework, in particular on the overt adaptation factor 'economic skill specialisation'. This factor now guides team roles to be adapted to a generalist skill set while reducing the division of labour at development level.

6.1.1.3 Agile organisation (international vendors)

The agility features (similarities) of the two organisations are identified in Table 12. Both had excelled at international level as software vendors. Their senior management provided and encouraged the *culture for change* enabling these two organisations to adapt their structures (Ren, Yusuf, & Burns, 2003). The agility features in Table 12 indicate that both were agile organisations prior to adoption of agile development approaches.

Some of the agility characteristics identified through background analysis of the two organisations were *market leadership* (Fliedner & Vokurka, 1997), *competitive strategy*- had highly quality product features (Goldman, Nagel, & Preies, 1995), *innovation proactivity*- released new product features regularly (Yusuf, Sarhadi, & Gunasekaran, 1999), *proactively anticipated customer requirements*- worked closely with customers (Vokurka & Fliedner, 1998), *cross-functional approach* – business function identified new features while engineering implemented them (van Oyen, Gel, & Hopp, 2001), *strategic alliance*- had partnership with international vendors (Duguay, Landry, & Pasin, 1997), and *empowerment culture* – had autonomous development teams (Sharifi & Zhang, 1999).

These agility characteristics provide a possible explanation as to how both organisations have successfully adopted an agile approach; they have adaptable management structures which accepted a need for process reform. These indicate that their organisational factors are dynamic which are influenced by market forces. The implication is that organisational (development environment) factors in Fitzgerald's

(1998) adaptation framework must be structurally adaptable for agile development, supporting dynamic development capabilities.

Table 12 Agility features (similarities) of two case studies prior to agile adoption

Agility structures	Meldevelopment	Akldevelopment
Market leaders (Fliedner & Vokurka, 1997)	International	International
Competitive Strategy- highly quality products (Goldman, Nagel, & Preies, 1995)	Highly technical products	Highly strategic product
Innovation proactivity (Yusuf, Sarhadi, & Gunasekaran, 1999)	Seven major product lines and growing their potential	Adding new features and capabilities with emerging technologies
Proactively anticipates customer requirements (Vokurka & Fliedner, 1998)	Closely engage with customers	Closely engage with customers
Cross-functional approach (van Oyen, Gel, & Hopp, 2001) Strategic alliance (Duguay, Landry, & Pasin, 1997)	Product manager identifies new features and products Global partnership	Product manager identifies the market requirements Strategic alliance
Empowerment (Sharifi & Zhang, 1999)	Engineers responsible for development methods, process, and product delivery.	Engineers responsible for development methods, process, and product delivery

Next, analysis is provided on differences in development structures and how they were adapted with agile approaches.

6.1.2. Differences in development structures and related agile adaptation

The differences (Table 13) in the development structures highlight the unique organisational factors that impact products. Discussed below are some of these key organisational factors and how the two organisations have incorporated them with their agile approaches.

Table 13 Case study background (differences) prior to agile adoption

Differences	Meldevelopment	Akldevelopment
Product type	Systems software (utilities)	Business application
Customer size	Medium to large organisations	Large organisations
Target market	International - wide range	International- single sector
Engineering functional teams	Development (3 teams with 24 engineers), quality assurance and documentation	Development (up to 6 teams with 100 developers), quality assurance and documentation.
Development team size	Up to 7 individuals	Up to 15 individuals
Development roles	Generalist role- performs multiple tasks.	Specialist roles- perform specific task only.
Development method	Rational Unified Process	Most teams adopted Rational Unified Process

Development method related problems	Ineffective stakeholder partnership for requirement definition, non-emergent architecture, and unsustainable development pace	Unreasonable schedules (estimates) and ineffective requirement definition process
Other development challenges	Reallocation of development resource, in accurate task estimation, cross-functional (engineering) communication problems	Keeping up-to-date with regulatory changes, product adaptability with emerging technologies, and lack of local customer collaboration

6.1.2.1 Target markets

A key difference is in target markets; Akldevelopment's product focuses on one particular business sector while Meldevelopment has a wide range of sectors. Despite this difference both provide a general-purpose product for the international market while catering for individual customer needs. Both organisations have to learn as much as possible on how and where their products would be used to deliver outstanding products. Both adapted to enhance their agile approach with usability engineering and user-centred design approaches. This adaptation is in accordance with Baskerville (1996) that an agile approach provides the third-level abstraction to incorporate other key issues such as usability, which development methods usually ignore.

This has an impact on Fitzgerald's (1998) adaptation framework, in particular on the organisational factor 'in-house development'. The implication is that the in-house development factor must be adaptable for agile development, enabling an adapted in-house capability to take care of usability concerns.

6.1.2.2 Work domains

Another difference was their work domains. Akldevelopment's product is for a business work domain. The acquisition of specific information relating to end-users is critical to enhance their performance when using Akldevelopment product. They adapted with a usability engineer. In contrast, Meldevelopment's products are for a technical work domain. The acquisition of specific information relating to such domains is also critical, having quality implications. At Meldevelopment, technical writers adapted their skills to become usability experts and adapted their agile approach with a *generative rule;* a compulsory task analysis practice to identify likely user issues so that they can provide troubleshooting capabilities (Highsmith, 2002). Both provided *value added capabilities* with their products (Yusuf, Sarhadi, & Gunasekaran, 1999). This is in accordance with Plonka (1997) that agile organisations must adapt with *human factor specialists.*

This has an impact on the framework, in particular on the organisational factor 'in-house development'. The framework for agile adaptation must have a dynamic in-house development factor where in-house development capability is enhanced through adaptation to fix any usability concern through usability practice or skills.

6.1.2.3 Size of engineering department

The other difference is in the sizes of the engineering departments. Akldevelopment had up to six teams with one hundred developers. In contrast, Meldevelopment had fewer development teams (three teams with twenty-four engineers)

However, the agile team at Akldevelopment was allocated a fixed capacity (twenty-four individuals) similar to Meldevelopment. They both maximised their development effort by adapting the *team structures* to have sub-teams for projects (Meredith & Francis, 2000).

This has an impact on the adaptation framework, in particular on the organisational factor 'size of IS department'. In an agile adaptation framework this factor must be dynamic, allowing adaptation to team structures and roles.

6.1.2.4 Development roles

Another difference highlighted in Table 13 is with their development roles. Akldevelopment had specialist roles while Meldevelopment had generalist roles. However, Akldevelopment adapted with *multi-skilled* (generalist) engineering roles with an agile approach (Breu et al., 2001). At both organisations, these roles adapted to have wider development responsibilities and empowerment to make decisions.

This has an impact on the adaptation framework, in particularly on the organisational factors 'in-house development' and 'responsible autonomy'. In an agile adaptation framework these factors must be dynamic, allowing in-house development role adaptation to have appropriate skills and empowerment to deliver products impacted by frequent changes in the marketplace.

6.1.2.5 Development methods

The other difference in Table 13 highlights the adopted methods organisation-wide. At Akldevelopment, the development teams adopted their own methods. In contrast,

Meldevelopment had adopted a single method organisation-wide. However, with both organisations the engineers are empowered to adapt their agile practices. This empowerment for method adaptation enables them the *adaptive performance* to implement strategic features according to their market-driven business objectives (Pulakos et al., 2000).

This has an impact on the adaptation framework. The factors for agile adaptation must be dynamic, allowing appropriate adaptation in the product development environment according to the market forces.

6.1.2.6 Development problems and challenges

While both organisations had unique development problems and challenges, there were some common ones as shown in Table 13. These related to requirement definition and task estimation practices since they had ineffective stakeholder participation. These issues were also discovered by Ahituv, Hadass, & Neumann (1984) and Bantleman & Jones (1984) as significantly impacting the delivery schedules and costs of projects. For both, this problem had cost and quality implications since any slip in the quality downgraded their competitive advantage; their market releases were thoroughly scrutinised by industry analysts and prospective clients. Task estimation often produced unreliable schedules seriously stretching their development efforts. Similarly, the other problems and challenges listed in Table 13 impacted the ability to continue to deliver competitive features.

At Akldevelopment and Meldevelopment these issues warranted improvements for *strategic reasons* (Mathiassen, Pries-Heje, & Ngwenyama, 2002). These were the reasons to adopt an agile approach; it provides them with the *flexibility* to quickly adapt as dictated by their *culture for improvement* (Yusuf, Sarhadi, & Gunasekaran, 1999).

6.2. Original formalised methodology vs. methodology-in-action

For both organisations, the agile approach rather than a single agile method *paradigm* (Lyytineen, 1987b) served as the basis for the “Original Formalised Methodology”. These are the *four values and twelve principles* of an agile approach (Highsmith, 2002). For their market-driven product development, this provides them with the flexibility to *swiftly* adapt chosen practices and structures to keep pace with their evolving markets

and technologies (Lindvall et al., 2004). As described by Flaatten (1992), the development method must support achieve business goals and objectives.

The match between the method paradigm and business objectives/goals is critical to determine an original formalised agile method appropriate for a market-driven product development environment, allowing for swift adaptation on projects. Hence, the original formalised methodology vs. methodology-in-action component in the framework is adapted to include this key feature (match between the method paradigm and business objectives) for adopting an appropriate agile method (Figure 71).

At Akldevelopment, the engineering manager helped to develop an understanding of the agile approach and individual agile method paradigms, and their relative fit with their product development environment. This was one of the main reasons for a successful deployment. This individual had a strong *belief and attitude* for this approach (Dahlbom & Mathiassen, 1992). These were shaped by his *extensive experiences* with agile approaches (Ørvik, Olsen, & Sein, 1999) allowing him to *coach* a new team on agile deployment (Aydin, Harmsen, Slooten, & Stegwee, (2005).

However, the successful deployment at Meldevelopment's was due to the engineers' past method adaptation experience and an understanding of method paradigms. Meldevelopment had *process champions* (engineers) who ensured successful agile approach deployment (Shane, Venkataraman, & MacMillan, 1995). Clearly, these individuals had a strong *innovative* behaviour, where they had identified problems and suggested an agile approach for development (Crant, 2000*)*. These were shaped by their attitude to achieve a *concrete solution* (Ashford, Rothbard, Piderit, & Dutton, 1998) where they tried an agile approach for a development project and demonstrated the results. This was a key factor for agile buy-in.

Experience in deployment provides for the *proactive behaviour* (Crant, 2000) of the method champions; at Meldevelopment they took control of their development situation when they felt pain with their development approach.

The role of an agile coach or method champions is also critical for swift deployment of an agile method (original formalised method) in a market-driven product development environment, minimising the impact on delivery and quality of implemented features.

Hence, the original formalised methodology vs. methodology-in-action component in Fitzgerald's (1998) adaptation framework is adapted to include this important requirement (agile coach or method champions) for swift agile adoption (Figure 71).

Neither organisation limited their agility to a single agile method. Any adaptation to counter emergent issues would have been limited to the adopted method only, limiting their development agility. Akldevelopment have regular sprint reflection meetings while Meldevelopment has regular project reflection meetings (including during projects on an as needs basis) to make adaptation decisions. As described by Kumar & Welke (1992), method adaptation is necessary because of different situations under which projects are undertaken. Most important is that both have an *amethodical* approach where the methodology-in-action is adapted to be relevant to projects through an *informal or fluid* adaptation process (Baskerville, Travis, & Truex, 1992). As described by Highsmith (2002), agile adaptation is a low ceremony process. This approach enables adaptation decisions on the fly as a result of quick decisions by method uses.

For both organisations, their method-in-action was adapted from the start. While both have a market-driven product development, they have different method fragments highlighting their organisational differences. As described by Parsons, Ryu, & Lal (2007) organisations adopt practices from different agile methods. Akldevelopment had practices adopted from the XP and Scrum methods, giving them a hybrid agile method (methodology-in-action). However, Meldevelopment had practices adopted from several agile methods; DSDM, Scrum, Feature Driven Development, and XP. It also included practices from the Rational Unified Process (RUP); and the usability engineering method, User Centred Design (UCD).

Meldevelopment requires prototypes and models to identity the features and for developing product architecture. Their stakeholders cannot provide the information on requirements upfront. They adopted practices to learn in-depth about their visionary requirements; this includes practices from the Feature Driven Development method (Salo & Abrahamsson, 2008). The others such as the DSDM practice of developing *evolutionary* architectural prototypes (Barritt, 2002), the RUP practices of developing *use-case models* and *throwaway prototypes* (Hirsch, 2002), and the user centred design approach for prototyping user interfaces by creating *personas and user profiles* (Ungar & White, 2008); these are part of their design phase.

Both have a selection of XP method practices; *test driven development, refactoring, pair programming, collective ownership, continuous integration, on-site customer, coding standard, open workspace* and *just rules* (Rosenberg, Stephens, & Collins-Cope, 2005; Salo & Abrahamsson, 2008). These practices define the team rules and how work should be done at their development level.

Both have adopted a planning approach that is inclusive of the needs of their business and senior management; their development schedules and tasks are *visible* and *transparen*t (Ramesh & Devadasan, 2007). Akldevelopment achieved this by adapting with the Scrum method practices; *product backlog*, *effort estimation*, *sprint*, *sprint planning*, *sprint backlog*, *daily scrum* and *sprint review meetings* (Schwaber & Beedle, 2002). These practices provide them with *project management* essences (Sutherland, 2001).

However, Meldevelopment has achieved this by adapting with practices from Feature Driven Development, XP, and Scrum; the feature driven development method practices of *build a feature list* (product backlog), *plan by feature* and *design by feature* (Salo & Abrahamsson, 2008), XP method practices of *iteration planning, stand-up meetings* (Rosenberg, Stephens, & Collins-Cope, 2005) and Scrum practice of *review meetings* (Schwaber & Beedle, 2002). The difference in project management practices is due to Meldevelopment developing several products concurrently.

The agile approach paradigm provides both organisations with the flexibility to adapt their development environment in an ideal way where they have several agile methods at their disposal from which they can select and adapt practices. As described by Highsmith (2004), the ingredient for success is based upon team discussion and understanding about the relevance and issues of practices. The approach provides them with the development agility to continually meet business objectives (they frequently change) and the flexibility to swiftly deal with emergent issues.

For both, the key drivers of development agilities are *empowerment* (van Oyen, Gel, & Hopp, 2001*)* and *speed* (Crocitto & Youssef, 2003). As a result, individuals are given a *wide span authority* (Sherehiy et al., 2007) for selecting and adapting agile practices, techniques and tools. As described by Markus & Bjørn-Anderson (1987), the adoption decision must not be made solely by the managers. For both organisations, the

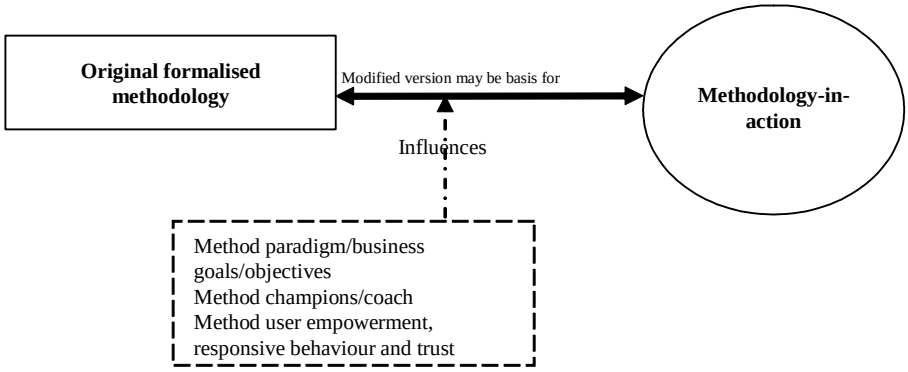
individual authority is *communally-based* (Hopp & Van Oyen, 2004). Communally-based involves collective decision making. At both organisations, the agile team members (on-site customer, software engineers, quality assurance engineers and usability professionals) have *responsive capability*, which is reflected through their competency to speedily identify and immediately adapt their agile approach to deliver products (Sharifi & Zhang, 1999).

Another reason for development agility is that both organisations have a culture of *trust* (Hopp & Van Oyen, 2004; Gunasekaran, 1998). The senior management have the confidence in their agile teams. This was the key factor for shifting the decision making authority to the agile team allowing for faster decision making. This also accords with Stapleton (2003) and Poppendieck & Poppendieck (2003); the agile teams are trusted and empowered to make decisions on a wide range of development issues.

The empowerment, responsive behaviour, and management trust for method adaptation by method users are critical features for swift agile method adaptation in a market-driven product development environment. Hence, the original formalised methodology vs. methodology-in-action component in Fitzgerald's (1998) adaptation framework is adapted to include these important features (method champions or agile coach) (Figure 71).

Finally, these findings at Akldevelopment and Meldevelopment show the influencing factors on the method in-action in an agile software development environment; method paradigms match with business goals and objectives, use of method champions and coaches for deployment, and method user empowerment, responsive behaviour and trust for method adaptation. Hence, Fitzgerald's (1998) framework has been adapted to reflect these findings (Figure 71). This shows part of the adaptation framework only (original formalised methodology vs. methodology-in-action).

Figure 71 Influences for method-in-action



6.3. Profile of the development environment

An analysis of the organisational factors is now provided for both case studies; the cross-case analysis of the adaptation factors under the profile of the development environment which includes the in-house development, size of IS department, project duration, legacy systems development, developer autonomy, and productivity-rigor trade-off.

6.3.1 In-house development

For Akldevelopment and Meldevelopment, in-house development was one of the major organisational factors that drove the adaptation of their agile approaches. Both built their in-house development function as their *core competency* (Jin-Hai, Anderson, & Harrison, 2003). This *core competency* (a central factor) has enabled them to keep their status as market leaders.

They adapt swiftly to shifting market prospects to deliver unanticipated product innovations. They have a *dynamic* rather than a *static core competency* (Lei, Hitt, & Bettis, 1996). This factor enables both organisations to create *firm-specific* development competencies (Prahalad & Hamel, 1990).

Both have strategic alliances and outsourcing to *complement* their core capabilities (Hagedoorn, 1995) enabling *performance differential* in the marketplace (Duysters & Hagedoorn, 2000). Performance differential involves developing and making available outstanding features and products.

Both organisations have three major factors driving adaptation of their agile in-house development to achieve dynamic core competency. These are team approach, technical

expertise and domain knowledge. These are critical agile in-house adaptation factors which drive continuous adaptation to have a dynamic development function, providing market-driven development capabilities. Hence, the in-house development as a development environment factor in Fitzgerald's (1998) adaptation framework is adapted to include these three (team approach, technical expertise and domain knowledge) as sub-factors of the in-house development factor (Figure 72). First, an analysis of team approach is provided.

6.3.1.1 Team approach

The team philosophy enables both companies to achieve *team effort* (sharing of tasks*)* and *team ownership* (the entire team collectively responsible for the delivery of committed features) including *cross-functional collaboration* with their business function at the development level. As described by Edmondson & Nembhard (2009), product development requires functional units to work as a team on projects to deliver quality products in the shortest time. This team philosophy extends to their external collaborators; the strategic alliance partners and development contractors. This gives both the *organisational agility* for their market-driven product development (Sherehiy, Karwowski, & Layer, 2007).

While team strategy enables both organisations to provide *customer enrichment* and *customer satisfaction*, Meldevelopment also have *customer-driven innovations* (customers identify and drive implementation of new features). Such dynamic core competency enables both to have highly distinct products.

Both provide c*ustomer enrichment* through business function contributions at development level. Akldevelopment incorporates emerging technologies and web service functions, and Meldevelopment provides diagnostic or troubleshooting capabilities. Customer enrichment is providing high performing features.

Akldevelopment and Meldevelopment both provide *customer satisfaction* through product customisation and customer requested features, enabled by cross-functional collaboration. At Akldevelopment, software engineers are involved in sales negotiations, while the Meldevelopment product manager directly negotiates with development teams to implement customer requested features. *Customer-driven*

innovations enable the delivery of better products. This enables Meldevelopment to identify and build the right features and products.

A more strategic team approach is developed through internal and external *cooperation.* This cooperation enables both organisations to effectively *coordinate* their domain and technical expertise to speed development and availability of new features (Faraj & Sproull, 2000).

External cooperation enhances both organisations' *core competency* to deliver quality products. Through strategic alliances they acquire strategic technical information to incorporate them with their products.

Both use contractors and outsourcing companies to further enhance their team approach for development. Contractors and outsourcing companies are used for low risk development tasks to meet their development demands, keeping the technical knowledge of their innovations in-house. As described by Clark, Jr., Zmud, & McCray (1995), the loss of internal technical knowledge is one of the major pitfalls of outsourcing. The use of the external sources enables effective coordination of their technical expertise to focus on new innovations.

The above analysis highlights the importance of a team approach in agile product development. This team approach distinguishes agile software development from other approaches. For this reason, the in-house development factor in Fitzgerald's (1998) adaptation framework is modified to incorporate a team approach as a sub- factor (Figure 72).

Next, analysis of team approach with high-level product planning is provided.

6.3.1.1.1 High-level product planning

Akldevelopment and Meldevelopment had adapted their agile planning practice to have a team approach for product (vision and roadmap) planning. In a market-driven development, product plans are even more important agile *method fragments* than project plans. As described by Groenveld (2007), these plans motivate learning through organisational openness in attaining milestones and commitment of individuals in developing products. With both organisations, the related projects are identified based on product plans.

Initially, at both organisations the project specification was the artefact given to the agile teams identifying features to be implemented, having no input from the engineers. This approach for product planning creates implementation delays, requiring significant effort by engineers to learn what is to be implemented.

Both organisations adopted vision and roadmap planning practices as key agile *method fragments*. This made their high-level planning a cross-functional task, having a team effort through internal cooperation between business and development functions for making better priority decisions on new innovations and setting more achievable schedules while allowing immediate implementation.

At Akldevelopment and Meldevelopment, the product manager and engineering roles are key *method fragments* for this practice. While the product managers identify new innovations, engineers provide the high-level estimates for developing business cases. The external cooperation (industry publications, market investigations, clients and industry consultants) and internal cooperation (sales and marketing department) provide ideas for new features. These are strategic *method fragments* for both organisations, enhancing their team effort for product planning. This approach for product planning is according to Nuseibeh & Easterbrook's (2000) study that found that this type of a team elicitation technique promotes stakeholder agreement and buy-ins whilst eliciting a richer affluent knowledge of requirements.

This product planning practice enables *organisational learning* to extract implied features from various sources (Sherehiy, Karwowski, & Layer, 2007). For both companies, it also includes organisational learning with regards to identifying skills and knowledge support at development level. At Akldevelopment, this enabled the product manager to co-locate and support their agile development team. At Meldevelopment, it led them to identify the technical skills requirement for their product manager to effectively communicate with engineers to develop product visions. Hence, this product planning practice enabled both organisations to adapt to have a better team approach for in-house development. This *dynamic adaptation* was *mutually accepted* enhancing both organisations' core competency.

Another *dynamic adaptation* of the planning practice involved client organisations providing feedback on simulated features, helping to determine the strategic value of features. As described by Cheng & Atlee (2007), adapting to a feedback technique for

requirement definition helps to extract early reactions to a proposed system. This *external cooperation* is a strategic *method fragment* for both organisations. This *mutually accepted* adaptation enhanced both organisations' team approach for in-house development of their products.

The *internal cooperation, resilience and responsive behaviour* of the agile teams at both companies to build functional prototypes enable this *external cooperation* (Sherehiy, Karwowski, & Layer, 2007). Resilience behaviour is the ability to cope with change. These extra developments are done while the engineers deliver their regular commitments. This *adaptive (responsive) behaviour* and flexibility to experiment proposed innovations raises their *core competency*.

Both Akldevelopment and Meldevelopment had another *dynamic adaptation* of their agile product planning practice, creating new *agile method fragments*. Akldevelopment adapted with a product strategy group and a steering committee for collective project approvals. Meldevelopment created a product planning group, adapting with a collaborative approach for project approvals. This *mutually accepted* adaptation further enhanced both organisations' team approach for in-house development of their products.

This new organisational structure for product planning enables *strategic decision making* in a market-driven product environment (Vãzquez-Bustelo, Avella, & Fernãndez, 2007). This planning structure ensures that any new innovations identified are of strategic value since the joint decision making ensures first time right decisions. They entailed a cross-functional membership enhancing both organisations core competency. This organisational structure gives both organisations the responsive ability.

The above analysis highlights team approach as being critical to agile product planning practice. Next, analysis of team approach with low-level planning is provided.

6.3.1.1.2 Low-level planning

At Akldevelopment, creating product backlogs is one of the key agile *method fragments*. This was a *dynamic adaptation* of their planning practice. In contrast, Meldevelopment had a *static adaptation* with a design phase in their agile approach, since they adopted it from their previous development approach. Because of their

exceptional technical innovations they still required this phase with their agile approach. This phase is one of the key agile *method fragments*.

For both organisations, this phase is *time-boxed* (Helming, Koegel, & Hodaie, 2009). This time-box practice was a *dynamic adaptation* of the agile development environment and an important *agile method fragment*. The time-boxed method is critical to track the progress of development activities and to keep focus on the task in hand, enabling a highly productive team effort.

Both organisations have a cross-functional cooperation, enabling them to swiftly identify and immediately start implementations. At Akldevelopment, product analysts decompose high-level requirements to have new features developed exactly as determined by the product strategy group. At Meldevelopment, the product manager became proactive at the development level, setting high-level priorities, taking part in design activities and testing builds. This adaptation at development level with business contribution enhances the team effort for achieving swift implementations in agile development environments.

There were additional *mutually accepted* adaptations in relation to this practice. Akldevelopment adapted to include a usability engineer, while Meldevelopment adapted to include quality assurance (provide testing perspective) and documentation function (provide usability perspective) participation for analyzing the high-level requirements. As described by Lukanuski, Milano, Bruin, Rochford, & Bosman (2008), the usability engineer's contribution in planning the product backlog helps to shape user experiences. For both, this was a *dynamic adaptation* and important *agile method fragments*. These structures enable a *multi-disciplinary* agile team to become quality focused right from the beginning (Medhat & Rook, 1997). Hence, these adaptations further enhanced the team effort at development level.

At both organisations, these roles perform multiple roles; they help to plan the product backlogs, highlighting usability and quality issues to be addressed. During implementation they also coach, providing on-the-job training for software engineers to learn usability and quality concepts. The coaching up-skills software engineers and makes them *proactive* with product testing issues. The software engineers require *adaptive behaviours* to acquire usability skills and the mind-set of testers to implement unit tests.

Akldevelopment and Meldevelopment made another key *dynamic adaptation* to improve their product backlog planning practice. At both organisations, all the engineers on a project were involved with this activity. This task sharing enables both organisations to achieve team effort, making their core competency strategic. This practice was adapted to involve only senior engineers to enable continuous implementation. This *mutually accepted* adaptation enables a better coordination of expertise and to have a more productive team effort. Hence, this adaptation enhanced the team approach at development level.

Both Akldevelopment and Meldevelopment made another *mutually accepted* adaptation to improve their backlog planning practice; these involved use of tools and techniques. As described by Cook & Churcher (2006), an agile development environment requires good tool support for collaboration. At Akldevelopment, it involved both *dynamic* and *static adaptation* of tools. They used both index cards and a software tool (XPlanner), judging index cards to enable a better team performance for capturing user stories.

Meldevelopment adapted with use case modelling. This was a *static adaptation* and an important *agile method fragment*. Use case diagrams are appropriate artefacts to learn in-depth about their highly technical innovations. Use case diagrams enable a mutual understanding of features where this artefact is a useful communication tool between the product manager and engineers. Another *dynamic adaptation* which further improved this practice was a user-centred design approach. This enables their documentation team to become proactive at this phase. When compiling documentation they usually end up performing usability tests. Meldevelopment also adapted with prototyping practice. This was a *dynamic adaptation* and very useful *agile method fragment,* since they get instantaneous feedback on their complex innovations. Hence, these *mutually accepted* adaptations at both organisations further enhanced their team effort at development level.

The above analysis highlights team approach as critical in agile planning practice at development level to swiftly identify low-level requirements for implementation. Hence, this team approach for project planning at development level distinguishes the agile approach from the others.

Next, analysis of team approach with development and testing is provided.

6.3.1.1.3 Development and testing

The pair programming practice enables Akldevelopment to achieve a team effort for user-story implementations. Meldevelopment have solo effort for writing code; this is the only part of their task implementation that has individual effort. They have adapted task implementation with code reviews requiring input from another engineer and also collaboration with a quality assurance engineer to implement unit tests. Akldevelopment and Meldevelopment both have task-sharing enabling a team approach at the development level. This team approach is critical to deliver implemented features in short development cycles.

Pair programming was a *dynamic adaptation* at Akldevelopment and their means to up-skill all engineers to an appropriate level of required in-house development competency. As described by Hulkko & Abrahamsson (2005), pair programming is most useful for on-job learning and training. At Meldevelopment, solo programming was a *static adaptation*; the solo coding effort is a motivation for ownership for future enhancements and growth of the features by engineers. This difference between the two organisations is a result of development experience, where Meldevelopment has more senior engineers than engineers.

At Akldevelopment and Meldevelopment, *training and education* (Gunasekaran & Yusuf, 2002) is a key factor for building their core competencies. They provide a development environment through their coding practices that facilitates on the-job training and development of multiple engineering skills. This generalist skill set is an integral part of their core competency to enhance team effort for feature implementation.

Meldevelopment engineers require a solid technical foundation through their own product implementation experience, which they achieve through solo programming. This provides the engineers with a solid foundation for innovation and speed on their highly complex projects. The solo programming and code review practices together provide engineers with on-the-job training and development of engineering skills.

At Akldevelopment and Meldevelopment, the software engineers had *adaptive behaviours*, where the engineers have interpersonal and cultural adaptations enabling spontaneous face-to-face collaboration in their team work environment (Dyer & Shafer,

2003). These adaptations are critical to ensure a highly effective team approach for agile development. According to Griffin & Hesketh (2003) interpersonal skills not only enable spontaneous collaboration in an agile workforce, but also make the team more proactive and resilient.

At both organisations, engineers developed personal *initiative* to learn new skills, tasks and responsibilities when collaborating with other individuals. Adapting with *resilience behaviour* was critical for engineers to be able to task-share, enabling them to deal with new ideas and suggestions, frequent interruptions, noise and any other stressful situations.

They both made a *dynamic adaptation* of their practices. Akldevelopment adapted to a daily partner change for user-story implementation, allowing multiple perspectives for implementing a story. This adaptation enables a better quality code being implemented. Meldevelopment adapted to include pair discussion, involving collaboration with quality assurance engineers. This adaptation enables them to identify all the likely breakages that may happen and also enables better quality unit tests to be implemented. This early support helps to identify and fix bugs within iterations and avoid costly fixes later. These *mutually accepted* adaptations at both organisations further improved their team effort at development level.

This analysis highlights team approach as critical to be successful at implementation level with agile development. Hence, this team approach for implementation distinguishes the agile development from the others. Finally, the findings show that agile in-house development has three phases for market driven product development; product (high-level) planning, low-level planning and an integrated testing and development.

Next, analysis on technical expertise is provided.

6.3.1.2 Technical expertise

Previously, Akldevelopment had a specialised technical capability (architects, analysts, designers, programmers, performance experts and testers). With frequent advances in emerging technologies, these *static* roles restricted their *ability* to develop technologically adaptable products (Lei, Hitt, & Bettis, 1996).

Their development routines with deeply embedded knowledge and skills created a *rigid core competency* where individuals lacked holistic knowledge and experience relating to the overall product design and architecture to deliver innovative functions (Leonard-Barton, 1992).

Akldevelopment made a *dynamic adaptation* of their specialist development roles. These are now generalist roles and important *agile method fragments*. Meldevelopment had generalist engineering roles, which were enhanced by incorporating testing tasks on agile adoption. They also had a *dynamic adaptation* of their engineering roles.

Both organisations now have dynamic engineering roles, carrying out almost every task involved with their product development. The *complexity* and *dynamism* of their development routines shape their engineering roles (Lei, Hitt, & Bettis, 1996). As described by Hissey (2000), a modern day engineer is no longer an isolated innovator with a specialist skill set. Adapting roles to perform multiple development tasks is critical to enhance technical expertise in agile teams.

These roles allow for more efficient resource utilization, allowing organisation-specific multi-skilled and flexible workforces. For both organisations, these roles also enable *organisational learning* for support roles (Akldevelopment- product analyst, usability engineer and agile tester and Meldevelopment- product development manager and agile tester) to help the development of new engineering skills (Cohen & Levinthal, 1990). Hence, this *mutually accepted* adaptation to have support roles enables both organisations to further enhance the technical expertise of their engineers.

At both companies, product innovations are now implemented through *architectural innovation* as a result of engineering roles (Henderson & Clark, 1990). They now have the ability to swiftly incorporate new technologies with their products, strengthening both organisations' leadership in their markets. This *technology relatedness* capability is now part of their *core competency* (Teece, Rumelt, Dosi, & Winter, 1994). This *dynamic adaptation* has enhanced both organisations *performance differential* (Hamel & Prahalad, 1994).

The engineering roles also brought in a flattened team structure at Akldevelopment and Meldevelopment, removing the adherence to authority and control. This made product development activities a shared effort where engineers contribute to common tasks,

achieving loyalty and commitment in their agile teams. Everyone contributes regardless of their level of experience, skills or seniority.

At both organisations, the senior engineers are now heavily involved with the implementation, sharing their technical and domain knowledge with others. This involvement creates a continuous learning and skills development environment. Through this learning environment technical expertise is continuously being enhanced in agile teams.

At both organisations, the engineering roles reduce power differentials. The distinction between those who are commanding and those who are executing the tasks is erased. Engineers adapt their individual competency with *problem-solving abilities* and *decision making* competency (Breu et al., 2002; Plonka, 1997). Engineers also adapt with *responsive behaviour* to deliver features according to new priorities (market or business objective changes).

At both Akldevelopment and Meldevelopment, the engineers have become *proactive, reactive,* and *tolerant* in their development environment (Plonka, 1997). The proactive ability of their engineers enables creative problem solving (to deal with emerging technologies, identify, analyse and split epic stories or tasks, and deliver commitments on a regular basis).

Adapting with reactive behaviour ensures that engineers swiftly acquire interpersonal skills for collaboration and contribution to team effort. This behavioural adaptation also allows engineers to speedily learn new skills when working with other individuals including physically adapting to working in an open space that has free flowing interaction and collaboration.

Adapting with tolerant behaviour enables engineers to deal with stressful situations such as fast approaching release dates and to work effectively in their team work area frequently disrupted by high noise levels.

The engineering roles enable Akldevelopment and Meldevelopment to enhance technical expertise by maintaining the *tacit knowledge* of their product development (Lei, Hitt, & Bettis, 1996). This is achieved through a *personal reward system* based on their engineering levels; graduate engineer, engineer, senior engineer and principal engineer (Vãzquez-Bustelo, Avella, & Fernãndez, 2007). As described by Murawski,

Olds, & Miller (1996), the levels are used to define technical competence. Providing a clear career path is critical for agile team to enhance and maintain technical expertise through local product development experiences.

At both organisations, individuals in engineering roles have *above average* capabilities (Gunasekaran & Yusuf, 2002). This ensures them a core competency based on speed for learning, adaptability and responsive capabilities. Akldevelopment and Meldevelopment adapted their hiring process to help them to identify individuals with behavioural flexibility. This was a *dynamic adaptation* of their hiring process. New recruits are selected by a group of engineers using a practical hiring approach which includes collaboratively solving design tasks. This enables them to check the personality of an individual to determine their level of adaptability and suitability. This adaptability to work in a team environment is a critical factor to enhance technical competency in an agile development environment.

Both organisations also made *a dynamic adaptation* of their senior engineering roles. This *mutually accepted* adaptation requires senior or principal engineers to perform in multiple roles; a senior individual takes a team or technical leader role temporarily while still being an engineer in a sub-team. This adaptation creates informal authority that facilitates engineer consensus on technical and team issues. Senior engineers are the unofficial leaders and a key part of their core competency. As described by Schwaber & Beedle (2002), every agile team must have at least one very experienced engineer. This role adaptation is important to enhance technical competency of senior engineers in agile teams.

The above analysis highlights technical expertise as a critical factor for market-driven product development environments. This technical expertise (generalist) distinguishes agile software development from other approaches. For this reason, the in-house development factor in Fitzgerald's (1998) adaptation framework is modified to incorporate technical expertise as a sub- factor (Figure 72).

Next, analysis on domain knowledge is provided.

6.3.1.3 Domain knowledge

Akldevelopment and Meldevelopment both provide domain knowledge support at the development level through on-site customer roles. As described by Stapleton (2003),

agile teams must have a permanent business skilled individual. This was a *dynamic adaptation* of their agile approach and important agile *method fragment*. For both, the role improved their core competency, enabling them *quickness* in implementation and *timeliness* for delivery of new features (Sharifi & Zhang, 1999).

A key part of their core competency is knowledge management and continuous learning in regard to their product domain. Akldevelopment and Meldevelopment acquire strategic product knowledge from internal and external sources. The on-site customers ensure organisation-wide integration of this knowledge through communication of the strategic vision of proposed features. As described by Yusuf, Sarhadi, & Gunasekaran (1999), an excellent grasp of the domain is the most important attribute of agile organisations.

For engineers, on-site customers provide them with continuous learning opportunities in domain knowledge. The engineers at Akldevelopment and Meldevelopment adapted with *generative behaviour* to *learn and educate* themselves with product knowledge (Dyer & Shafer, 2003).

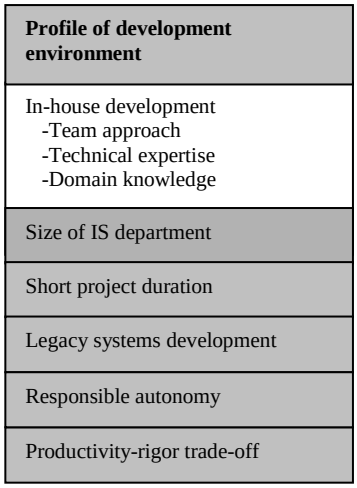
Both Akldevelopment and Meldevelopment made a *dynamic adaptation* of their domain support practice by acquiring domain experts. This was a new development structure for their product development environment. Through this *mutually accepted* adaptation, engineers at both organisations now have access to the experts to help them understand tasks and stories. Adapting to enhance domain support is critical for swift implementation of new innovations.

Akldevelopment and Meldevelopment both made further *dynamic adaptation* of their domain support practice. Akldevelopment enhanced the on-site customers' source for product knowledge. Rather than just liaising with the product strategy group, they also liaised with their marketing strategist and sales team. This mutually accepted adaptation further enhanced the on-sites customer's ability to provide domain support for engineers. Meldevelopment hired a principal engineer with significant domain expertise. They also adapted to have client presentations to acquire this knowledge. These *mutually accepted* adaptations enhance product knowledge at development level.

The above analysis highlights domain knowledge as critical factor for market-driven product development environments. This domain knowledge support distinguishes agile

software development with other approaches. For this reason, the in-house development factor in Fitzgerald's (1998) adaptation framework is modified to incorporate domain knowledge support as a sub- factor (Figure 72).

Figure 72 Sub-factors for in-house development



6.3.2 Size of IS department

This factor enables dynamic development structures (team setup and roles) of an engineering function at both organisations. It drives adaptation of their development structures to maximise product delivery. Analysis of data collected from Akldevelopment and Meldevelopment suggests two sub-factors associated with the size of IS department factor, which drives their adaptations; functional setup and informal engineering roles.

6.3.2.1 Functional setup

Akldevelopment had a *dynamic adaptation* with their software engineering department. They adapted the functional setup of their engineering unit with two functional units, development and documentation functions from the three that they had. The quality assurance unit was made redundant. Meldevelopment also had a *dynamic adaptation* of their functional setup of the engineering unit through adopting a collaborative approach between their three functional units (development, quality assurance and documentation units). This functional setup adaptation of engineering unit is critical in agile development to deliver implemented features regularly in short development cycles by incorporating quality assurance tasks upfront with development.

The quality assurance tasks at both organisations are integrated with the development function; software engineers adapted with this additional task which made their roles *fluid* and required role *flexibility* for carrying them out (Sherehiy et al., 2007). At Meldevelopment, other engineering roles also became fluid; quality assurance engineers became agile testers and technical communication engineers adapted with usability tasks. Hence, this functional setup adaptation is critical with agile development to have fluid and flexible roles within the engineering department. These engineering roles are *fluid* (multi-skilled), perform a wide variety of tasks and motivate joint efforts between the functional units and individuals.

Both Akldevelopment and Meldevelopment have organised their development function using *work teams* (Cohen, 1993). Akldevelopment have a large development function; six work teams. In contrast, Meldevelopment have a small development function; it is organised into a single work team. Hence, the agile *work teams* (development teams) are organised based on fixed development capacity.

Both organisations allocated a stable number of fulltime individuals and a team authority to their work teams. They are autonomous work teams; the company executives have no involvement in their operations and decision making. This is delegated to the team authority, engineering manager at Akldevelopment and director of software engineering at Meldevelopment. As described by Crocitto & Youssef (2003), agility requires managers to accept a more participative decision-making process through employee empowerment.

The agile team is one of Akldevelopment's *work teams* and was allocated a fixed number of individuals (twenty-four). At Meldevelopment too, the *work team* was allocated stable number of fulltime engineers (twenty-four) and two project managers.

With both, the project management and decision-making is delegated to the entire team. These agile work teams have *empowering leadership* (Stewart, 2006), where they are *self-managing* and *self-directing* (Cohen & Bailey, 1997). The team ownership and commitment is achieved through *empowering leadership style* (Manz & Sims Jr., 1991).

These are successful work teams since the individuals adapted to *performing a wide variety* of development tasks (Cohen & Bailey, 1997). A key factor for this adaptation is that these *tasks* are *meaningful* to the engineers (Campion, Medsker, & Higgs, 1993).

These tasks at Akldevelopment and Meldevelopment provide engineers with *skills variety, task identity, and task significance* (Stewart, 2006). At both organisations, the manual acceptance tests and technical writing are not part of their engineering tasks. The functional setup driving agile adaptation to have fluid software engineering roles must not incorporate repetitive and mundane tasks, which does not enhance technical skills.

At Akldevelopment and Meldevelopment, their strategy was to adapt their engineers with heterogeneous skill sets so that development tasks can be allocated to sub-teams. This adaptation enables them to have *high performance* at *work team* levels (Stewart, 2006). As a result, these work teams are continuously implementing new projects.

At both organisations, the work teams make *dynamic adaptation* of their single team structure into several sub-teams to assign individuals to development projects. This adaptation of team structure is *mutually accepted* at both organisations. For both, this philosophy for their work allocations is an important *agile method fragment*. The sub-teams are allocated different parts of a project. These are project teams, where a part of a project is like a project to a sub-team (Cohen & Bailey, 1997). This adaptation of development teams is critical in agile development to simultaneously implement different parts of projects, reducing the implementation timeframe.

At Akldevelopment and Meldevelopment, adapting the development function into small-sized sub-teams contributes to high performance. Akldevelopment usually adapts to have 4 to 6 engineers per sub-team and Meldevelopment adapts to have up to seven engineers per sub-team. As described Wheelan & McKeage (1993), small teams are more effective and productive. At both organisations, this adaptation of development function facilitates effective interaction among individuals. The co-location of on-site customers, agile testers and usability engineers also allows engineers to swiftly seek further *assistance* on tasks (Lee, 1997). This functional setup adaptation factor is critical in agile development, since it provides flexibility to adapt sub-team sizes, enhancing team productivity.

Their project teams are *temporary* structures, existing only for the duration of their projects (Cohen & Bailey, 1997). At both organisations, these teams have cross-functional structures; at Akldevelopment each is allocated a product analyst whilst the usability expertise and domain expertise are shared. At Meldevelopment product

development managers, domain experts, usability and testing expertise are shared (Ford & Randolph, 1992). This enables them to achieve *technical excellence* in development projects (Kolodny, 1981). This functional setup adaptation factor is critical in agile development, since it provides flexibility to adapt the development function appropriately and according to the business needs of organisations.

At both organisations, projects allocated to work teams are *new developments* (Mankin, Susan, & Bikson, 1996). Therefore, their sub-teams require *adaptive performance* to solve new problems, deal with uncertainty and learn new skills or tasks. Each project has a new set of sub-teams. This ensures that engineers are not *restricting and isolating* themselves from information, help or learning (Katz, 1982).

For both organisations, their agile *work teams* accumulate a wealth of development expertise (Ford & Randolph, 1992). Their *work teams* are permanent, allowing development experience, skills and knowledge to be retained within the group. They continuously enhance their *transactive memory* (Wegner, 1986). Help and advice is always available in their setup. *Transactive memory* is the team memory system for collective encoding, storing, and retrieving of development knowledge.

At Akldevelopment and Meldevelopment, a participative *and collaborative* approach enables the sub-teams to have high performance (Ford & Randolph, 1992). They have effective project *communication* at the work team level through regular and joint daily stand-up meetings including various planning (backlog/iteration/sprint) and review (sprint/iteration/project/release) meetings (Stevens & Campion, 1994). This enables sub-team project participation at the work team level. Similarly, joint product backlog planning and co-location in a single workspace enables them to spontaneously collaborate at the work team level.

At both organisations, the *communication style* required *behavioural adaptations;* engineers adapted with *intrapersonal* (Edmondson & Smith, 2006) and *interpersonal* (Pulakos et al., 2000) competencies. The *intrapersonal competency* is extremely valuable; it enable engineers to listen and respond to situations in their work team environment.

The development team meetings have an open style of communication, where it is *informal*, *comfortable* and with *no personal tensions* (Argyris, 1966). Engineers are

open, show interest in other ideas, ask questions and consider issues from other engineers' perspectives (Stevens & Campion, 1994). Their communication style also is *supportive* (Burleson, 2009). Their meetings are *task-oriented* (Stevens & Campion, 1994). This style of communication helps to avoid defensive responses during their review meetings, making *objective* evaluations of their short development cycles (Gibb, 1961).

The above analysis highlights functional setup as critical factor driving the adaptation of the structures of the software engineering department. This ability to adapt engineering structures distinguishes agile software development from other approaches. For this reason, the size of IS department in Fitzgerald's (1998) adaptation framework is modified to incorporate functional setup as a sub-factor (Figure 73).

6.3.2.2 Informal engineering roles

Both Akldevelopment and Meldevelopment made a *dynamic adaptation* of their sub-team structures to create leadership roles. The team and technical leader roles are *informal* leadership role that the sub-teams adapt with when implementing projects (Day, Gronn, & Salas, 2004). At both organisations, these were *mutually accepted* adaptations.

At Akldevelopment, the appointments for these two roles are temporary (for a release or a project) and also function as engineers in their sub-teams. In contrast, at Meldevelopment the appointments for informal team leadership roles are both permanent and temporary. The product development managers function as permanent informal team leaders, performing in multiple roles. Senior or principal engineers are appointed to informal team or technical leader roles for projects, also performing in multiple roles.

At both organisations, the senior engineers are given the informal leadership roles since they have more project management experience in their work team environments. Through their guidance, the sub-teams deal with *adaptive challenges* (time-boxed meetings, moderate between engineers, organise backlogs, and provide information for decision making) (Drath, 2001). As described by Jamail (2009), leadership roles bring knowledge creating confidence in the team. Hence, these informal leadership roles are

critical in agile development to guide and facilitate tasks, maintaining a constant pace for development with sub-teams.

These two informal leadership roles have defined role expectations, which positively influence their *sub-team performance*; they facilitate tasks, champion new development, coach, and motivate to take care of team issues to meet delivery commitments (Zaccaro & Klimoski, 2002). At both organisations, this informal leadership emerges as a result of their engineers sharing project management and development tasks.

In their sub-teams engineers *participate* in the *leadership process* for collective decision making (Day, Gronn, & Salas, 2004). The leadership and team performance influence each other rather than being driven by their leadership roles only (Marks, Mathieu, & Zaccaro, 2001).

This informal structure is a result of the *interrelationship* of the engineers in their sub-teams (Day, Gronn, & Salas, 2004). At Akldevelopment and Meldevelopment, they collectively do product (strategic vision plans) and project plans (release, sprint/iteration and daily plans) including the monitoring tasks (plotting the burn-down chart).

At Akldevelopment, they adapt their pair programming practice to have a daily partner change and at Meldevelopment, they have compulsory design phase participation; this ensures a combined effort of engineers. This, at both organisations drives *collective behaviour* and creates leadership resource (Ellemers, De Gilder, & Haslam, 2004).

At both organisations, the individuals in these roles are *task- and relationship-oriented* engineers (Gratton & Erickson, 2007). These engineers require *behavioural adaptation* to perform their multiple roles; they adapt with *reactive behaviour* to switch between engineering and leadership roles and vice-versa (Dawis & Lofquist, 1984).

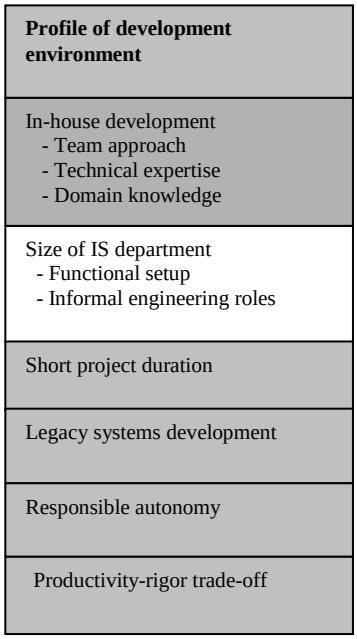
At Akldevelopment, the agile work team made further *dynamic adaptations* with other informal structures such as a BuildMaster and coaching roles. These were mutually accepted adaptations. These two new structures resulted from the expansion of the agile work team. They maintain their high performance and ensured quality was not comprised as their development capacity grew.

At both Akldevelopment and Meldevelopment, the coaching role reflects the contribution of the senior engineers to their high team *performance* (Edmondson, 1999).

This is another of the informal *leadership behaviours* that the senior engineers have to adapt to (Wageman, 2001). At both organisations, it is a *consultative* and *educational* role in the sub-teams (Hackman & Wageman, 2005).

The above analysis highlights informal engineering roles as a critical factor driving the adaptation of agile development teams. This ability to adapt with informal roles distinguishes agile software development from other approaches. For this reason, size of IS department in Fitzgerald's (1998) adaptation framework is modified to also incorporate informal engineering roles as a sub-factor (Figure 73).

Figure 73 Sub-factors for size of IS department



6.3.3 Short project duration

This factor enables adaptation of their project deliverables into shorter cycles. This short project duration factor drives adaptation to achieve the shortest time period for feature delivery in a market-driven environment. Analysis of data collected from Akldevelopment and Meldevelopment suggests that short cycles are a sub-factor associated with short project duration.

6.3.3.1 Short development cycle

Akldevelopment and Meldevelopment both now have shorter duration projects (one to six months) with their agile approach. Akldevelopment adapted to shorter duration projects by reducing their scope. This was achieved by adapting product planning

practice to identifying each major business value feature as a high-level requirement; this was a *dynamic adaptation*. At Akldevelopment, projects implement one or two high-level requirements. In contrast, Meldevelopment always had shorter duration projects but they also adapted their product planning practice to identifying each major business value feature as a high-level requirement; this was a *dynamic adaptation*. This planning adaptation also helped Meldevelopment to allocate one or two high-level requirements per project. Using high-level requirements, projects are much easier to break into smaller parts to have simultaneous implementation by sub-teams, reducing their implementation time.

This high-level requirement adaptation is a critical factor for achieving shorter development cycles for having frequent market releases. Adapting with high-level requirement practices enabled both organisations to reduce the duration between releases. Akldevelopment now has four month cycles, down from eighteen months. Meldevelopment always had a major release every three months but is now capable of even shorter releases. This short development cycles make agile development different from others.

The short duration projects are allocated with a few high-level requirements for implementation. This approach creates a project portfolio; a market-driven prioritised list of projects which enables the highest value project to be implement at any point in time. These projects incrementally add new features to expand Akldevelopment and Meldevelopment products. All developments are feature-driven; a new feature identified as a high-level requirement is split into number of independent stories or tasks to be iteratively implemented. As described by Vähänitty (2005), this enables focusing on features one at a time to analyse, implement, measure progress, and get feedback for the early detection and correction of errors.

The shorter duration projects reinforced Akldevelopment and Meldevelopment's strategic business advantage at the marketplace. They now make new innovations available *faster* than their competitors (Nagel & Bhargava, 1994). As such, *rapid introduction* of their new features makes their products higher performing (Duguay, Landry, & Pasin, 1997), increasing the *customer-perceived value* of their products (Goldman, Nagel, & Preies, 1995).

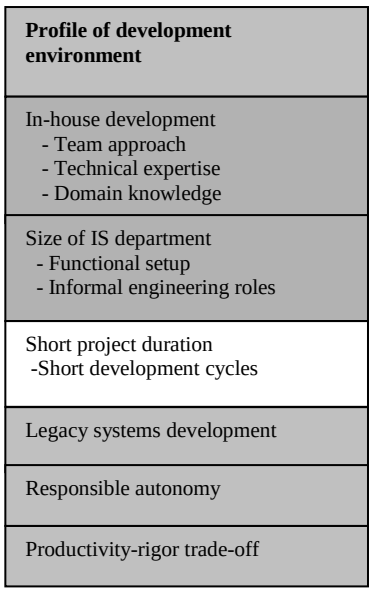
The short-term projects re-aligned Akldevelopment and Meldevelopment's business strategies with their market-orientated product requirements, becoming *customer focused* (Hadcroft & Jarratt, 2007). Their agile work teams adapt with market *responsive* behaviour; an *incremental and iterative* approach allows aligning their function with the business strategy (Hadcroft & Jarratt, 2007).

Both organisations had further *dynamic adaptations* of their short development cycle times, reducing the duration between release cycles. At both companies, the adaptation of their project deliverables with shorter time-frames enable a part of a major feature to be ready for production at a client site. This is achieved through the sprint cycles at Akldevelopment and iteration release cycles at Meldevelopment.

Akldevelopment adapted to three week sprint cycles before adapting to shorter, one week sprint cycles. Meldevelopment too adapted to three week iteration cycles and to shorter, one week cycles to meet their business commitments. Later, they adapted back to two week iteration release cycles. As described Yusuf, Sarhadi, & Gunasekaran (1999), adapting to shorter development cycle times is one of the key attributes of an agile organisation. At both organisations, these were mutually accepted adaptations.

The above analysis highlights short development cycles as a critical factor for driving the adaptation to achieve frequent releases. For this reason, the short project adaptation factor in Fitzgerald's (1998) adaptation framework is modified to incorporate short development cycles as a sub-factor (Figure 74).

Figure 74 Sub-factor for short project duration



6.3.4 Legacy systems development

This factor enables adaptation of their agile approach to enable product development driven by business value projects, having the highest value projects for implementation in a market-driven environment. Analysis of data collected from both Akldevelopment and Meldevelopment suggests that business value is a sub-factor associated with legacy systems development driving their agile approach adaptations.

6.3.4.1 Business value projects

Akldevelopment and Meldevelopment have *development flexibility* (Vãzquez-Bustelo, Avella, & Fernãndez, 2007). Using the agile approach, they implement projects which implement new features and also projects that enhance their legacy features. Adapting to improve any agile practice to implement new features also improves their approach for legacy developments.

Both organisations have high-level planning practices to determine the business value of any proposed enhancement of features. Such features need to have the highest priorities in the portfolio to be allocated as a project. The high-level planning practice provides them both with a transparent indication of the viability of such projects. This high-level planning makes the agile approach different from the other development approaches. Implementing the highest business value project to deliver features enables strategic advantage at the marketplace.

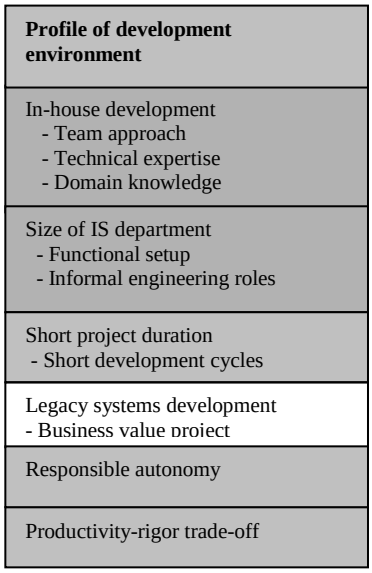
At Akldevelopment improvements on features are mostly market-driven based on investigation by product managers or on requests by marketing individuals. However, existing clients also request improvements, which are given higher priorities. At Akldevelopment, they *adapt* how they assign priorities to determine the business value of features proposed for improvements. At Meldevelopment, they also adapt on how they assign priorities of features proposed for improvements to determine their business value. At times priorities are driven by customer requests, according to product strategy or when such changes were requested. This priority setting at both organisations was a *dynamic adaptation* of their product planning practice and an important *agile method fragment.* At both organisations, these were mutually accepted adaptations.

The business value project is a critical agile adaptation factor, providing development flexibility for both organisations and allowing for their *responsive behaviour* in a market-driven environment (Sherehiy, Karwowski, & Layer, 2007). This adaptation

factor enables them to deliver high value features according to their business objectives; existing customer preference or new market demands.

The above analysis highlights business value project as a critical agile adaptation factor to deliver high value features at the marketplace. For this reason, the legacy systems development adaptation factor in Fitzgerald's (1998) adaptation framework is modified to incorporate business value project as a sub-factor (Figure 75).

Figure 75 Sub-factor for legacy systems development



6.3.5 Responsible autonomy

This factor enables adaptation of their agile approach to make engineers fully responsible for product delivery. Analysis of data collected from Akldevelopment and Meldevelopment suggests that full empowerment is the factor that drives their agile approach adaptations for full engineer responsibility for feature delivery.

6.3.5.1 Full empowerment

At Akldevelopment and Meldevelopment, engineer *delegation* was a *dynamic adaptation* of their agile product development environment (Gunasekaran, 1999). Initially, engineers were delegated for project planning and monitoring including agile process improvements, having full empowerment in projects. As described by Duguay, Landry, & Pasin (1997), a key agile development trait is team empowerment. This level of engineer empowerment makes agile development different from other development approaches. Engineer empowerment is an important factor for their *high team performance* and is an important *agile method fragment* (Ulich & Weber, 1996).

However, engineers at both organisations do not assign implementation priorities, but they are delegated to negotiate them for technical reasons. Priority setting for implementation is a business decision (market-driven). Hence, the full engineer empowerment with agile development is tied with the *tasks* that define the software engineering roles at project level (Sherehiy, Karwowski, & Layer, 2007). Full empowerment is a critical agile adaptation factor so that engineer delegation at project level can be adapted. This adaptation ensures that decisions made at development level do not impact business goals and objectives or decisions are not solely being made based on engineering perspectives only.

With this engineer delegation practice at Akldevelopment and Meldevelopment, development teams now have less *adherence* to *authority* and more *control* where engineers make quicker decisions since they *spontaneously collaborate* to deal with project issues and problems (Dyer & Shafer, 2003). These delegations make them responsible for timely deliverables and quality of the features.

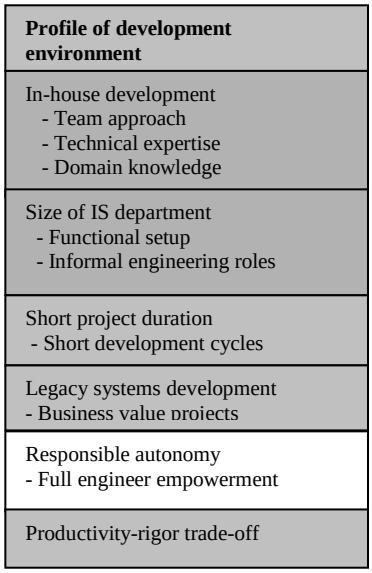
Both organisations adapted full empowerment practice with a *wider span of control*; at Akldevelopment engineers can re-negotiate story/task estimates during their sprint/iteration planning meetings (Sherehiy, Karwowski, & Layer, 2007). At Meldevelopment, engineers provide task estimates during the design phase; some tasks are poorly estimated, these are mostly underestimated. They had a *dynamic adaptation* of this practice where the on-site customers collaborate to re-estimate, if they feel that a task is under-estimated. At both organisations, the re-negotiation practice is *integrative;* a *win-win situation* for engineers and on-site customers (Lewicki, Barry, & Saunders, 1995). For both, this re-estimation practice is an important agile *method fragment* enabling more accurate estimates. At both organisations, this was a *mutually accepted* adaptation.

Re-negotiation practice in agile development environment is critical to enable effective team effort in a market-driven environment involving engineers, management and other functional units. Hence, full empowerment is a critical agile adaptation factor so that engineer delegation at project level is adapted with re-negotiation practice for ensuring organisation-wide development support.

Another *dynamic adaptation* of the full empowerment practice allowing engineers *a* wider span of control is that the engineers are delegated with hiring of new engineers for their team at both organisations. This mutually accepted adaptation makes them responsible for selecting the best candidate; individuals who they think are most capable of swiftly adopting their agile culture for software development. At Meldevelopment, engineers are also delegated with product ownership to do further research to improve their products. Full empowerment is a critical agile adaptation factor so that engineer delegation is further adapted, making them fully accountable for product development and delivery.

The above analysis highlights full engineer empowerment as a critical agile adaptation factor to make them accountable for delivering features. For this reason, the responsible autonomy adaptation factor in Fitzgerald's (1998) adaptation framework is modified to incorporate full engineer empowerment as a sub-factor (Figure 76).

Figure 76 Sub-factor for responsible autonomy



6.3.6 Productivity and rigour trade-off

This factor enables a highly productive product development environment without compromising rigour. Analysis of data collected from Akldevelopment and Meldevelopment suggests that both productivity and rigour (for quality) are critical in their agile development environment, having tools and practices (to enhance team productivity and quality) as a sub-factor.

6.3.6.1 Productivity

At both Akldevelopment and Meldevelopment, their agile work teams adopted practices and tools to enhance team productivity. Their development effort is team-based rather than separate individual efforts where productivity is measured based on their team effort. This removes competitive behaviour while enforcing the collective behaviour required for their market-driven product environments. This practice to measure productivity based on team effort makes agile development different from other development approaches. Team effort is an important *agile method fragment* for both organisations.

At both organisations, productivity is measured against their release commitments. In a market-driven environment, strategic releases enable market leadership. The *delivery performance* of their agile work teams determines their productivity (Ward, Bicklord, & Leong, 1996). At both organisations, the agile work teams continuously deliver their release commitments; this reliability measures the agile work team's productivity.

A key practice for delivery performance at the two organisations is their short development cycles, providing the measure of their delivery speed (at Akldevelopment- weekly sprint cycles and at Meldevelopment- two weekly iteration cycles). In agile development environment, the short development cycles determine the productivity capability of development teams.

At Akldevelopment, they target practices such as sprint planning, team work area and co-location, pair programming, minimum design, and good tool support to enhance their team productivity. At Meldevelopment, they also target iteration planning and good tool support practices to enhance their team productivity. However, they also target good project visions; the likely reason being that their projects are more fluid in nature.

These practices at two organisations are adapted with *generative rules* (Highsmith, 2002). At both organisations, this was *mutually accepted* adaptation.
This generative rules practice was a *dynamic adaptation and* an important *agile method fragment*. At Akldevelopment, they have rules such as an effective sprint plan, which clearly states the stories the team commits to deliver. The team work area and co-location practice ensures that all team members are in their team work area, requiring being (physically) in this area for their engineering hours. The pair programming

practice has a driver and navigator rule for a pair to remain focused on their task. At Akldevelopment, they adapted their story sizing practice with a *generative rule*; stories to be sized to be no more than two days for implementation, enabling more stories to be delivered.

At Meldevelopment, the rule of an effective iteration plan clearly states the tasks committed to be delivered. The practice for good tool support ensures that their engineers do not wait for long to find broken builds; the rule is that they must have multiple build systems. The practice of good project vision has a rule which requires the product development managers to thoroughly communicate the vision plans at development level. Both organisations also adapted their tool support used to create backlogs to have a more productive one.

The above analysis highlights tools and practice as a critical agile adaptation factor to enhance team productivity in a market-driven product development environment. For this reason, the productivity and rigour trade-off factor in Fitzgerald's (1998) adaptation framework is modified to incorporate tools and practices as a sub-factor (Figure 77).

6.3.6.2 Rigor: quality

Akldevelopment and Meldevelopment have rigorous testing procedures with their *agile development* to ensure the *highest quality* for new features (Duguay, Landry, & Pasin, 1997). Agile development does not comprise product *quality* (Yusuf, Sarhadi, & Gunasekaran, 1999).

At both organisations, most of the testing and quality assurance tasks are incorporated within their development function. Their development environments are adapted with *new technologies* which allow automation of these tasks in their short development cycles, giving them *agility* for frequent releases (Duguay, Landry, & Pasin, 1997). Acceptance testing prior to major releases is done manually, giving a real user perspective to enhance the quality of their product. In a market-driven environment, quality of products is another critical factor, ensuring market leadership.

Tools that ensure quality of their products include version control systems (VCS) that allow for automation of a source code repository of their products, new features, and simulated features (for trialling at client sites); automated trigger systems for running

various tests and regression tests, automated continuous integration and build systems; and automated testing systems- code style tests, unit tests and systems tests (acceptance tests). As described by Spinellis (2005), adopting a VCS is an important technological improvement for software development.

These tools also support both organisations' test driven development (TDD) practice; TDD too is a *dynamic adaptation* of their agile development environments. Both organisations adapted with an automated testing framework for a better performance; a XUnit testing framework is made available for engineers to write unit tests.

Both organisations also adapted with Integrated Development Environment (IDE) tools for development. An IDE provides technologies such as code editors, compiler and build automation tools for implementing and running unit and acceptance tests on development machines (Chen & Marx, 2005). These development tools enable implementation speed and also ensure the quality of features.

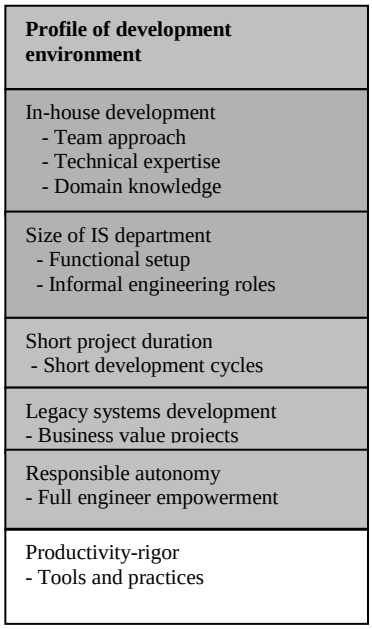
At both organisations their agile work teams adapted their acceptance and unit testing practices. At Akldevelopment, they adapted Selenium (acceptance) tests with Java code rather than HTML code for refactoring purposes, improving quality attributes for higher performance of Selenium tests. They also adapted unit tests with mutation testing, checking the quality of their unit tests.

At Meldevelopment, they adapted their testing practice with manual tests to ensure the quality of their products; some unit tests cannot be automated. The acceptance test is adapted to be done by proxy-customers to deliver higher quality products. At both organisations, these were *mutually accepted* and *dynamic adaptations,* and additions to their *agile method fragments.*

At Akldevelopment and Meldevelopment, trade-off is minimised between productivity and rigour. With their agile development scope is the trade-off; the lowest priority stories or tasks may not be implemented.

The above analysis highlights tools and practice as a critical agile adaptation factor to deliver implemented features in short development cycles, enhancing product and quality in a market-driven product development environment. For this reason, the productivity and rigour trade-off factor in Fitzgerald's (1998) adaptation framework is modified to incorporate tools and practices as a sub-factor (Figure 77).

Figure 77 Sub-factor for productivity and rigour



6.4 Overt factors

The cross-case analysis of the overt factors is now provided: project management: improved visibility, reduced risk; reduction of variety and complexity; economic- skill specialisation and division of labour; epistemological- transfer of knowledge, template for inexperienced developers, and learning from past projects; and facilitation of intercommunication among developers.

6.4.1 Project management

This factor enables adaptation of their agile approach to improve visibility and to reduce development risks.

6.4.1.1 Improved visibility

Akldevelopment and Meldevelopment adopted five different levels of agile planning; vision, roadmap, backlog, sprint and daily plans. For both, these planning practice were extremely important *agile method fragments.*

At Akldevelopment and Meldevelopment, the vision plan provides project visibility at functional levels; management, business and development. For both organisations, the vision plan establishes a *strategic vision* of new features organisation-wide (Sharifi & Zhang, 1999). As described by Smits (2006), a product vision plan is amongst the 5 different levels of agile plans.

Both organisations use a collective decision making approach (Akldevelopment has a product strategy group and a steering committee while Meldevelopment has a product planning team) to allocate development projects based on vision plans. The members are representatives of different functional teams. They collaborate and communicate within their units, creating visibility of proposed features.

Both organisations use on-site customers to communicate vision plans. This cross-functional collaboration makes projects more visible at the development level. The engineers have support to gain a better understanding of high-level requirements. The on-site customer is a major factor in providing implementation speed with agile development when compared to other approaches.

This vision planning artefact identifying proposed innovations is developed based on consultation with clients and industry and through market research, including having internal inputs (sales, marketing and development functions). The vision plan enables both organisations to ascertain the business value of features, enforcing organisational *learning behaviour* for effective *strategic decision making* (Yusuf, Sarhadi, & Gunasekaran, 1999). Vision planning is a critical agile practice, enabling strategic project decisions. This planning practice installs confidence for making funding decisions and also enables learning amongst the key stakeholders of innovations, providing a better project clarity organisation-wide.

At both organisations, the vision planning practice had *dynamic adaptations*. Akldevelopment adapted it by creating the product strategy group, steering committee and product analyst roles while Meldevelopment adapted product manager roles with technical skills and with product development manager roles. At both organisations, these were *mutually accepted* adaptations.

Most importantly, both organisations had *adaptive performance* to successfully use this practice (Pulakos et al., 2000). Their management made an *emotional adaptation* to have a *positive* reaction and accept employee participation in strategic decision making (Allworth & Hesketh, 1999). Their agile work teams made a *cognitive adaptation* to become product focused (Allworth & Hesketh, 1999). Engineers provide business cases for proposed innovations and enhancements - they solve costing problems to determine product expansions. The business function (sales and marketing) also made *cognitive adaptation*, having a better coordination to deliver improvements and additions that

they promise to clients. Hence, the vision plan is a critical agile adaptation factor for both organisations.

The vision plan makes agile development different from other approaches, enhancing project clarity. For this reason, the project management: improved visibility factor in Fitzgerald's (1998) adaptation framework is modified to incorporate product plan (vision) as a sub-factor (Figure 78).

At Akldevelopment and Meldevelopment, the short development cycle plans are other artefacts that provide clear visibility of projects. This plan provides regular and frequent information on features that will be implemented in the next cycle and also enables provision of weekly or two weekly implementation reports on project status for the stakeholders. At the work team level, the plan clearly shows the team deliverable for a short development cycle.

This *short range planning* (short cycle planning) practice allows swiftly dealing with requests for change to story priorities for implementation (Tersine & Wacker, 2000). This plan provides flexibility to quickly modify it during short development cycles. Changes are easily made since the short cycles consist of a small number of features that are easily reshuffled.

The short development cycle plans provide both organisations with project visibility. First, their agile work team plans for the next short cycle a few days before its start, usually on the last day of their current iteration. They have an accurate picture of their development capacity for the next cycle and accordingly make delivery commitments.

Second, this planning is a collaborative effort; engineers take part as a project team. This effort enables a collective perspective on sizes and estimates of features. This reassessment improves sizes and estimates of features. The collaborative effort ensures team *cohesiveness, interdependence, acceptance* and *ownership* of the short development cycle plans (Forsyth, 1990).

Third, it is a cross-functional effort, involving negotiations with on site customers. At both organisations, they use this approach to obtain *genuine resolutions,* driven by *trust and openness* to get the best possible sizes and estimates acceptable for both parties (Stevens & Campion, 1994). For these reasons, short range planning practice is critical for providing project visibility not only at development level but also organisation-wide.

The short development cycle planning practice makes agile development different from other approaches.

At Akldevelopment and Meldevelopment, they made a *dynamic adaptation* in relation to their short cycle planning practice. They both adapted with a planning poker game enabling the committed estimates to be well thought-out by the team (Greening, 2002). As described by Moløkken-Østvold & Haugen (2007), this approach provides more realistic estimates while reducing the impact of social comparisons. Adapting estimation practice to provide reliable estimates is critical in agile development, since other stakeholders plan their work based on dates the implemented features are expected to be available. At both organisations, this was *mutually accepted* adaptation.

Akldevelopment had a few other *dynamic adaptations* to their short cycle planning practice. Their development priority setting method has high, medium and low classifications. They adapted to avoid assigning two stories with the same priority. Their short development cycles now are quicker to plan. Akldevelopment also had another *dynamic adaptation* to their short cycle planning practice. The story estimates are now provided in points rather than hours where a point is equivalent to an ideal day. This approach for estimation allows for a buffer for considering technological issues with story implementations making their estimates more accurate and achievable.

Hence, short cycle planning is also a critical agile adaptation factor to improve project visibility organisation-wide. For this reason, the project management: improved visibility factor in Fitzgerald's (1998) adaptation framework is modified to incorporate project plan (short cycle) as another sub-factor (Figure 78).

6.4.1.2 Reduced risk

Both Akldevelopment and Meldevelopment have their own unique issues and risk factors that impact their development projects. At Akldevelopment, these are unfamiliar technology, outdated backlog plan and missing out low priority story implementation, while at Meldevelopment they are unclear scope, product quality and inaccurate estimates. For both organisations, the agile approach provides them with flexibility to deal with these risks. They both made *dynamic adaptations* through *generative rules* by adapting their development environment with new practices (Highsmith, 2002).

For both organisations, inaccurate estimates are one of their risk factors. Akldevelopment adapted with the ‘spike’ practice; a time-boxed approach for learning and researching stories with technological influences. At Meldevelopment, they adapted with a re-negotiation practice to provide realistic estimates, allowing buffering an estimate if it appears to be an underestimate.

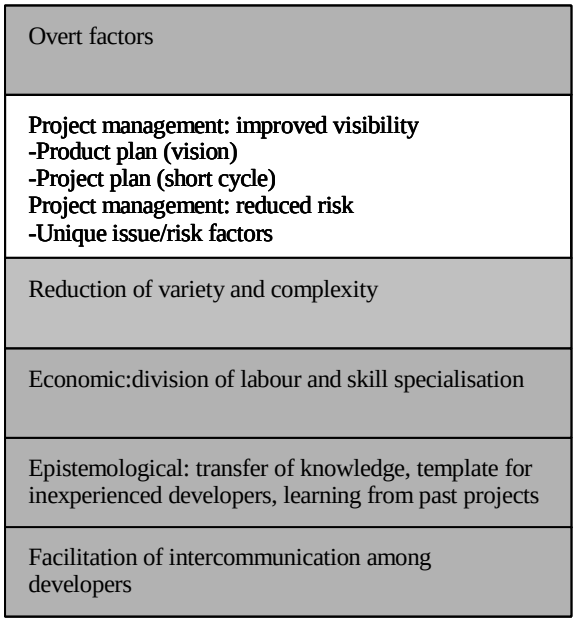
There are other instances for both organisations of using *generative rules* to counter their unique issues and risk factors. Akldevelopment has a rule to re-evaluate product backlogs; over time with new knowledge and information, this practice ensures the accuracy of size and estimate of stories. Meldevelopment, to establish a stable vision, has a rule requiring doing only “sufficient enough” planning for high-level requirements, helping to avoid incurring high costs if it turns out to be a low value or unwarranted feature.

Meldevelopment also has another rule requiring a non-engineering individual to test implemented features to ensure quality. At Akldevelopment, they have another rule to invoke a shorter sprint cycle (less than one week), to implement missed out low priority stories to enhance the product value. With both organisations, these were their other unique issue and risk with product development.

At both organisations, these adaptations to deal with their unique issues and risks required the *responsive behaviour* of their agile teams. These adaptations were *mutually accepted* at both organisations.

Agile development provides both organisations the flexibility to adapt their agile approach with relevant practices to deal with their unique issues and risks. This flexibility to adapt with new practices makes agile development different from others. Hence, the project management: reduced risk factor in Fitzgerald’s (1998) adaptation framework is modified to incorporate unique issue/risk as a sub-factor (Figure 78).

Figure 78 Sub-factors for project management



6.4.2 Reduction of variety and complexity

This factor enables adaptation of agile approaches to reduce variety and complexity with their development projects.

At Akldevelopment and Meldevelopment, the roadmap, release and daily planning practices are key part of their agile development environment. These *method fragments* enable agile work teams to reduce the variety and complexity of their vision plans.

Akldevelopment's roadmap plan covers a year, which has a collection of likely features and their market release dates. In contrast, Meldevelopment's roadmap plan includes features up to eighteen months ahead. Due to several product lines and smaller development capacity, they have a bigger roadmap plan to add new innovations to their product suite.

The vision plans contain the high-level features. Using the high-level features and based on the roadmap plan, release backlogs are created. This backlog planning practice is a key *agile method fragment* that gives both organisations implementation clarity. Meldevelopment also has a design phase as an important *agile method fragment* for this purpose. This backlog planning practice makes agile development different from other approaches.

The backlog creation enables the reduction of variety and complexity of the high-level features. These features are split and sized into stories or tasks for implementation. The reduction of variety and complexity is further achieved through setting priorities for their orderly implementation, creating a release backlog. Priority setting is a critical agile practice to deliver appropriate features in a market-driven environment where the requirements and customer preferences change frequently.

Due to low value or time factors low priority tasks or stories may not be implemented. At both organisations, this priority setting practice is adapted to determine the highest value features. At both organisations, this was a dynamic adaptation and important *agile method fragment*. Priority is critical to *eliminate wastage*, have implementation flexibility and to maximise the development speed (Hormozi, 2001) in agile development environment.

This adaptation at both organisations was *mutually accepted* requiring *responsive behaviour* of their agile teams to decide this adaptation, delivering features according to business goals and objectives.

Hence, backlog planning is a critical agile adaptation factor for reduction of variety and complexity of vision plans (high-level requirements). For this reason, reduction of variety and complexity factor in Fitzgerald's (1998) adaptation framework is modified to incorporate product plan (backlog) as a sub-factor (Figure 79).

At Akldevelopment and Meldevelopment, the reduction of variety and complexity of short development cycles is achieved through their daily plans (scrum planning and stand-up meetings). Short development cycles (sprint or iteration cycles) are used to implement a release backlog (sprint/iteration backlog). At Akldevelopment, scrum planning meetings are focused sessions that have a set agenda and are time-boxed. At Meldevelopment, this *agile method fragment* has an informal structure. This daily planning practice makes agile development different from other approaches.

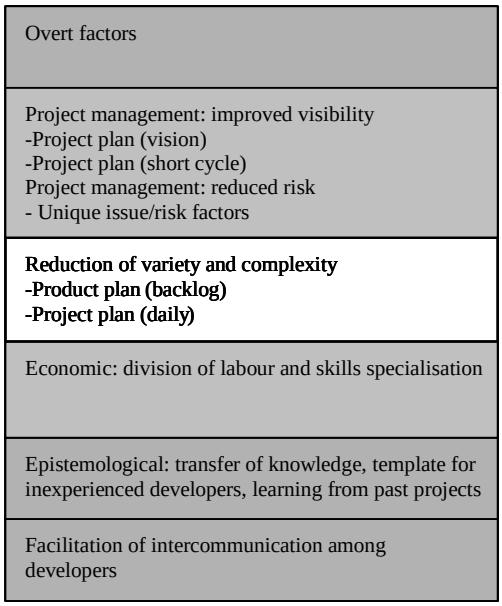
At both organisations, they made *dynamic adaptations* to their daily planning meetings, adapting to address implementation issues rather than the implementation status of stories. There is no value in providing implementation status as the co-location in the team work area provides daily progress status. The new format enables an immediate team awareness and swift action to fix implementation issues.

Akldevelopment and Meldevelopment made another *dynamic adaptation* of their daily planning meetings. At Akldevelopment, due to their doubled development capacity they adapted to having it in their sub-teams if they were on different projects or on distant parts of projects. Meldevelopment adapted to have daily planning meetings on as needs basis.

At both organisations, these adaptations were *mutually accepted* and required *responsive behaviour* of their agile teams to decide these adaptations.

Daily planning for story or task implementation is another important agile adaptation factor for reduction of variety and complexity of sprint or iteration backlogs. For this reason, the reduction of variety and complexity factor in Fitzgerald's (1998) adaptation framework is modified to incorporate project plan (daily) as a sub-factor (Figure 79).

Figure 79 Sub-factors for reduction of variety and complexity



6.4.3 Economic: division of labour and skill specialisation

This factor with its sub-factors enables adaptation of the development environment to allow business contribution at development level and achieve a general skill set to swiftly deliver new features.

6.4.3.1 Economic: division of labour

At Akldevelopment and Meldevelopment, the on-site customer roles enable business function contribution at the development level. This on-site customer role adaptation

created a division of labour within the development function with business and software engineering roles. These are generalist types of roles, performing a wide variety of tasks. This was a *dynamic adaptation* of their agile development environment.

At both organisations, this adaptation required *adaptive performance behaviour*. Individuals required adapting with interpersonal skills to collaborate on each others' functional tasks to implement features.

At both organisations, the business role (on-site customer) is the hub for the *external communication* of their work team's project status (Ancona & Caldwell, 1992), liaising with their product management, sales and marketing teams about implementation priorities and availability of implemented features. Simultaneous sales and marketing activities are possible, increasing both organisations' capability for marketing *speed* (Sarin & Mahajan, 2001).

Through this division of labour, *timely* product expertise and knowledge is provided to the engineers. This engineer support is a major factor that contributes to their work team's *performance* (Cummings, 2004; Ancona & Caldwell, 1992), including their high team performance in *innovation (implementation) capability, quality, delivery and planning* (Edmondson & Nembhard, 2009). At both organisations, this division of labour also enables the business function to manage product development, requiring on-site customers to adapt their product management tasks with project management skills. The business contribution for product development has a major economic impact. Quality, a *critical issue* is immediately evaluated and fixed on a daily basis rather than costly fixes at the end of projects (Jassawalla & Sashittal, 1999).

Another major economic impact through such division of labour is that the specialist project management role is no longer favourable for both organisations' product development. Through *adaptive performance and proactivity*, project management is a shared task between the business and engineering roles.

At Akldevelopment and Meldevelopment, the on-site customer performs in multiple roles at development level; product and domain expert, quality assurance engineer, and project manager. This is a generalist role with the ability to do a wide variety of tasks that provides the speed for decision making at development level.

At both organisations with division of labour, there was *a dynamic adaptation* of the engineering roles. These roles now perform tasks associated with project management, analysis, development and testing. This is achieved through engineers performing in multiple roles through *adaptive performance behaviour*. All these adaptations at both organisations were *mutually accepted*. The engineering roles have an economic impact at both organisations where specialist testing and project manager roles are no longer required. They still deliver quality as expected.

This division of labour makes agile development different from other approaches where generalist skills are a critical agile adaptation factor for having economical product development. For this reason, the division of labour factor in Fitzgerald's (1998) adaptation framework is modified to incorporate generalist as a sub-factor (Figure 80).

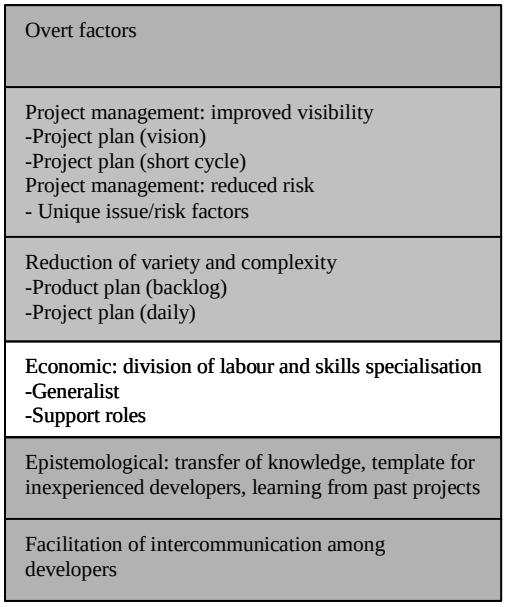
6.4.3.2 Economic: skills specialisation

At both organisations, their agile work teams adapted with specialist roles. These specialist roles are an important *agile method fragment* and a *dynamic adaptation* of their development function. However, these roles are support roles for engineers, collaborating on development tasks to enhance engineers' skill sets. This support role adaptation at both organisations was *mutually accepted*.

The agile tester is a supportive role that helps engineers and on-site customers to deal with quality assurance issues up-front on a daily basis. They also coach engineers on quality assurance skills. At both organisations, agile testers help engineers to minimise costly after-release fixes. At Akldevelopment, the usability engineer is another of their support roles that up-skill the software engineers to deal with the usability issues of features up-front, on a daily basis. Engineers required *adaptive performance behaviour* to learn new skills.

Hence, support roles is a sub-factor associated the economic: skill specialisation factor, providing coaching to enhance skills adaptations of both organisations' development functions to enable an economical product development environment. For this reason, the skill specialisation in Fitzgerald's (1998) adaptation framework is modified to incorporate support roles as a sub-factor (Figure 80).

Figure 80 economic: division of labour and skill specialisation



6.4.4 Epistemological role of methodology

At both Akldevelopment and Meldevelopment, this factor with its sub-factors enabled adaptation to allow learning in their product development environments.

6.4.4.1 Adapting the learning practice for transfer of knowledge

Akldevelopment and Meldevelopment adopted agile practices of teamwork and collaboration for learning and transfer of knowledge. These agile practices include on-the-job training where individuals learn new skills through task sharing.

This was further adapted by both organisations with mentoring practices for graduate engineers where a graduate engineer pairs with a senior engineer for a period of time. This practice shapes a graduate into an innovative and accomplished engineer to serve future organisational needs. At both organisations, an expert engineer exposes the unique development situations and provides guidance for problem solving, developing a graduate engineer's development expertise. This is according to Schwaber & Beedle (2002) that senior agile professionals must train and mentor junior or less skilled individuals.

Both organisations also adapted with support roles (on-site customers, usability engineers and agile testers), providing on the-job training for engineers to develop other skills needed for their projects. These adaptations were *mutually accepted* at both

organisations. The on the-job training practice was a *dynamic adaptation* of their agile development environment and an important *agile method fragment.* Most importantly, the engineers have swiftly developed multi-skills through their *adaptive performance behaviour*.

6.4.4.2 Adapting template for inexperienced developers

At Akldevelopment and Meldevelopment, their agile practices of teamwork and collaboration to learn and transfer knowledge were also adapted with a template for inexperienced developers. This adaptation at both organisations involved incorporating tutorials (off-job training) for new recruits to learn about the product domain.

This *dynamic adaptation* supplemented their on the-job training practice. The *responsive* behaviour of both organisations' agile teams enabled them to decide this adaptation of their on the-job training practice, facilitating quick integration of new recruits into development teams. This off the-job training practice minimises the impact on team performance due to the diversion of the senior engineer's effort towards training and it also ensures that new recruits are immediately productive. This adaptation was *mutually accepted* at both organisations.

6.4.4.3 Adapting the practice to learn from past projects

Both Akldevelopment and Meldevelopment adopted the agile practice for regular reflections to learn from projects as a way to become more effective. This learning helps to improve team performance and enables them to make the development approach more effective on projects.

They both adapted their regular reflection practice. Akldevelopment adapted with an additional monthly project review meeting with their weekly sprint reviews to determine any emerging scope while Meldevelopment adapted with two weekly reflection meetings for complex projects with their reflection meetings for releases. This was a *dynamic adaptation* and important *agile method fragment* for their product development environment. Both have to adapt to continuously learn to become better at delivering quality products. This adaptation was *mutually accepted* at both organisations.

At both organisations, these practices contribute to the high team performances. The engineers have the authority to deal with project implementations issues. They have *responsive* and *proactive behaviours* to identify and creatively solve any project issues.

Adapting learning practices is also critical in agile development to deliver features swiftly. Hence, the epistemological role of methodology factor in Fitzgerald's (1998) adaptation framework is also relevant for agile development.

6.4.5 Facilitation of intercommunication among developers

At both Akldevelopment and Meldevelopment, this factor enabled adaptation to allow effective interaction in their product development environments.

At both organisations, their agile work teams adopted the agile practice of face-to-face interaction and communication. This practice was an important *agile method fragment* in their development environment. The face-to-face interaction practice makes agile development different from other approaches.

This required *adaptive performance behaviour* for interpersonal adaptation by the engineers, allowing spontaneous collaboration in their development environment. As such, they get immediate responses to any request for help, easy and instant access to the required information, and quick collective decision making to solve issues on the fly. The face-to-face interaction at both organisations is achieved through co-location and an open workspace. Both organisations have co-located their on-site customers, product managers, engineers, support roles (usability, agile testers, and technical writers) and the external leadership (the team authority). These are the key stakeholders that collaborate and work with one another on a daily basis to develop products. They all have some critical domain or technical information; co-location makes it easier and quicker for others to access it.

At Akldevelopment and Meldevelopment, an open workspace enables co-location, creating awareness of the physical presence of others and motivating for immediate collaboration. The workspace and co-location are two important *agile method fragments* for their face-to-face interaction practice.

The open workspace and co-location practices contribute immensely to their delivery speed; the informal coordination is faster and more effective, information required to solve issues is within their open workspace and easily accessible for everyone, the

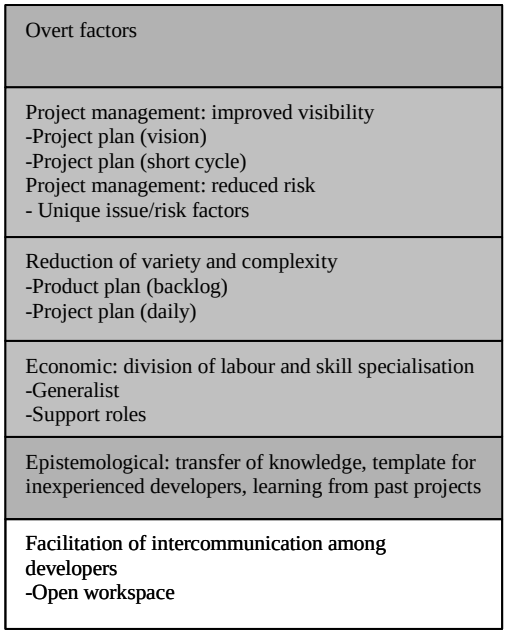
degree of cooperation is hugely enhanced, and builds a helping culture - any request for help is immediately responded to. Most importantly, the status of daily progress is visible to the entire team.

The workspace was adapted to co-locate sub-teams in their own area within the open workspace (Akldevelopment uses clusters while Meldevelopment uses partitions). This enhanced the informal coordination within the sub-teams while minimising the noise levels from one sub-team impacting the others. The open workspace was also adapted with tools and resources such as whiteboards, noticeboards and meeting rooms, encouraging effective interaction and collaboration.

For both organisations, the meeting rooms are the most productive resource for collective sub-team discussions, preventing interruptions from the other sub-teams during important discussions. The clusters and partitions, whiteboards, noticeboards and meeting rooms were *dynamic adaptations* of their open workspace and are important *agile method fragments*, contributing greatly to their high team productivity. These adaptations were *mutually accepted* at both organisations.

Hence, open workspace is a critical factor for achieving effective face-to-face interactions in an agile product environment. For this reason, the facilitation of intercommunication among developers factor in Fitzgerald's (1998) adaptation framework is modified to incorporate open workspace as a sub-factor (Figure 81).

Figure 81 Sub-factor for facilitation of intercommunication among developers



6.5 Covert factors

The cross-case analysis of the covert factors is now provided: comfort factor, legitimacy factor, aura of professionalism, confidence factor, audit trail and raise the profile of IS department.

6.5.1 Comfort factor

This covert factor is an important agile adaptation factor at Akldevelopment and Meldevelopment to minimise individual burn-outs by adapting their agile development environment with human resource practice. Clearly, this factor is related to one of the *agile principles* for software development, that *agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.* This was the main driver for the *dynamic adaptation* of their agile development environment with a *generative rule*- a 40 working hour week for engineers.

This adaptation through a human resource practice enforces a development regulation that minimises individual *burn-out* (Babakus, Cravens, Johnston, & Moncrief, 1999), providing a comfort factor for the engineers. The human resource practice is an important *agile method fragment* at both organisations, where this regulation enables high team performance. The engineers through high productivity meet their release deadlines without working long hours or weekends. At Meldevelopment, on very rare occasions they have worked extra hours, which was a norm with their previous process.

Preventing engineer burn-outs is critical in agile development, contributing towards high enthusiasm and motivation for teamwork through high-levels of engineer *interactions* (Jackson & Schuler, 1983). At both organisations, engineers consistently provide full *in-role* (functional tasks) and *extra-role* (cross-functional tasks) efforts to contribute to their collective team performance (Demerouti, Verbeke, & Bakker, 2005).

At both organisations, the human resource practice facilitates very low levels of engineer *absentees* (Firth & Britton, 1989), contributing to continuous delivery of their short cycle commitments even when they have reduced their sprint/iteration duration to the shortest possible cycle time. They have also increased their quality and quantity targets by regularly achieving zero-defects with releases and are able to implement more

stories/tasks in short cycles. Having the ability to get to the marketplace ahead of competitors on a continuous basis is critical for maintaining the product leadership.

This human resource practice also provides engineers with the opportunity for quality time away from their agile work environment, having balanced *work and home* environments (Hall, 1987). This is a critical factor which reduced their *staff turnover* (Jackson, Schwab, & Schuler, 1986), helping both organisations' agile work teams to *retain* and *attract* high calibre engineers (Podsakoff & MacKenzie, 1997). This human resource practice enables both organisations to achieve a dynamic core competency, requiring local development experiences and tacit knowledge to continue implementing new innovations.

Hence, human resource practice is a critical factor for achieving sustainable development in an agile product environment. For this reason, the comfort factor in Fitzgerald's (1998) adaptation framework is modified to incorporate human resource practice as a sub-factor (Figure 82).

Figure 82 Sub-factor for comfort factor

6.5.2 Legitimacy factor

This covert factor is an important agile adaptation factor at Akldevelopment and Meldevelopment for achieving effective cross-functional collaboration. This factor relates to one of the *agile principles* for software development, that *business people and developers must work together daily throughout the project.* At both organisations, they have *agile method fragments* that integrate the business and development functions' efforts to develop their products.

At Akldevelopment and Meldevelopment, for their business functions this is a legitimate approach because they manage the product development, test quality as

features are developed, can easily change implementation priorities, have real-time implementation status, have short cycle builds (implemented features in a sprint/iteration release cycle) to show new innovations to potential clients, and have simulated visions to test viability of proposed innovations. These are some of the key benefits that an agile approach provides their business function for them to identify and deliver strategic products, providing legitimacy for their agile approach. Hence, business benefits are critical to gain organisation-wide support for agile development. As such, both organisations have full-time business involvement in product development through the on-site customers.

For the engineers, the dynamism in their role gives them a wide variety of tasks. At both organisations, engineers make contributions in strategic decision making, have early knowledge of innovations, have full-time business support, negotiate schedules and plan their own work, carry out tasks as a team, learn skills and knowledge from one another, and make decisions about tools and improvements to their development practices. These benefits are critical to gain their support for agile development, providing legitimacy for their agile approach. At both organisations, the engineer benefits are achieved through their *agile method fragments* such as the engineering roles, team effort and empowerment.

At both organisations, agile development is a legitimate approach for the management. With their agile approach, projects have a clear business value for them to get innovations much quicker to the marketplace and they have strengthened their product development capability.

These benefits are achieved through their *agile method fragments* such as product planning (at Akldevelopment- strategy group, steering committee and at Meldevelopment- product planning team), shorter release cycles, and dynamic core competency. These *agile method fragments* provide legitimacy for their agile approach. At both organisations, cross-functional collaboration between their three functional units is facilitated by the agile approach, providing strategic benefits for their functional activities.

Both organisations provide bonding activities through social events (documented with in-house development under organisational factor). The social interactions encourage a shared belief and trust in their ability to achieve high value features and to acquire a

mutual understanding of one another's needs. The social interaction is an important *agile method fragment*, creating a *social capital* (Tansley & Newell, 2007) through which their cross-functional agile work teams adapt to become a cohesive unit. Social capital is the connections established through social events.

The legitimacy factor is also a reason behind the adoption of appropriate agile method practices and their adaptation through organisational and overt factors. Hence, any adaptation to achieve new or improved method fragments must be mutually acceptable and beneficial for the functional teams, without bringing any disadvantage to each of them. The organisational-wide acceptance is a critical factor for gaining support for agile development. For this reason, the legitimacy factor in Fitzgerald's (1998) adaptation framework is modified to incorporate organisational-wide support as a sub-factor (Figure 83).

Figure 83 Sub-factor for legitimacy factor

6.5.3. Aura of professionalism

The aura of professionalism is an important agile adaptation factor at Akldevelopment and Meldevelopment. This factor relates to one of the *agile principles* for software development; g*ive them the environment and support they need, and trust them to get the job done.* At both organisations, this was one of the main reasons for providing empowerment in their agile work team for them to have the ability to continuously deliver products ahead of their major competitors.

The engineer empowerment is a major *agile method* fragment for both organisations to ensure an aura of professionalism. The engineers themselves decided how they should work, including taking the responsibility for their practices and tools because they are collectively accountable for timely implementation of features, giving a full-

empowerment on development projects. This level of empowerment is critical with agile development, providing an aura of professionalism to make swift decisions and be able to collectively negotiate instead of development projects and tasks being forced upon the engineers.

Through empowerment, engineers plan their product backlogs and short development cycles more *effectively* (Cooke, 1994). From a technical perspective, they have better knowledge and skills than their team authority for producing reliable and achievable schedules. At both organisations, this empowerment practice is a key factor contributing for higher agile team performances. High development performance is most critical in a market-driven environment.

Such team performance is also due to the engineer empowerment to negotiate estimates; their collective autonomy allows for an increase in the availability of relevant information for providing more accurate estimates (Hollenbeck, Ilgen, LePine, Colquitt, & Hedlund, 1998). This also provides them with *motivation* to deliver their commitments (Latham, Winters, & Locke, 1994).

Another reason for the *higher team performance* is that the engineers are empowered with adaptation responsibility (Cohen & Ledford Jr., 1994). This makes engineers more *committed* to implementing and using new and improved method fragments that they themselves decide (Locke & Schweiger, 1979). At both organisations, they achieved shorter development cycles and the ability to analyse requirements into the smallest sized tasks.

At both organisations, they made a further *dynamic adaptation* of engineer empowerment practice. Engineers are given the responsibility for hiring new recruits for their agile work teams. The collective effort for achieving high team performance, their *team composition* is an important factor (Tesluk & Mathieu, 1999). Hence, it is only appropriate that the engineers themselves judge the temperament and capacity of the potential individuals for rapid integration into their agile work team and their ability to contribute to their team performance. Hence, the engineers are in the best position to match the personality traits and cognitive abilities of individuals with their requirements.

Full empowerment is a critical factor for achieving higher development team performance. For this reason, the aura of professionalism factor in Fitzgerald's (1998) adaptation framework is modified to incorporate full empowerment as a sub-factor (Figure 84).

Figure 84 Sub-factor for aura of professionalism

6.5.4 Confidence Factor

At Akldevelopment and Meldevelopment, this covert factor is an agile adaptation factor. Their agile work teams have adapted into a cohesive and high performing units through adoption and adaptation of appropriate agile practices (identified with organisational- profile of development environment and overt factors). This covert factor relates to several principles for agile software development; *deliver working software frequently, working software is the primary measure of progress, continuous attention to technical excellence and good design enhances agility,* and *at regular intervals, the team reflects on how to become more effective.* These were also the basis for adaptation at both organisations.

As a result of adapting with cross-functional teams, agile teams at both organisations function effectively on their development projects. They have adapted swiftly with their agile approach, developing into cohesive work teams.

At both organisations, the team authorities adapted with external leadership behaviour where they have become outward (product rather than project) focussed. As a result, the self-managing project teams have become a part of their core competency. The team authorities provided a supportive environment through empowerment, enabling the practice of self-managing teams. At Akldevelopment and Meldevelopment, the

emotional adaptation by the team authorities is due to confidence in their team's ability to become a high performing unit (Pulakos et al., 2000). The agile teams at both organisations have adapted with the shortest development cycles to achieve higher implementation velocities.

At Akldevelopment and Meldevelopment, the key to achieving high performing agile development teams was *strategic support* (having executive support for agile adaptation). Without such support, the effective cross-functional effort would not been possible. Adapting the product development environment with cross-functional effort required new roles such as product analyst and usability analyst (Akldevelopment), product development manager (Meldevelopment), and agile tester. Without the *strategic support*, these new roles would not have been created. Hence, a supportive environment is critical for adapting a product development environment with new roles, enabling high development performance.

The *strategic support* also enabled the work teams at the two organisations to move out individuals, roles and tasks who were mis-matched with their agile development. At both organisations, individuals who did not fit in or accept agile development approach left the organisation. At Akldevelopment, the quality assurance unit was abandoned but the tasks were integrated with the development function. At Meldevelopment, they out-sourced their testing function and quality assurance engineers became agile testers. Hence, strategic support is critical for adapting agile product development environment by removing mismatches (such as individuals, roles and tasks) to have high performing development teams.

This strategic support makes agile development different from the other approaches. Strategic support is a critical factor for achieving higher development team performance. For this reason, the confidence factor in Fitzgerald's (1998) adaptation framework is modified to incorporate strategic support as a sub-factor (Figure 85).

Figure 85 Sub-factor for confidence factor

6.5.5 Audit trail

At Akldevelopment and Meldevelopment, this covert factor is an agile adaptation factor. In agile development, audit trail (manual task) is adapted to avoid impacting the delivery speed of their short development cycles. The audit trail adaptation factor has a relationship with the following agile principles for software development; *build projects around motivated individuals, give them the environment and support they need and trust them to get the job done.* At Akldevelopment and Meldevelopment, these were also the basis for adopting and adapting agile practice (identified with organisational) to achieve the shortest development cycles.

At both organisations, the software engineers adapted with a quality assurance function, they are responsible for the audit activities in their agile development environment. Engineers also check that all development procedures and standards are implemented properly. The engineers are trusted to ensure the quality of their own work.

At Akldevelopment, the pair programming and at Meldevelopment, the code reviews is the means for doing the audit checks by software engineers. Continuous code review is a critical agile practice, enabling short development cycles. At both organisations, this was further improved by adapting with agile tester roles. While engineers are still responsible for auditing, the agile testers make sure that they became better with this activity. This additional responsibility enforces a quality focus; engineers now have an absolute responsibility for their product quality. Auditing is upfront and integrated with development and is done simultaneously as requirements are implemented.

The automation of audit trails is another key factor for engineer responsibility for this task. Parts of their integrated development environment (IDE) and version control systems (VCS) are coverage checks, code style checks, code documentation check, and issue tracking automation. At both organisations, automation enables their agile work teams not to compromise their delivery speed while taking this addition responsibility.

Continuous code review and automation for carrying out audit trails make agile development different from the other approaches. These practices are critical for achieving short development cycles. For this reason, the audit trail factor in Fitzgerald's (1998) adaptation framework is modified to incorporate continuous code review and automation as sub-factors (Figure 86).

Figure 86 Sub-factors for audit trail

6.5.6 Raise the profile of IS department

This covert factor is one of the most important agile adaptation factors at Akldevelopment and Meldevelopment. Both organisations' agile work teams provided evidence to prove the usefulness of the agile approach for product development. This was done through their high performance in *delivery, productivity and quality* of their features (Sherwood, 1988).

This covert adaptation factor is related to the following *agile principles* for software development; *our highest priority is to satisfy the customer through early and continuous delivery of valuable software; deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale;* and *working software is the primary measure of progress.* At Akldevelopment and

Meldevelopment, these were the other reasons for adopting and adapting agile practices (identified with organisational and overt factors) to become high performing work teams.

They both required behavioural and process adaptations by their development and business function to create a common *strategic vision* for product development (Gordon, Stewart, Sweo, & Luker, 2000). Both organisations had individual *commitment* through the *strategic vision* to make the agile development approach, structures, and work style (team effort) including the planning (micro) strategies function effectively (Woodman, 1989).

Most importantly, both organisations had *political motivation* (having support (executive) for agile adoption) and the change happened through the *change agents* (individuals who drove agile adoption) (Greiner & Schein, 1988). The engineers knew that the team authority was empowered to ensure that the new development approach worked. Hence, the *cost of choice* was clear to the individuals (Woodman, 1989). This was one of key the reasons for the successful behavioural adaptations at both organisations.

At Akldevelopment, this organisational change had a *micro orientation* (used one development team only for agile adoption) (Abernathy & Clark, 1985). Akldevelopment had several other work teams but created an agile work team, using *incremental* (one team at a time) approach to achieve the change for agile development (Nadler & Tushman, 1989). In contrast, Meldevelopment had a *macro orientation* (all the teams adopted agile development), using a *revolutionary* approach (have organisational-wide change for agile development all at once). Meldevelopment had a small-sized engineering function; this was a factor in their limited (manageable) upheaval and impact on their new projects.

The *micro orientation* strategy at Akldevelopment minimised the risk of upheaval as a result of sudden organisational-wide change and its impact on their new projects. The incremental change was through temporary assignment of engineers to the agile team to learn about and implement appropriate changes in their work teams. Akldevelopment also have a practice that requires work team presentations- this approach highlighted the achievement of the agile work team to others, creating an interest in the approach. At Meldevelopment, presentation of a highly successful agile project was also done,

highlighting the achievements to create an interest in the approach. Hence, presentation of agile results is critical for creating an interest in the approach.

At both organisations, the agile work teams raised their profile through high *performance-high commitment* behaviour, achieved through *delegation, cross-functional teamwork, empowerment and integration of people and technology* (Sherwood, 1988). The engineers at both organisations adapted their process and technology including adapting their behaviours to fit their social work system, implementing features and making them available frequently and on a regular basis for their business function. Such development output is critical to highlight the benefits of the agile approach to others.

Publishing results and practices is critical to create an interest in successful agile adoption. For this reason, the profile of IS department in Fitzgerald's (1998) adaptation framework is modified to incorporate publishing results and practices as a sub-factor (Figure 87).

Figure 87 Sub-factors for raise the profile of IS department factor

6.6 Summary

The cross-case analysis shows that both organisations have similar adaptation factors despite having different product markets and domains. At a conceptual level the market-driven organisations have the same issues to deal with to be successful (the organisational, overt and covert factors identify these issues). The cross-case analysis of the two organisations shows that in practice (at a physical level) there are some key

development practices that must be achieved in a market-driven environment, such as; a solid vision (organisational) for product development, a dynamic core competency, proactivity for internal collaboration and sharing of tasks (fluid boundaries between roles and functional units), and ability for shorter development cycles with micro-planning for frequent market releases. Therefore, another key issue that emerged is development agility is achieved mostly through dynamic adaptations; product development teams continuously learn and retain development experience to be able to swiftly adopt or adapt a practice. Next key issue that emerged from the cross-case analysis is that behavioural (work behaviour) adaptation is also critical; individuals must adapt their behaviour to work and share tasks with others. The other key issue that emerged is that agile adaptation decisions are collectively (mutually) made at development level; teams have cross-functional membership, ensuring organisation-wide benefit from the development approach. There is also a key difference between the two organisations. This is in relation to their code development efforts; pair programming in comparison to solo programming practice. This difference was due to one having a large number of senior engineers and also, they were multi-skilled; they adopted solo programming practice. The other was a new team and with most at engineer level having specialist skills; they adopted pair programming to become generalist. Therefore, while an agile approach is driven by team effort, adaptation must also enable strategic advantage through a productive development environment. Finally, the cross-case analysis highlights the following important aspects of agile adaptation: factors which drive agile adaptation, behavioural adaptation to have workforce agility, mutual adaptation to provide strategic development approach driven by cross-functional effort, and dynamic adaptation to facilitate a flexible in-house development competency.

Chapter 7 concludes this study, summarising both its contributions and limitations, and providing suggestions for future research.

Chapter seven: Conclusions and future research

7. 1 Introduction

The overall aim of this study is to analyse factors that influence the adaptation of agile development methods; adaptation enables development agility in a dynamic business environment. Failure to adapt has enormous development implications in a market-driven product environment (lack of appropriate competences, inability for innovations, cost and time overruns, and inferior products). This study used a positivist case study research method to apply an adaptation framework, the Adaptation Framework for Software Development Approaches, developed by Fitzgerald (1998). This is a non-agile framework adopted for this research to determine agile adaptation factors. The relevance of this empirical grounded theoretical framework to the study was realised through discussion with agile practitioners during the preliminarily investigation with possible case study participants. The constructs from this framework served as a basis to collect, analyse and report the research findings; a single-theory multi–level analysis is used to analyse the empirical data. It involved analysing separately the case study background and case study description based on constructs of the framework to identify adaptation factors. This research is explanatory in nature but an exploratory approach is also employed to identify emergent adaptation factors and modify the framework to address the nature of agile development. This study treats each case study as a separate unit for making TE (Theoretical to Empirical) Generalizability; the theoretical constructs (factors from the adaptation framework) are matched with empirical statements for each case study separately to determine their individual adaptation factors (Lee & Baskerville, 2003). Therefore, the study considers each organisation as unique with different influences resulting in distinct adaptation factors. The behavioural adaptations and adaptation concepts of work processes identified through the literature review are adopted for explanation building for the identified adaptation factors. A cross-case analysis aims to provide better understanding and insights based on practical information on these methods and their adaptation factors.

The principal research question of this inquiry is ‘how does adaptation work in an agile approach?’. The empirical data for this study was compiled using two case studies, one from New Zealand and the other from Australia. An orderly one week observation of each case study organisation’s product development lab was made to understand and

learn about their agile environments. This knowledge was used to do an in-depth investigation, using interviews at both organisations.

7.2 Adaptation factors of the agile approach

This research contends that software development success with agile methods is dependent on the appropriate adaptation of those methods. Although a wealth of agile adoption information is available, very little exists on this key aspect of these methods (Aydin et al, 2004; Abrahamsson et al., 2003; Fitzgerald et al., 2003). The analysis of the adaptation results of the two organisations reveal that their individual success is attributed to the adaptation of their product development environment. Their coordinated development environments have organisational functions, external partners and clients collaborating as a single unit with proactive behaviours and fluid boundaries. Both achieved this through organisational, method, role and behavioural adaptations.

The agile adaptation factors identified for each organisation are found through answering the main research question and its related sub-questions. A summary of findings relating to these research questions is now provided.

Answering the first and second sub-questions, ‘how and why do organisational factors influence the adaptation of an agile approach?’, and ‘how and why do political and intellectual factors influence the adaptation of an agile approach?’ provided an analysis of the reasons for adapting the relevant agile practices.

Two case studies were investigated and reported separately in Chapters four and five. Empirical statements were compiled on their background, methodology-in-action, organisational factors, covert factors, and overt factors based on the theoretical framework. Each case study is described accordingly. These descriptions made it possible to understand their individual needs for an agile approach (adaptable and dynamic), strategically driven and supported agile adoption, a hybrid agile method (more flexibility and adaptability), and assigning adaptation responsibility to the actual method users (for swift decision making and performance accountability).

This also lead to an understanding of the relationship between the agile approach and organisational factors (both influence adaptation to become more strategic), and also, product quality is an additional organisational goal (agility provides strategic benefits-both quality and productivity are maximised rather than one being compromised to

achieve the other). Knowledge gained about covert and overt factors in agile method adaptations through case study descriptions is on the individual need for adapting their agile approach (to enhance the strategic development approach, team and environment).

Answering the main research question ‘how does adaptation work in an agile approach?’ identified and explained the adaptation factors of the individual product development environments in Chapter six. Rationalizations for these factors were applied through adaptation lenses on behavioural and work processes. These factors are briefly summarized next.

7.3 Research contributions and implications

This is the first research to identify adaptation factors of agile approaches for software development. The research findings are to be regarded as particularly important for software development methodology researchers and the software development community. The main contributions and theoretical and practical implications for both are explained in the next section.

7.3.1 Contributions

The study has made several contributions:
The main contribution of the thesis is that it has identified agile approach adaptation factors and provides an adapted agile approach adaptation framework (Figure 88).

Another contribution of this thesis is that it has identified behavioural adaptations or changes in the individual workforce behaviour as a critical factor to successfully apply and adapt agile practices in a market-driven product development environment.

The thesis also shows that agile approach adaptations are mutual in a market-driven environment supported by cross-functional product development effort.

The final contribution is that the thesis shows that agile adaptations for market-driven product development environment are mostly dynamic rather static where the method users make adaptation decisions and directly use the adapted practices on a project.

7.3.2 Implications for theory

This thesis extends the agile software development adaptation literature in the ways listed below:

This thesis proposes a taxonomy of agile adaptation factors and provides an adapted agile adaptation framework (Figure 88). These factors identify and provide an insight into individual organisational, overt and covert factors driving agile approach adaptation, creating a strategic (organisational factors and agile approach) product development environment. These individual factors are the main basis for adapting the organisational structures, development structures, and agile approach to swiftly identify and deliver business value innovations. The adapted agile adaptation framework provides a basis for further theoretical application to improve the framework for market-driven product development environments and to determine adaptation factors in other types of agile setups (contract and in-house software development for organisational use only) to make it relevant for those types of agile development.

This thesis suggests that the changes in the individual work behaviour of team members enable successful outcomes of agile approach deployment and adaptations on development projects in a market-driven environment. These outcomes require highly adaptive individuals capable of swift behavioural adaptations to continuously learn and develop new skills, share different development tasks, and work in a team environment.

This thesis puts forward that the agile approach for market-driven product development is mutually adapted by the actual method users rather than adapted solely by a single entity. Identifying and developing high value software features and products is dependant upon effective cross-functional team (collective) effort, resulting in cross-functional development teams being formed. Any adaptation decision to improve the agile development approach is collectively made to avoid limiting the benefits for others.

This thesis suggests that agile method adaptation in a market-driven environment is mostly dynamic. Dynamic adaptation requires method users to have substantial local product development experience to swiftly make collective decisions to improve, construct or adopt a method fragment on-the-fly for immediate application on projects.

7.3.3 Implications for practice

For the software development community the thesis provides documented experiences of agile approach adaptations from two highly successful organisations, providing an understanding of the nature of agile method adaptation in a market-driven environment.

The practical adaptation lessons acquired from the two case study organisations are as follows:

For achieving strategic product development environment, a three-phase agile development strategy with high-level planning, low-level planning, and development and testing phases driven by short development cycles is required for developing business value features and products in a market-driven environment. This three-phase development facilitates swift cross-functional collaborations through task sharing and providing support on functional responsibilities. Agile approach adaptations to create fluid boundaries, structures and roles are crucial to create flexibility, motivating individuals for such collaboration.

Development efforts should be organised into small sub-teams, with informal technical and leadership roles. Individuals must adapt to work or to be re-located in any sub-teams. Development roles must be adapted to have no separation of duties, creating software engineering roles. The formal team leadership must adapt to become proactive with product management to bring field and product experience into the agile work team. These adaptations maximise productivity in agile development teams, providing a strategic product development environment.

Specialist roles should adapt to provide on-the-job training, coaching and support for development teams, fostering a generalist skill base to deliver features in short development cycles. On-site customers (business role) must also adapt to have a software development background to understand the implementation complexities of high-level requirements. Domain support should continuously be adapted to enhance domain knowledge, enabling swift backlog creation and implementation. These adaptations facilitate a strategic product development environment based on a dynamic core capability for steady, high team performance.

The development and testing environment must be highly integrated and supported by automation for coding and testing (TDD, acceptance testing, and code reviews) for having short development cycles. Engineers must have usability and quality assurance support to implement features and unit tests. These adaptations develop a strategic product development environment with the capability for frequent feature releases on a regular basis.

For behavioural adaptations, hiring practices have to be adapted to target recruiting individuals who demonstrate flexibility to change their work behaviours to work collectively in team environment, learning new skills and acquiring new knowledge through task sharing.

For mutual adaptation with agile development, full individual empowerment for collective decision making on projects at development level is required. This empowerment facilitates swift decision making and taking full responsibility for team performance. Individual empowerment to improve development practices, plan and monitor projects, and provide inputs on hiring and learning practices motivates individuals with collaborative and supportive behaviour for overall team effort.

For dynamic adaptation, individuals must adapt to have multiple skills to perform a wide variety of development tasks and to continuously enhance their development experiences and product knowledge. This role adaptation develops the tacit knowledge of local product development, providing individual ability to provide useful information and insights for swiftly making adaptation decisions or solving development problems.

Figure 88 Adapted agile adaptation framework for market-driven development.

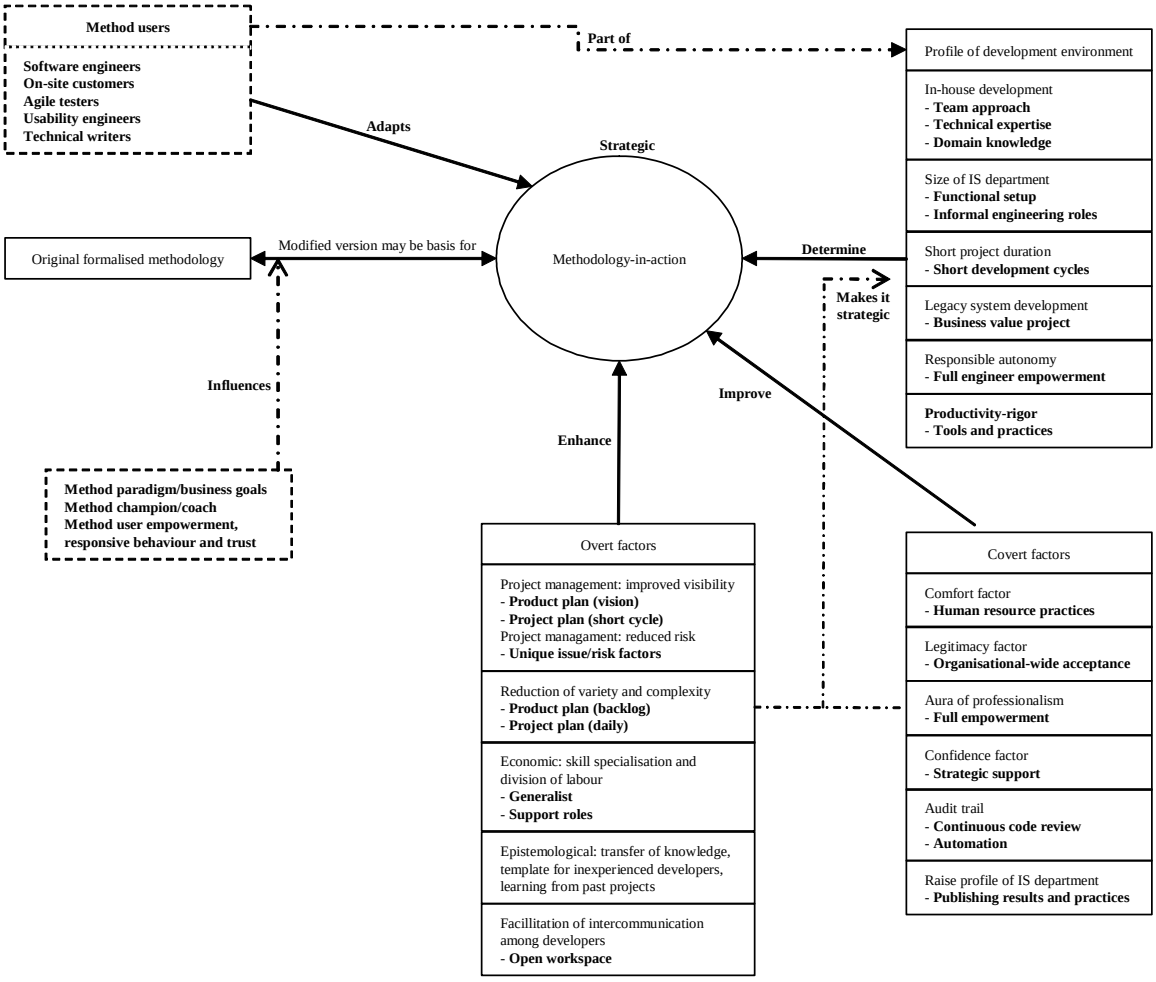


Figure 88 presents an adapted agile adaptation framework for market-driven environments based on the key findings. Items which appear in **bold, broken boxes and broken arrows** are adapted in comparison to Fitzgerald's (1998) framework (Figure 2), giving the adapted agile adaptation framework.

The adapted agile adaptation framework shows that the main structures (development environment factors, overt factors and covert factors) of the framework are consistent when moving to agile development. However, each element of the main structures (individual factors associated with each structure) is adapted. In a market-driven product development environment, these individual factors transform an organisation into an agile organisation. Hence, in this type of development environment the organisational structures (functional units, practices, roles, skills, tasks, and tools) are organic- they keep evolving due to emerging market factors to deliver differential products. In the following section the adapted agile adaptation framework (Figure 88) and how it differs from the Fitzgerald's (1998) adaptation framework (Figure 2) are explained.

7.3.4 Adapted agile adaptation framework

The main components of the adapted agile adaptation framework (Figure 88) are method users, original formalised methodology and methodology-in-action, profile of development environment, covert factors and overt factors. These are largely consistent with Fitzgerald's adaptation framework with the exception that "method users" component has replaced the "developer-embodied factors" component. Fitzgerald's framework also has other components such as "development context: problem situation/business opportunity", "development/methodology user" and "information processing systems". The component "development context: problem situation/business opportunity" is not included in adapted agile framework since this framework is for market-driven development, solving problem (development) situations and taking business opportunities. The components "development/methodology user" and "information processing systems" are not included since the adapted framework assumes that the adapted method will be used by the method users to develop software as shown in Fitzgerald's framework.

7.3.4.1 Method users

The "method users" component in the adapted agile adaptation framework (Figure 88) identifies the individuals who are co-located and are members of the agile development teams. These are the individuals (including any supporting roles) who use agile method on a daily basis and are empowered to make adaptation decisions. These roles include on-site customers, software engineers, quality assurance engineers, usability professionals and technical writers. Hence, the developer-embodied factors component identified in the original framework is replaced by the "method users" component. This component highlights that the adapted agile adaptation framework emphasises the cross-functional membership of agile development teams as critical as they are collectively responsible for method adaptation. This is shown in the adapted agile adaptation framework by linking ("adapts") the "method users" component with the "methodology-in-action" component. The adapted agile adaptation framework also highlights that this cross-functional membership is determined by the profile of development environment factors. This is shown in the adapted agile adaptation framework by linking ("part of") the "method users" component with the "profile of development environment" component.

7.3.4.2 Original formalised methodology and methodology-in-action components

The adapted agile adaptation framework (Figure 88) has the "original formalised methodology" and methodology-in-action components. This adapted framework shows that these two components differ from the two identified in Fitzgerald's (1998) adaptation framework (Figure 2) through their underlying factors. The adapted agile adaptation framework highlights critical factors that influence these two components to have successful product development in a market-driven environment: paradigm/business goals, method champions/coach, and method user empowerment, responsive behaviour and trust. First, the adapted agile framework stresses that for adoption (original formalised methodology) the agile paradigm must match business goals for software development and requires method champions or coaches for successful deployment. The agile paradigm drives product development based on change and close interactions with stakeholders as required in a market-driven environment. Agile adoption requires support to enable swift changes to development practices and work behaviours. Second, the adapted agile framework emphasises method user empowerment, responsive behaviour and trust as critical factors for making successful adaptation decisions to have a strategic methodology-in-action. The method user factors enable issues to be solved immediately, avoiding bureaucracy and enabling on-the-fly decision making, since the method users develop substantial development knowledge and experience.

7.3.4.3 Profile of development environment

The adapted agile adaptation framework (Figure 88) has adapted the "profile of development environment" component. This adapted agile adaptation framework differs from Fitzgerald's (1998) adaptation framework, since each of the adaptation factors (organisational factors) under the "profile of development environment" component have their own sub-factors. These sub-factors drive agile approach adaptation making the organisational factors and the development approach strategic in the market-driven environment. In the adapted agile adaptation framework the "profile of development environment" component is linked ("determine") to methodology-in-action component ("strategic"), showing that organisational adaptation factors enhance the methodology-in-action to make it a strategic development approach. In the adapted agile adaptation framework, the "overt and covert" components (overt and covert roles of agile

approach) are also linked ("makes it strategic") to the "profile of the development environment" component, showing that overt and covert adaptation factors enhance the organisational factors to make them strategic in a market-driven environment. Next, the sub-factors for each of the organisational factors of the "profile of development environment" component in the adapted agile adaptation framework (Figure 88) are identified and explained.

The in-house development factor has sub-adaptation factors; team approach, technical and domain knowledge. The adapted agile adaptation framework emphasises these as critical factors for adapting in-house development capability driven by an agile approach, making in-house development a strategic organisational factor. These sub-factors drive adaptation of agile product development environments to achieve a dynamic core competency for continuously developing differential features and products for a market-driven environment.

The size of the IS Department factor has sub-adaptation factors; functional setup and informal engineering roles. The adapted agile adaptation framework emphasises these as critical sub-factors for adapting the product development environment to provide a dynamic engineering capacity on development projects, having strategic feature releases through early and continuous delivery of software.

The short project duration factor has short development cycles as a sub-factor driving adaptations. The adapted agile adaptation framework highlights short development cycles as a critical sub-factor for adapting agile development projects, enabling strategic cycle times for development and releases.

In the adapted agile adaptation framework, the legacy system development factor has business value as a sub-factor driving agile approach adaptations. The adapted framework stresses business value as a critical sub-factor for driving agile approach adaptation, providing a strategic product (high-level) planning capability to determine the business value of proposed improvements of legacy features and products.

The responsible autonomy factor has full engineer empowerment as a sub-factor driving agile approach adaptation. The adapted agile adaptation framework emphasises full engineer empowerment on development projects as critical to achieve high team

performance on a regular basis and to have collective accountability for feature delivery.

In the adapted agile adaptation framework (Figure 88), the productivity-rigour factor has tools and practices as a sub-factor driving agile approach adaptations. The adapted agile adaptation framework emphasises this sub-factor as critical to maximise development productivity and to deliver high quality products to keep ahead of competitors in a market-driven environment. In Fitzgerald's (1998) framework (Figure 2), this factor is highlighted as "productivity-rigour trade-off". The "trade-off" is removed since productivity and rigour (for quality) are two critical factors to be successful with software products in a market-driven environment.

7.3.4.4 Overt factors

The adapted agile adaptation framework (Figure 88) has adapted the "overt" component. This differs from Fitzgerald's (1998) framework, since each of the adaptation factors under the "overt" component (except the epistemology role factor) have their own sub-factors. These sub-factors drive agile approach adaptation, making the development approach strategic in a market-driven environment. In the adapted agile adaptation framework, the "overt" component is linked ("enhance") to the "methodology-in-action" component (strategic), showing that overt adaptation factors enhance the methodology-in-action to make it a strategic development approach. Next, sub-factors for each of the adaptation factors of the "overt" component in the adapted agile adaptation framework are identified and explained.

The project management factor has product planning (vision), project planning (short cycles) and unique issue/risk as sub-factors driving agile approach adaptation. The adapted agile adaptation framework emphasises these sub-factors as critical to enable effective organisational learning and input on proposed innovations, improving project visibility and effective risk management for timely development and releases.

The reduction of variety and complexity factor has product planning (backlog) and project planning (daily) as sub-factors driving agile approach adaptation. The adapted agile adaptation framework stresses these sub-factors as critical to minimise product and project complexities, eliminating wastage in the development effort and providing implementation flexibility and speed.

The economic division of labour factor has generalist and support roles as sub-factors driving agile approach adaptation. The adapted agile adaptation framework emphasises these sub-factors factors as critical to have affordable product development based on a generalist skill set.

The epistemological role of methodology factor enables adaptation of agile approach practices in relation to transfer of knowledge, template for inexperienced developers and learning from past project practices for learning and knowledge acquisition. The adapted agile adaptation framework emphasises these sub-factors factors, enabling quick learning and upskilling to function in a team and creatively solving problems in a market-driven environment. These sub-factors are also in Fitzgerald's (1998) adaptation framework (Figure 2).

The facilitation of intercommunication among developers factor has open workspace as a sub-factor driving agile approach adaptation. The adapted agile adaptation framework stresses this sub-factor as critical to enhance interaction, achieving cooperation and delivery speed.

7.3.4.5 Covert factors

The adapted agile adaptation framework (Figure 88) has adapted the "covert" component. This differs from Fitzgerald's (1998) framework, since each of the adaptation factors under the "covert" component have their own sub-factors. In the adapted agile adaptation framework, the "covert" component is linked ("improve") to the "methodology-in-action" component ("strategic"), showing that covert adaptation factors enhance the methodology-in-action, making it a strategic development approach. Next, sub-factors for each of the adaptation factors of the "covert" component in the adapted agile adaptation framework are identified and explained.

The comfort factor has human resource practices as a sub-factor driving agile approach adaptation. The adapted agile adaptation framework stresses this sub-factor as critical to have an attractive work environment to achieve full development capacity on a regular basis.

The legitimacy factor has organisational-wide acceptance as a sub-factor driving agile approach adaptation. The adapted framework emphasises this sub-factor as critical for

integrating the separate functional efforts into a collective unit for product development while serving the needs of all the stakeholders.

The aura of professionalism factor has full empowerment as a sub-factor. The adapted framework stresses this sub-factor as critical for motivating engineers for high performance and taking full responsibility for their team performance.

The confidence factor has strategic support as a sub-factor driving agile approach adaptation. The adapted framework emphasises this sub-factor as critical for implementing changes to achieve high team performance in delivery of high quality features and products in a market-driven environment.

The audit trail factor has continuous code review and automation as sub-factors driving agile approach adaptation. The adapted framework emphasises these sub-factors as critical to achieve strategic market releases through automation, minimising manual and post hoc quality assurance tasks.

The raise the profile of IS department factor has publishing practices and results as a sub-factor driving agile approach adaptations. The adapted framework emphasises this sub-factor as critical to achieve organisation-wide support for agile development.

The components of the adapted agile adaptation framework (original formalised methodology and methodology-in-action, profile of development environment, covert factors and overt factor) show that they are comprehensively adapted for agile development. The framework shows the critical factors associated with the “original formalised methodology and methodology-in-action” components for having an appropriate method-in action for development projects. The adapted agile framework also shows the critical underlying elements or sub-factors of the organisational factors (profile of development environment), overt factors and covert factors, which enable a strategic product development environment by driving its evolution based on emerging market factors and improvements in product development technologies.

7.4 Limitations

There are some limitations to this research. First, it had only two case studies, both were large software vendors- this meant that both cases were homogenous in nature. Second,

a traditional approach was used for data collection based on asking questions about the past. These research design limitations are explained below.

This study would have been enhanced with more varied case study organisations such as contract software development teams, in-house software development teams for business organisations and small scale-software vendors. This would have enabled a better understanding on agile approaches across different contexts. This would have helped to determine if they have different agile adaptation factors, providing insights into different development environments, method practices and fragments.

With this limitation, the research design was enhanced by including an in-depth study approach. The major obstacle to the sample size was not a lack of interest from the software development community but their suitability as mature, large scale agile adapters.

The research design could also have been improved by using a long term, direct observation aspect of the ethnography approach. This type of observation of daily participation in production labs would have enhanced data collection and enabled deeper analysis of adaptation factors, in particular of behavioural adaptations. Although this type of participation was discussed with the participants, it was agreed that it might cause disruption to their development targets. Long term participation would have helped to identify when the adaptation happened and under what circumstances. Participating organisations have no formal documentation for their agile process and adaptations- these are driven by tacit knowledge.

The research design was enhanced by short term direct observations of production labs, using a variety of informants, using citations from interviews as evidence, and validating the case study reports.

7.5 Future research

This study has provided the basis for future research to further improve our understanding of agile adaptation.

Further research could consider validating the agile adaptation factors identified in this study through a larger case study sample with more varied case study organisations. This might include small sized in-house development teams, software vendors and

contracting companies. It would provide an improved understanding of agile adaptation and a further refinement of the agile adaptation factors identified in this research.

Future research might also include a large scale post-positive study research involving world-wide agile practitioners to validate and refine the adaptation factors discussed in this thesis. Future research could also take an alternative approach using grounded theory to build case studies and to create a taxonomy of agile adaptation factors.

Finally, future research might consider investigating adaptation factors at different levels in agile software development organisations such as the management, project, team level and individual levels. This would be a similar approach to Fitzgerald, Russo, & O'Kane's (2003) study and would investigate method adaptations at different levels. Such research would enhance adaptation knowledge.

References

Abernathy, J., & Clark, B. (1985). Innovation: mapping the winds of creative destruction. *Research Policy, 14*, 3-22.

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile software development methods: review and analysis*. Retrieved May, 10th, 2005, from http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf.

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New directions on agile methods: a comparative analysis. In *proceedings of the 25th international conference on software engineering* (pp. 244-254). Portland, Oregon. IEEE Computer Society.

Ackerman, R. (1973). How companies respond to social demand. *Harvard Business Review, 51*(4), 88-98.

Addison, T., & Vallabh, S. (2002). Controlling software project risks: an empirical study of methods used by experienced project managers. In *proceedings of the 2002 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology* (pp. 128-140). Port Elizabeth, South Africa: South African Institute for Computer Scientists and Information Technologists.

Ahituv, N., Hadass, M., & Neumann, S. (1984). A flexible approach to information systems development. *MIS Quarterly, June*, 69-78.

Aitken, J., Christopher, M., & Towill, D. (2002). Understanding, Implementing and Exploiting Agility and Leanness. *International Journal of Logistics: Research & Applications, 5*(1), 59-74.

Albright, R. E., & Kappel, T. A. (2003). Roadmapping the corporation. *Research Technology Management, 46*(2), 31.

Allison, T. (1971). *Essence of decision*. Boston: Little, Brown.

Allworth, E., & Hesketh, B. (1999). Construct-oriented biodata: Capturing change-related and contextually relevant future performance. *International Journal of Selection and Assessment, 7*(2), 97-111.

Alwardt, A. L., Mikeska, N., Pandorf, R. J., & Tarpley, P. R. (2009). *A lean approach to designing for software testability*. Paper presented at the AUTOTESTCON, 2009. IEEE Computer Society (pp. 178 - 183).

Ambler, S. (2005). Great leaders are made. *Software Development, 13*(11), 55-57.

Ambler, S. (2006). *Agile works in practices*. Retrieved August, 2006, from http://www.ambysoft.com/surveys/agileMarch2006.html.

Ambler, S. (2007). *Agile adoption rate survey results: March 2007*, from http://www.ambysoft.com/surveys/agileMarch2007.html.

Ambler, S. (2009). Q&A: What's next for agile?, *InformationWeek* (pp. 44-44).

Ancona, D. G., & Caldwell, D. F. (1992). Bridging the boundary: External activity and performance in organisational teams. *Administrative Science Quarterly, 37*(4), 634-665.

Argyris, C. (1966). Interpersonal barriers to decision making. *Harvard Business Review, 44*(2), 84.

Arogyaswamy, B., Byles, C., & (1987). Organisational culture: Internal and external fits. *Journal of Management, Winter*. (13), 647 - 658.

Asadi, M., & Ramsin, R. (2009). Method engineering process patterns. http://doi.acm.org/10.1145/1506216.1506249 . In *proceedings of the 2nd India software engineering conference* (pp. 143-144). Pune, India ACM.

Ashford, S. J., Rothbard, N. P., Piderit, S. K., & Dutton, J. E. (1998). Out on a limb: the role of context and impression management in selling gender-equity issues. *Administrative Science Quarterly, 43*(1), 23-57.

Augustin, L., Bressler, D., & Smith, G. (2002). Accelerating software development through collaboration. http://doi.acm.org/10.1145/581339.581409 . In *proceedings of the 24th international conference on software engineering* (pp. 559-563). Orlando, Florida ACM.

Avison, D. E., & Myers, D. M. (1995). Information systems and anthropology: an anthropological perspective on IT and organisational culture. *Information Technology & People, 8*(3), 43-56.

Avison, D., & Fitzgerald, G. (1995). *Information systems development: methodologies, techniques and tools*. London: McGraw-Hill Publishing.

Avison, D., & Fitzgerald, G. (2003). *Information systems development: methodologies, techniques and tools* (3rd Ed.). London: McGraw-Hill Publishing.

Avison, D., & Wood-Harper, T. (1991). Information systems development research. *The Computer Journal, 34*(2), 98-112.

Aydin, M., Harmsen, F., Slooten, V., & Stegwee, A. (2005). On adaptation of an agile information systems development method. *Journal of Database Management, 16*(4), 24-40.

Ayed, B., Ralyte, J., & Rolland, C. (2004). Constructing the Lyee method with a method engineering approach. *Knowledge-Based Systems, 17*, 239-248.

Babakus, E., Cravens, D. W., Johnston, M., & Moncrief, W. C. (1999). The role of emotional exhaustion in sales force attitude and behaviour relationships. *Journal of the Academy of Marketing Science, 27*(1), 58.

Babbie, E. R. (2007). The practice of social research. Belmont, CA: Thomson/Wadsworth.

Bagnall, A. J., Rayward-Smith, V. J., & Whittley, I. M. (2001). The next release problem. *Information and Software Technology, 43*(14), 883-890.

Baird, S. (2003). *Teach yourself extreme programming in 24 hours*. Indianapolis, Indiana: Sams Publishing.

Bajec, M., & Vavpotic, D. (2008). A framework and tool-support for reengineering software development methods. *Informatica, 19*(3), 321-344.

Bajec, M., Vavpotic, D., & Krisper, M. (2007). Practice-driven approach for creating project-specific software development methods.

http://dx.doi.org/10.1016/j.infsof.2006.05.007 . *Information and Software Technology, 49*(4), 345-365.

Bakker, B., Demerouti, E., Taris, W., Schaufeli, B., & Schreurs, G. (2003). A multi-group analysis of job demands-resources model in four home-care organisations. *International Journal of Stress Management, 10*, 16-38.

Baldwin, A., & Chung, J. (1995). A formal approach to managing design processes. *Computer, 8*, 43-53.

Barley, R. (1983). Semiotics and the study of occupational and organisational values. *Administrative Science Quarterly*, 28, 393-413.

Barnett, L. (2007). Widespread adoption emphasis on productivity and quality. *Agile Journal, 2(7)*, 2-4.

Barritt, D. (2002). IEC 61131 and DSDM in real-time process control applications. *Computing & Control Engineering Journal, 13*(2), 94-100.

Baskerville, R. (1996). Structural artifacts in method engineering: the security impreative. In S. Brinkkemper, K. Lyytinen & R. Welke (Eds.), *Method engineering : principles of method construction and tool support* (pp. 8-28). London: Chapman & Hall.

Baskerville, R., & Pries-Heje, J. (2001). A multiple-theory analysis of a diffusion of information technology case. *Information Systems Journal, 11*(3), 181-212.

Baskerville, R., & Stage, J. (2001). Accommodating emergent work practices: ethnographic choice of method fragments. In N. Russo, B. Fitzgerald & J. DeGross (Eds.), *Realigning research and practice in information systems development* (pp. 11-27). Boston: Kluwer Academic.

Baskerville, R., Travis, J., & Truex, D. (1992). Systems without method: the impact of new technologies on information systems development projects. In K. E. Kendall, K. Lyytineen & J. I. De Gross (Eds.), *the Impact of computer supported technologies on information systems development* (pp. 241-264). Amsterdam: North-Holland, Netherlands.

Beck, K. (1999a). *Embracing change with extreme programming*. Computer, 32(10), 70-77.

Beck, K. (1999b). *Extreme programming explained: Embrace change*: Addison-Wesley.

Beck, K. (2001). Aim, fire. *Software, 18*(5), 87-89.

Beck, K., & Andres, C. (2004). Extreme programming: Embrace change (2nd ed.). Boston: Addison-Wesley.

Beck, K., & Boeham, B. (2003). Agility through discipline: a debate. *IEEE Computer (June)*, 44-66.

Becker, J. D., Insley, R. G., & Endres, M. L. (1997). Communication skills of technical professionals: a report for schools of business administration. http://doi.acm.org/10.1145/568495.568496 . *SIGCPR Computer Personnel, 18* (2), 3-19.

Begel, A., & Nagappan, N. (2007). *Usage and perceptions of agile software development in an industrial context: an exploratory study.* Paper presented at the first international symposium on empirical software engineering and measurement. IEEE Computer Society (pp. 255-264).

Benbasat, I., Goldstein, D. K., & Mead, M. (1987). The case research strategy in studies of information systems. *MIS Quarterly*, 11(3), 369-386.

Benbasat, I., Goldstein, K. D., & Mead, M. (1987). The case research strategy in studies of information systems. *MIS Quarterly, September*, 369-386.

Benyon, D., & Skidmore, S. (1987). Towards a tool kit for systems analyst. The Computer Journal, 30(1), 2-7.

Berger, H. (2007). Agile development in a bureaucratic arena - a case study experience. *International Journal of Information Management, 27*(6), 386-396.

Berger, H. (2007). Agile development in a bureaucratic arena--A case study experience. *International Journal of Information Management, 27*(6), 386-396.

Bhat, T., & Nagappan, N. (2006). Evaluating the efficacy of test-driven development: industrial case studies. http://doi.acm.org/10.1145/1159733.1159787 . In *proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering* (pp. 356-363). Rio de Janeiro, Brazil ACM.

Boehm, B. (1981). *Software engineering economics*. Englewood Cliffs, NJ: Prentice-Hall.

Boehm, B. (2002). Get ready for agile methods, with care. *Computer, 35*(1), 64-69.

Boehm, B. (2008). Making a difference in the software century. *Computer, 41*(3), 32-38.

Boehm, B., & Turner, R. (2003). Using risk to balance agile and plan-driven methods. *Computer, 36*(6), 57-66.

Boehm, B., & Turner, R. (2004). *Balancing agility and discipline*. Boston: Addison-Wesley.

Bottani, E. (2009). On the assessment of enterprise agility: issues from two case studies. International Journal of Logistics: *Research & Applications, 12*(3), 213-230.

Bouldin, B. M. (1990). Software manager-the nature of change agents. *Software, IEEE, 7*(1), 125-126.

Bowyer, J., & Hughes, J. (2006). Assessing undergraduate experience of continuous integration and test-driven development http://doi.acm.org/10.1145/1134285.1134393. In *proceedings of the 28th international conference on software engineering* (pp. 691-694). Shanghai, China ACM.

Braun, C., Wortmann, F., Hafner, M., & Winter, R. (2005). Method construction - a core approach to organisational engineering. http://doi.acm.org/10.1145/1066677.1066971 . In *proceedings of the 2005 ACM symposium on applied computing* (pp. 1295-1299). Santa Fe, New Mexico ACM.

Breu, K., Hemingway, C. J., Strathern, M., & Bridger, D. (2001). Workforce agility: the new employee strategy for the knowledge economy. *Journal of Information Technology, 17*(1), 21-31.

Brinkkemper, S. (1996). Method engineering: engineering of IS development methods and tools. *Information and Software Technology, 38*(4), 275-280.

Brinkkemper, S., Saeki, M., & Harmsen, F. (1998). *Assembly techniques for method engineering*. (Vol. 1413): Springer.

Britton, C., & Doake, J. (1996). *Software system development: a gentle introduction* (second Ed.). London: McGraw-Hill Companies.

Brock, S., Hendricks, D., Linnell, S., & Smith, D. (2003). A balanced approach to IT project management. In *proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on enablement through technology* (pp. 2-10).ACM

Brooks Jr, F. (1987). No silver bullet: Essence and accidents of software engineering. *IEEE Software, 20*(4), 98-112.

Brooks, G. (2008). *Team pace keeping build times down*. Paper presented at the agile, 2008 conference . IEEE (pp294-297).

Brooks, P. (1975). *The mythical man month*: Addison-Wesley.

Brownsword, L., Oberndorf, T., & Sledge, C. (2000). Developing new processes for CPTS-based systems. *IEEE Software(July/August)*, 48-55.

Burleson, B. R. (2009). Understanding the outcomes of supportive communication: a dual-process approach. *Journal of Social & Personal Relationships, 26*(1), 21-38.

Burns, T., & Stalker, M. (1994). *The management of innovation*. Oxford: Oxford University Press,Tavistock.

Bygstad, B. (2004). IS development as mutual adaptation of technology and business process. In O. Vasilecus, A. Caplinskas, W. Wojtkowski, G. Wojtkowski, J. Zupanèiè & S. Wrycza (Eds.), *information systems development: advances in theory, practice, and education* (pp. 89-101). New York: Springer.

Bynjolfson, E., van Alstyne, M., Bernstein, A., & Renshaw, A. (1997). *Tools for teaching change management*. Paper presented at the IAIM'97, Atlanta.

Byrne, Z. S., & Hochwarter, W. A. (2008). Perceived organisational support and performance. *Journal of Managerial Psychology*, 23(1), 54-72.

CambrideDictionariesOnline. from http://dictionary.cambridge.org/ .

Campion, M. A., Medsker, G. J., & Higgs, A. C. (1993). Relations between work group characteristics and effectiveness: implications for designing effective work groups. *Personnel Psychology, 46*(4), 823-850.

Canfora, G., Cimitile, A., Garcia, F., Piattini, M., & Visaggio, C. A. (2006). Evaluating advantages of test driven development: a controlled experiment with professionals. http://doi.acm.org/10.1145/1159733.1159788 . In *proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering* (pp. 364-371). Rio de Janeiro, Brazil ACM.

Cao, L., & Ramesh, B. (2007). Agile software development: Ad hoc practices or sound principles? *IT Professional, 9*(2), 41-47.

Cao, L., & Ramesh, B. (2008). Agile requirements engineering practices: An empirical study. Software, *IEEE, 25*(1), 60-67.

Carnall, C. (2007). *Managing change in organisation* (5th Ed). Harlow: Prentice Hall.

Charette, R. N. Why software fails? *Spectrum, IEEE, 42*(9), 42-49.

Checkland, P. B. (1999). *Systems thinking*. Oxford: OUP.

Chen, G. (2005). Newcomer adaptation in teams: multilevel antecedents and outcomes. *Academy of Management Journal, 48*(1), 101 - 116.

Chen, Z., & Marx, D. (2005). Experiences with eclipse IDE in programming courses *Journal of Computing Sciences in Colleges, 21* (2), 104-112.

Cheng, B. H. C., & Atlee, J. M. (2007). *Research directions in requirements engineering*. Paper presented at the future of software engineering, 2007 conference. IEEE Computer Society (pp. 285-303).

Chiang, C.-C., & Urban, J. E. (1996). Incremental elicitation and formalization of user requirements through rapid prototyping via software transformations. In *proceedings of the 20th conference on computer software and applications* (pp. 240): IEEE Computer Society.

Chiang, R. I., & Mookerjee, V. S. (2004). Improving software team productivity. *Communications of the ACM, 47*(5), 89 - 93.

Chin, G. (2004). *Agile Project Management: how to succeed in the face of changing project management*. New York: American Management Association.

Cho, L. (2009). *Adopting an agile culture.* Paper presented at the agile conference, 2009. IEEE Computer Society (pp. 400 - 403).

Chong, J., & Hurlbutt, T. (2007). *The social dynamics of pair programming.* Paper presented at the international conference on software engineering. IEEE Computer Society (pp.354 - 363).

Christopher, M., & Towill, D. (2000). Supply chain migration from lean and functional to agile and customised. *Supply Chain Management: An International Journal, 5*(4), 206 - 213.

Chung, W. Y., & Guinan, P. J. (1994). Effects of participative management on the performance of software development teams. *In proceedings of the 1994 computer personnel research conference on reinventing IS: managing information technology in changing organisations* (pp. 252-260). Alexandria, Virginia, United States ACM.

Clark, D., Jr., Zmud, W., & McCray, E. (1995). The outsourcing of information services: transforming the nature of business in the information industry. *Journal of Information Technology, 10,* 221-237.

Cockburn, A. (1998). *Surviving object-oriented projects: a manager's guide*: Addison Wesley Longman.

Cockburn, A. (2000). Selecting a project's methodology. *IEEE Software(July/August)*, 64-71.

Cockburn, A. (2001). *Agile software development*. Boston: Pearson Education.

Cockburn, A. (2002). *Agile software development*. Boston: Addison-Wesley.

Cockburn, A. (2005). *Are iterations hazardous to your project?* Retrieved 10th Decemeber, 2010, from http://alistair.cockburn.us/Are+iterations+hazardous+to+your+project%3f.

Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. *Computer, 34*(11), 131-133.

Cockburn, A., & Highsmith, J. (2001). Agile software development: The people factor. *IEEE Computer, 34*(11), 131-133.

Cohen, B., & Thias, M. (2009). *The failure of the off-shore experiment: a case for co-located agile teams.* Paper presented at the agile conference, 2009. IEEE Computer Society (pp. 251 - 256).

Cohen, S. (1993). New approaches to teams and teamwork. In J. Galbraith & E. Lawler (Eds.), o*rganising for the future: the new logic for managing complex organisations* (pp. 194-226). San Francisco: Jossey-Bass.

Cohen, S. G., & Bailey, D. E. (1997). What makes teams work: group effectiveness research from the shop floor to the executive suite? *Journal of Management, 23*(3), 239.

Cohen, S. G., & Ledford Jr., G. E. (1994). The effectiveness of self-managing teams: a quasi-experiment. *Human Relations, 47*(1), 13-44.

Cohen, W. M., & Levinthal, D. A. (1990). Absorptive capacity: a new perspective on learning and innovation. *Administrative Science Quarterly, 35*(1), 128-152.

Cohn, M. (2004). *User stories applied*. Boston: Addison-Wesley.

Colares, F., Souza, J., Carmo, R., Padua, C., & Mateus, G. R. (2009). *A new approach to the software release planning*. Paper presented at the software engineering conference, 2009. IEEE Computer Society (pp. 207-215).

Cole, A. (1995). Runaway projects: causes and effects. *Software World (UK), 26*(3), 3-5.

Cook, C., & Churcher, N. (2006). Constructing real-time collaborative software engineering tools using CAISE, an architecture for supporting tool development. In p*roceedings of the 29th Australasian computer science conference* - volume 48 (pp. 267-276). Hobart, Australian Computer Society, Inc.

Cook, T. D., & Campbell, D. T. (1979). *Quasi-experimentation: design & analysis issues for field settings*. Boston: Houghton Mifflin Company.

Cooke, W. N. (1994). Employee participation programs, group-based incentives, and company performance: *Industrial & Labour Relations Review*, 47(4), 594.

Cooper, R. G. (1980). How to identify potential new product winners. *Research Management, 23*(5), 10-19.

Coplien, J., & Harrison, N. (2005). *Organisation patterns of agile software development*. Upper Saddle River, NJ: Pearson Reentice Hall.

Coram, M., & Bohner, S. (2005). *The impact of agile methods on software project management*. Paper presented at 12th IEEE international conference and workshops on the engineering of computer-based systems. IEEE Computer Society (363 - 370).

Cordes, C. L., & Dougherty, T. W. (1993). A review and an integration of research on job burnout. *Academy of Management Review*, 18(4), 621-656.

Crant, J. (2000). Proactive behaviour in organisations. *Journal of Management, 26*(431-462).

Creswell, J. W. (1994). *Research design: qualitative and quantitative approaches*. London: Sage Publications.

Creswell, J. W. (1998). *Qualitative inquiry and research design: choosing among five traditions*. CA: Sage.

Creswell, J. W. (2007). Qualitative inquiry and research design: choosing among five approaches (2nd Ed.). Thousand Oaks: Sage Publications.

Crocitto, M., & Youssef, M. (2003). The human side of organisational agility. *Industrial Management & Data Systems, 103*(6), 388.

Crockett, H. D., Hall, G. R., & Jeffries, C. J. (1993). Are undergraduate IS programs serving their markets. *Interface, 15*(2), 9-13.

Cummings, J. N. (2004). Work groups, structural diversity, and knowledge sharing in a global organisation. *Management Science, 50*(3), 352-364.

Cunningham, W. (2001). *Manifesto for agile software development*. Retrieved 20th, December, 2004, from http://agilemanifesto.org/.

Curtis, B., Krasner, N., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communication of ACM, 31*(11), 1268-1287.

Cusack, S. (1988). Moving up to management. *Computerworld, 22*(21), 82.

Dahlbom, B., & Mathiassen, L. (1992). Systems development philosophy. *Computer and Society, 22*(1-4), 12-23.

David, J., McCarthy, W., & Sommer, B. (2003). Agility-the key to survival of the fittest in the software market. *Communication of the ACM, 46*(5), 65-69.

Dawis, V., & Lofquist, L. (1984). *A psychological theory of work adjustment: an individual-differences model and its applications*. Minneapolis: University of Minnesota Press.

Day, D. V., Gronn, P., & Salas, E. (2004). Leadership capacity in teams. The *Leadership Quarterly, 15*(6), 857-880.

DeClue, T. (2003). Pair programming and pair trading: effects on learning and motivation in a CS2 course. *The Journal of Computing Sciences in Colleges, 18*(5), 49-56.

Demerouti, E., Verbeke, W. J. M. I., & Bakker, A. B. (2005). Exploring the relationship between a multidimensional and multifaceted burnout concept and self-rated performance. *Journal of Management, 31*(2), 186-209.

Dennis, A., Wixom, B., & Tegarden, D. (2005). *Systems analysis and design with UML Version 2*. Danvers, USA: John Wiley & Sons, Inc.

Denzin, N. K. (1978). *The research act: a theoretical introduction to sociological methods*. New York: McGraw-Hill.

Dey, I. (2004). Grounded theory. In C. Seale, G. Gobo, F. J. Gubrium & D. Silverman (Eds.), Qualitative Research Practice (pp. 80-93). London: SAGE Publications.

Dhillon, G. (2000). *Interpreting key issues in IS/IT benefits management*. Paper presented at the 33rd annual Hawaii international conference on system sciences, 2000. IEEE Computer Society (pp 1-9).

Dinakar, K. (2009). Agile development: overcoming a short-term focus in implementing best practices. http://doi.acm.org/10.1145/1639950.1639952 . In *proceeding of the 24th ACM SIGPLAN conference companion on object oriented programming systems languages and applications* (pp. 579-588). Orlando, Florida, USA ACM.

Ding, H. B., & Ravichandran, T. (2000). *Pre-emptive radical innovation: building inter-departmental common knowledge in a short product development cycle*. Paper presented at the engineering management society conference, 2000. IEEE Computer Society (pp 575 -580).

Do, H., Rothermel, G., & Kinneer, A. (2004). *Empirical studies of test case prioritisation in a JUnit testing environment*. Paper presented at the software reliability engineering 2004 conference. IEEE Computer Society (pp. 113-124).

Donmoyer, R. (2000). Generalizability and the single-case study. In M. Hammersley & R. Gomm (Eds.), *Case Study Method*. London: Sage Publications.

Doshi, C., & Doshi, D. (2009). *A Peek into an agile infected culture*. Paper presented at the agile conference, 2009. IEEE Computer Society (pp. 84 - 89).

Dougherty, D. (1992). Interpretive barriers to successful product innovation in large firms. *Organisation Science, 3*(2), 179-202.

Downs, J., Hosking, J., & Plimmer, B. (2010). *Status communication in agile software teams: a case study*. Paper presented at the fifth international conference on software engineering advances. IEEE Computer Society (pp. 82 - 87).

Drath, W. (2001). *The deep blue sea: rethinking the source of leadership*. San Francisco: Jossey-Bass.

Druskat, V. U., & Wheeler, J. V. (2004). How to lead a self-managing team. MIT *Sloan Management Review, 45*(4), 65-71.

Dube, L., & Pare, G. (2003). Rigor in information systems positivist case research: current practices, trends, and recommendations. *MIS Quarterly, 27*(4), 597-635.

Duguay, R., Landry, S., & Pasin, F. (1997). From mass production to flexible/agile production. *International Journal of Operations & Production Management, 17*(12), 1183-1195.

Duysters, G., & Hagedoorn, J. (2000). Core competences and company performance in the world-wide computer industry. *Journal of High Technology Management Research, 11*(1), 75.

Dyer, J. L. (1984). Team research and team training: a state-of-the art review. *Human Factors Review*, 286-309.

Dyer, L., & Shafer, R. (2003). Dynamic organisations: achieving marketplace and organisational agility with people. In S. Peterson & A. Mannix (Eds.), *leading and managing people in the dynamic organisation*. Mahwah, NJ: Laurence Erlbaum Associates.

Earl, J. (1996). The risks of outsourcing IT. *Sloan Management Review, 51*, 11- 27.

Edmondson, A. (1999). Psychological safety and learning behaviour in work teams. *Administrative Science Quarterly, 44*(2), 350-383.

Edmondson, A. C., & Nembhard, I. M. (2009). Product development and learning in project teams: The challenges are the benefits. *Journal of Product Innovation Management, 26*(2), 123-138.

Edmondson, C., & Smith, M. (2006). Too hot to handle? How to manage relationship conflict. *California Management Review, 49*(1), 6-31.

Ein-Dor, P., & Segev, E. (1978). Organisational context and the success of management information systems. *Management Science, 24*(10), 1064-1077.

Eisenhardt, M. (1989). Building theories from case study research. *Academy of Management Review, 14*(4), 532-550.

Ellemers, N., De Gilder, D., & Haslam, S. A. (2004). Motivating individuals and groups at work: a social identity perspective on leadership and group performance. *Academy of Management Review, 29*(3), 459-478.

Elliot, S., & Avison, D. (2005). *Discipline of information systems*. Amsterdam: Elsevier Butterworth Heinemann.

Engum, E. A., Racheva, Z., & Daneva, M. (2009). *Sprint planning with a digital aid tool: lessons learnt.* Paper presented at the software engineering and advanced applications conference, 2009. IEEE Computer Society (pp. 259-262).

Erdogmus, H., Morisio, M., & Torchiano, M. (2005). On the effectiveness of the test-first approach to programming. http://dx.doi.org/10.1109/TSE.2005.37 . *IEEE Transaction on Software Engineering, 31* (3), 226-237.

Ewusi-Mensah, K. (1997). Critical issues in abandoned information systems development project. *Communication of the ACM, 40*(9), 74-80.

Ewusi-Mensah, K. (2003). *Software development failures*. Cambridge: MIT Press.

Ewusi-Mensah, K., & Przasnyski, H. (1994). Factors contributing to the abandonment of information systems development projects. *Journal of Information Technology, 10*, 3-14.

Faraj, S., & Sproull, L. (2000). Coordinating expertise in software development teams. *Management Science, 46(*12), 1553 -1568.

Farrukh, C., Phaal, R., & Probert, D. R. (2001). *Industrial practice in technology planning - implications for a useful tool catalogue for technology management.*

Paper presented at the international conference on management of engineering and technology. IEEE Computer Society (pp. 200 vol.1).

Feldman, L., & Albert, P. (1984). Principle vs. practice in new product planning. *Journal of Product Innovation Management, 1,* 43-55.

Fernandez, D. J., & Fernandez, J. D. (2008). Agile project management - agilism versus traditional approaches. *Journal of Computer Information Systems, 49*(2), 10-17.

Ferrari, R., Madhavji, N. H., & Wilding, M. (2009). The impact of non-technical factors on Software Architecture. http://dx.doi.org/10.1109/LMSA.2009.5074862 . In *proceedings of the 2009 ICSE workshop on leadership and management in software architecture* (pp. 32-36): IEEE Computer Society.

Ferreira, C., & Cohen, J. (2008). Agile systems development and stakeholder satisfaction: a South African empirical study. http://doi.acm.org/10.1145/1456659.1456666 . In *proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: Riding the wave of technology* (pp. 48-55). Wilderness, South Africa ACM.

Finkelstein, S., & Hambrick, D. C. (1990). Top-management-team tenure and organisational outcomes: the moderating role of managerial discretion. *Administrative Science Quarterly, 35*(3), 484-503.

Firth, H., & Britton, P. (1989). 'Burnout', absence and turnover amongst British nursing staff. *Journal of Occupational Psychology, 62*(1), 55-59.

Fitzgerald, B. (1997). The use of systems development methodologies in practice: a field study. *Information Systems Journal, 7*(3), 2001-2212.

Fitzgerald, B. (1998). An empirically-grounded framework for the information systems development process. In *proceedings of the international conference on Information systems* (pp. 103-114). Helsinki, Finland. Association for Information Systems.

Fitzgerald, B., & Howcroft, D. (1998). Competing dichotomies in IS research and possible strategies for resolution In *proceedings of the international conference on Information systems* (pp. 155-164). Helsinki, Finland Association for Information Systems.

Fitzgerald, B., Hartnett, G., & Conboy, K. (2006). Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems 15,* (200–213).

Fitzgerald, B., Russo, N. L., & O'Kane, T. (2003). Software development method tailoring at Motorola. *Communication of the ACM, 46*(4), 65-70.

Flaatten, P. O. (1992). Requirements for a life-cycle methodology for the 1990s. In *Challenges and strategies for research in systems development* (pp. 221-234). John Wiley & Sons.

Fliedner, G., & Vokurka, R. J. (1997). Agility: Competitive weapon of the 1990s and beyond? *Production & Inventory Management Journal, 38*(3), 19-24.

Fogelstrom, N. D., Gorschek, T., Svahnberg, M., & Olsson, P. (2009). The impact of agile principles on market-driven software product development. *Software Process: Improvement and Practice*.

Folkes, S., & Stubenvoll, S. (1992). *Accelerated systems development*. New York: Prentice Hall.

Ford, R. C., & Randolph, W. A. (1992). Cross-functional structures: A review and integration of matrix organisation and project management. *Journal of Management, 18*(2), 267.

Forsyth, D. (1990). *Group Dynamics (4th Ed.)*. Belmont: Thomson/Wadswork.

Forsyth, S. (1997). Human factors in agile manufacturing: a brief overview with emphasis on communication and information infrastructure. *Human Factors and Ergonomics in Manufacturing, 7*(1), 3-10.

FØslstad, A. (2007). Work-Domain Experts as Evaluators: Usability inspection of domain-specific work-support systems. *International Journal of Human-Computer Interaction, 22*(3), 217 - 245.

Fowler, M. (2002). The new methodology. Retrieved January 19th, 2006, from http://www.martinfowler/articles/newMethodology.html.

Frank, A., & Hartel, C. (2009). Feature teams collaboratively building products from ready to done. Paper presented at the agile conference, 2009. IEEE Computer Society (pp. 320-325).

Fraser, J., & Mattu, B. S. (2007). *Test driven design challenges for faster product development.* Paper presented at the 1st annual systems conference. IEEE Computer Society (pp.1 - 5).

Fruhling, A., & Vreede, G.-J. D. (2006). Field experiences with extreme programming: Developing an emergency response system. http://dx.doi.org/10.2753/MIS0742-1222220403 . *Journal of Management Information Systems, 22* (4), 39-68.

Frye, C. (2008). *Software development groups take many routes to* agile, retrieved from http://searchsoftwarequality.techtarget.com/news/article/0,289142,sid92_gci1318987,00.html.

Gat, I. (2006). *How BMC is scaling agile development.* Paper presented at the agile conference, 2006. IEEE Computer Society (pp. 320 - 305).

George, B., & Williams, L. (2003). An initial investigation of test driven development in industry. http://doi.acm.org/10.1145/952532.952753 . In *proceedings of the 2003 ACM symposium on applied computing* (pp. 1135-1139). Melbourne, Florida ACM.

Germain, R., & Robillard, P. N. (2005). Engineering-based processes and agile methodologies for software development: a comparative case study. http://dx.doi.org/10.1016/j.jss.2004.02.022. *Journal of System Software, 75* (1-2), 17-27.

Gerwin, D., & Ferris, J. S. (2004). Organising new product development projects in strategic alliances. *Organisation Science, 15*(1), 22-37.

Gibb, R. (1961). Defensive communication. *Journal of Communication, 11*, 141-148.

Gillham, B. (2000). *Case Study Research Methods*. London: Continum.

Glaser, B. G. (1998.). *Doing Grounded Theory - Issues and Discussions.* (Vol. 11): Sociology Press.

Glass, R. (2004). Matching methodology to problem domain. *Communication of ACM, 47*(5), 19-21.

Goebel, C. J. (2009). *How being agile changed our human resources policies*. Paper presented at the agile conference, 2009. IEEE Computer Society (pp. 101-106).

Golafshani, N. (2003). Understanding reliability and validity in qualitative research. *The Qualitative Report, 8*(4), 597-607.

Goldman, L., Nagel, N., & Preies, K. (1995). *Agile competitors and virtual organisations: strategies for enriching the customer*. New York, NY: Van Nostrand Reinhold.

Goldman, R. N., & Nagel, R. A. (1993). Management technology and agility: the emergence of a new era in manufacturing. *International Journal of Technology, 8,* 18-83.

Gordon, S. S., Stewart, J., Wayne H., Sweo, R., & Luker, W. A. (2000). Convergence versus strategic reorientation: the antecedents of fast-paced organisational change. *Journal of Management, 26*(5), 911-945.

Gorschek, T., & Wohlin, C. (2006). Requirements Abstraction Model. *Requirements Engineering* (Vol. 11, pp. 79-101): Springer London.

Gratton, L., & Erickson, T. J. (2007). 8 ways to build collaborative teams. *Harvard Business Review, 85*(11), 100-109.

Greiner, L., & Schein, V. (1988). *Power and organisation development*. Reading. MA: Addison-Wesley Publishing.

Griffin, B., & Hesketh, B. (2003). Adaptable behaviours for successful work and career adjustment. *Australian Journal of Psychology, 55*(2), 65-73.

Groenveld, P. (2007). Roadmapping integrates business and technology. *Research Technology Management, 50*(6), 49-58.

Gruba, P., & Al-Mahmood, R. (2004). Strategies for communication skills development. In *proceedings of the sixth conference on Australasian computing education* - Volume 30 (pp. 101-107). Dunedin, New Zealand Australian Computer Society, Inc.

Guba, G. E. (1978). *Toward a methodology of naturalistic inquiry in educational evaluation* (Vol. 8). Los Angeles: University of California.

Gunasekaran, A. (1998). Agile manufacturing: enablers and an implementation framework. *International Journal of Production Research, 36*(5), 1223-1247.

Gunasekaran, A. (1999). Agile manufacturing: A framework for research and development. *International Journal of Production Economics, 62*(1/2), 87-105.

Gunasekaran, A., & Yusuf, Y. Y. (2002). Agile manufacturing: a taxonomy of strategic and technological imperatives. *International Journal of Production Research, 40*(6), 1357-1385.

Guzman, I. R., Stanton, J. M., Stam, K. R., Vijayasri, V., Yamodo, I., Zakaria, N., et al. (2004). A qualitative study of the occupational subculture of information systems employees in organisations. In *proceedings of the 2004 SIGMIS conference on computer personnel research: careers, culture, and ethics in a networked environment* (pp. 74-80). Tucson, AZ, USA ACM.

Hackman, J. R., & Wageman, R. (2005). A theory of team coaching. *Academy of Management Review, 30*(2), 269-287.

Hadcroft, P., & Jarratt, D. (2007). Market orientation: An iterative process of customer and market engagement. *Journal of Business-to-Business Marketing, 14*(3), 21-57.

Hage, J., & Dewar, R. (1973). Elite values versus organisational structure in predicting innovation. *Administrative Science Quarterly, 18*(3), 279-290.

Hagedoorn, J. (1995). A note on international market leaders and networks of strategic technology partnering. *Strategic Management Journal, 16*(3), 241-250.

Hall, D. (1987). Careers and socialisation. *Journal of Management, 13*(2), 301-321.

Hall, D. T. (1996). Protean careers of 21st century. *Academy of Management Executive, 10*(4), 8 -16.

Hamel, G., & Prahalad, K. (1994). *Competing for future*. Boston: Harvard Business School Press.

Hanks, B., McDowell, C., Draper, D., & Krnjajic, M. (2004). Program quality with pair programming in CS1. *ACM SIGCSE Bulletin, 36*(3), 176-180.

Hanssen, G. K., & Haugset, B. (2009). *Automated acceptance testing using fit.* Paper presented at the system sciences conference, 2009. IEEE Computer Society (pp. 1-8).

Hansson, C., Dittrich, Y., Gustafsson, B. r., & Zarnak, S. (2006). How agile are industrial software development practices? *Journal of System Software, 79*(9), 1295-1311.

Harmesen, F., Brinkkemper, S., & Oei, H. (1994). Situational method engineering for IS project approaches. In A. Verrijn-Stuart & T. Olle (Eds.), *Methods and associated tools for the IS life cycle* (pp. 169-194). North-Holland: Elsevier.

Harmsen, A. (1997). *Situational method engineering*. University of Twente, Netherlands.

Harmsen, F., Brinkkemper, S., & Oei, J. L. H. (1994). Situational method engineering for informational system project approaches. In *proceedings of the IFIP WG8.1 working conference on methods and associated tools for the information systems life cycle* (pp. 169-194). Elsevier Science Inc.

Haugen, N. C. (2006). *An empirical study of using planning poker for user story estimation.* Paper presented at the agile conference, 2006. IEEE Computer Society (pp. 34 - 42).

Haugen, N. C. (2006). *An empirical study of using planning poker for user story estimation*. Paper presented at the agile 2006 conference. IEEE Computer Society (pp 9).

Heil, M. R. (1999). Preparing technical communicators for future workplaces: a model that integrates teaming, professional communication skills, and a software development process. In *proceedings of the 17th annual international conference on Computer documentation* (pp. 110-119). New Orleans, Louisiana, United States ACM.

Heimgartner, S., & Locke, M. (2006). *A tale of two writing teams.* Paper presented at the agile conference, 2006. IEEE Computer Society (pp. 304 - 309).

Helming, J., Koegel, M., & Hodaie, Z. (2009). Towards automation of iteration planning. In *proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications* (pp. 965-972). Orlando, Florida, USA ACM.

Henderson, R. M., & Clark, K. B. (1990). Architectural innovation: the reconfiguration of existing product technologies and the failure of established firms. *Administrative Science Quarterly, 35*(1), 9-30.

Henderson-Sellers, B. (2003). Method engineering for OO systems development. *Communications of the ACM, 46*(10), 73-78.

Henderson-Sellers, B., Gonzalez-Perez, C., Serour, M. K., & Firesmith, D. G. (2005). Method engineering and COTS evaluation. In *proceedings of the second international workshop on models and processes for the evaluation of off-the-shelf components* (pp. 1-4). St. Louis, Missouri ACM.

Herderson-Sellers, B., & Serour, K. (2005). Creating a dual-agility method: the value of method engineering. *Journal of Database Management, 16*(4), 1-23.

Highsmith, J. (2000). *Adaptive Software development: a collaborative approach to managing complex systems*. New York: Dorset House Publishing.

Highsmith, J. (2002). *Agile software development ecosystem*. Boston: Addison-Wesley.

Highsmith, J. (2004). *Agile project management*. Boston: Addison-Wesley.

Highsmith, J., & Cockburn, A. (2001). Agile software development: the business of innovation. *Computer, 34*(9), 120-127.

Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. IEEE Computer, 34(9), 120-122.

Hildebrand, C. (1991). Managing the aftermath: Orchestration skills needed. *Computerworld, 25*(31), 58.

Hilton, T. (2008). Leveraging operant resources of consumers: improving consumer experiences or productivity? *Marketing Review, 8*(4), 359-366.

Hirsch, M. (2002). Making RUP agile. In *OOPSLA 2002 Practitioners Reports* (pp. 1-ff). Seattle, Washington ACM.

Hirschheim, R., & Klein, H. K. (1989). Four paradigms of information systems development. *Communication of ACM 32* (10), 1199-1216.

Hissey, T. W. (2000). Education and careers 2000. Enhanced skills for engineers. *Proceedings of the IEEE, 88*(8), 1367-1370.

Hitt, M., Ireland, D., & Hoskisson, R. (1997). *Strategic management: competitiveness and globalization, concepts and cases* (2nd Ed.): West Publishing Company.

Hoda, R., Noble, J., & Marshall, S. (2010). Organizing self-organizing teams. In *proceedings of the 32nd ACM/IEEE international conference on software engineering - Volume 1* (pp. 285-294). Cape Town, South Africa: ACM.

Hoegl, M., & Gemuenden, H. G. (2001). Teamwork quality and the success of innovative projects: a theoretical concept and empirical evidence. *Organisation Science, 12*(4), 435-449.

Hoegl, M., & Proserpio, L. (2004). Team member proximity and teamwork in innovative projects. *Research Policy, 33*(8), 1153 -1165.

Hoffer, J., George, J., & Valacich, J. (2008). *Modern systems analysis and design* (5th Ed.). Upper Saddle River, N.J.: Pearson Prentice Hall.

Hofstede, A., & Verhoef, T. (1997). On the feasibility of situational method engineering. *Information Systems, 22*(6/7), 401-422.

Hofstede, G. (1998). Identifying organisational subcultures: an empirical approach. *Journal of Management Studies, 35*(1), 1-12.

Hofstede, G., & Hofstede, J. (2004). *Cultures and organisations: software of the mind*. London: McGraw-Hill.

Hollenbeck, J. R., Ilgen, D. R., LePine, J. A., Colquitt, J. A., & Hedlund, J. (1998). Extending the multilevel theory of team decision making: effects of feedback and experience in hierarchical teams. *Academy of Management Journal, 41*(3), 269-282.

Holmes, A., & Kellogg, M. (2006). *Automating functional tests using selenium*. Paper presented at the agile conference, 2006. IEEE Computer Society (pp 275 - 281).

Hopp, W. J., & Van Oyen, M. P. (2004). Agile workforce evaluation: a framework for cross-training and coordination. *IIE Transactions, 36*(10), 919-940.

Hormozi, A. (2001). Agile manufacturing: the next logical step. *Benchmarking: An International Journal, 8*(2), 132-143.

House, R. J. (1991). The distribution and exercise of power in complex organisations: a meso theory. *The Leadership Quarterly, 2*(1), 23-58.

Huck, S. W., & Sandler, H. M. (1979). *Rival hypotheses: "minute mysteries" for the critical thinker*. London: Harper & Row.

Hulkko, H., & Abrahamsson, P. (2005). *A multiple case study on the impact of pair programming on product quality.* Paper presented at the international conference on software engineering, 2005. IEEE Computer Society (pp. 495 - 504).

Hulkko, H., & Abrahamsson, P. (2005). *A multiple case study on the impact of pair programming on product quality*. Paper presented at the software engineering conference, 2005. IEEE Computer Society (pp. 495-504).

Humphrey, W. S. (2000). *Introduction to team software process*. Massachusetts: Addison-Wesley Longman.

Iacovou, C. L., & Dexter, A. S. (2005). Surviving IT project cancellations. *Communication of the ACM 48*(4), 83-86.

Iivari, J. 1989. Levels of abstraction as a conceptual framework for an information system. In E. Falkenberg & P. Lindgren (Eds.), *information system concepts: an in-depth analysis* (pp 323-352). North-Holland.

Iivari, J., Hirschheim, R., & Klein, K. (2001). A dynamic framework for classifying information systems development methodologies and approaches. *Journal of Management Information Systems, 17*(3), 179-218.

Ingalls, P., & Frever, T. (2009). *Growing an agile culture from value seeds.* Paper presented at the agile conference, 2009. IEEE Computer Society (pp. 119 - 124).

Iyer, S., & Nagi, R. (1997). Automated retrieval and ranking of similar parts in agile manufacturing. *IIE Transactions, 29*(10), 859-876.

Jackson, E., & Schuler, S. (1983). Preventing employee burnout. *Personnel, 60*(2), 58-68.

Jackson, E., Schwab, L., & Schuler, S. (1986). Toward an understanding of the burnout phenomenon. *Journal of Applied Psychology, 71*, 630-640.

Jacques, F. M. (2007). Even Commodities Have Customers. *Harvard Business Review, 85*(5), 110-119.

Jamail, N. (2009). Daily motivation builds momentum. *Supervision, 70*(3), 12-13.

Jassawalla, A. R., & Sashittal, H. C. (1999). Building collaborative cross-functional new product teams. *Academy of Management Executive, 13*(3), 50-63.

Javed, T., Maqsood, M. E., & Durrani, Q. S. (2004). A study to investigate the impact of requirements instability on software defects. *ACM SIGSOFT Software Engineering Notes, 29*(3), 1-7.

Jayaratna, N. (1994). *Understanding and evaluating methodologies: NIMSAD, a systemic framework*. London: McGraw-Hill.

Jin, K., Wang, T., & Palaniappan, A. (2005). Improving the agility of automobile industry supply chain. In *proceedings of the 7th international conference on electronic commerce* (pp. 370-374). Xi'an, China ACM.

Jin-Hai, L., Anderson, R., & Harrison, R. (2003). The evolution of agile manufacturing. *Business Process Management Journal, 9*(2), 170-189.

Jiwen Song, L., Tsui, A. S., & Law, K. S. (2009). Unpacking employee responses to organisational exchange mechanisms: the role of social and economic exchange perceptions. *Journal of Management, 35*(1), 56-93.

Johnson, H. (2004). Chaos 2004, from http://www.standishgroup.com/ .

Judy, K. H., & Krumins-Beens, I. (2007). *Ript: innovation and collective product ownership*. Paper presented at the agile conference, 2007. IEEE Computer Society (p. 316).

Kachaner, N., & Deimler, M. S. (2008). How leading companies are stretching their strategy. *Strategy & Leadership, 36*(4), 40-43.

Kahkonen, T. (2004). *Agile methods for large organisations - building communities of practice.* Paper presented at the agile development conference, 2004. IEEE Computer Society (pp. 2 - 10).

Kajko-Mattsson, M. (2008). Problems in agile trenches. In *proceedings of the second ACM-IEEE international symposium on empirical software engineering and measurement* (pp. 111-119). Kaiserslautern, Germany ACM.

Kalliney, M. (2009). *Transitioning from agile development to enterprise product management agility.* Paper presented at the agile conference, 2009. IEEE Computer Society (pp. 209 - 213).

Kalliney, M. (2009). *Transitioning from agile development to enterprise product management agility*. Paper presented at the agile conference, 2009. IEEE Computer Society (pp. 209-213).

Karhatsu, H., Ikonen, M., Kettunen, P., Fagerholm, F., & Abrahamsson, P. (2010). *Building blocks for self-organising software development teams a framework model and empirical pilot study.* Paper presented at the 2nd international conference on software technology and engineering (ICSTE), 2010. IEEE Computer Society (pp. V1-297 - V1-304).

Karlsson, F., & Ågerfalk, P. (2004). Method configuration: adapting to situational characteristics while creating reusable assets. *Information and Software Technology, 46*, 619-633.

Karlström, D., & Runeson, P. (2003). Scaling extreme programming in a market driven development context. In *proceedings of the 4th international conference on extreme programming and agile processes in software engineering* (pp. 363-365). Geneva, Italy Springer-Verlag.

Kathuria, R., & Partovi, F. Y. (1999). Work force management practices for manufacturing flexibility. *Journal of Operations Management, 18*(1), 21-39.

Katz, R. (1982). The effects of group longevity on project communication and performance. *Administrative Science Quarterly, 27*(1), 81-104.

Katzenbach, J. R., & Douglas, S. (1993). *The wisdom of teams, creating the high-performance organisation*. Boston: Harvard Business School Press.

Kaufmann, R., & Janzen, D. (2003). Implications of test-driven development: A pilot study. In *companion of the 18th annual ACM SIGPLAN conference on object-oriented programming, systems, languages, and applications* (pp. 298-299). Anaheim, CA, USA ACM.

Keenan, F. (2004). *Agile process tailoring and problem analysis*. Paper presented at the 26th international conference on software engineering, 2004. IEEE (pp 45-47).

Keil, M., & Robey, D. (2001). Blowing the whistle on troubled software projects. *Communication of the ACM, 44*(2), 87-93.

Keil, P., & Kuhrmann, M. (2006). An approach to model the return on investment of organisation-wide improvement projects using the concept of external effects. In *proceedings of the 2006 international workshop on economics driven software engineering research* (pp. 19-24). Shanghai, China ACM.

Keller, R. T. (1986). Predictors of the performance of project groups in R & D organisations. *Academy of Management Journal, 29*(4), 101-116.

Kettunen, P. (2009). Adopting key lessons from agile manufacturing to agile software product development-A comparative study. *Technovation, 29*(6-7), 408-422.

Khetan, S., & Fowler, P. (1995). Managing high IC yields with short cycle times. *Paper presented at the 1995 gallium arsenide integrated circuit (GaAs IC) symposium*. IEEE Computer Society (pp. 119-123).

Kim, S., Park, S., Yun, J., & Lee, Y. (2008). Automated continuous integration of component-based software: An industrial experience. In *proceedings of the 2008 23rd IEEE/ACM international conference on automated software engineering* (pp. 423-426): IEEE Computer Society.

Kinoshita, F. (2008). *Practices of an agile team*. Paper presented at the agile. Conference, 2008. IEEE Computer Society (pp. 373-377).

Klein, H., & Canditt, S. (2008). Using opinion polls to help measure business impact in agile development. In *proceedings of the 1st international workshop on business impact of process improvements* (pp. 25-32). Leipzig, Germany ACM.

Kolodny, F. (1981). Matrix organisation designs and new product success. *Research Management, 23*(5), 29-33.

Korkala, M., Abrahamsson, P., & Kyllonen, P. (2006). *A case study on the impact of customer communication on defects in agile software development.* Paper presented at the agile conference, 2006. IEEE Computer Society (pp. 88 - 98).

Kraut, R. E., & Streeter, L. (1995). Coordination in software development. *Communication of ACM, 38*(3), 69 - 81.

Ktata, Q., & Levesque, G. (2009). Agile development: issues and avenues requiring a substantial enhancement of the business perspective in large projects. In p*roceedings of the 2nd Canadian conference on computer science and software engineering* (pp. 59-66). Montreal, Quebec, Canada: ACM.

Kumar, K., & Welke, R. (1992). Methodology engineering: a proposal for situation-specific methodology construction in challenges and strategies for research in systems development. In W. W. Cotterman & A. J. Senn (Eds.), w*iley series in Information Systems* (pp. 257-268). Chichester: John Wiley & Sons.

Kusiak, A., & He, W. (1997). Design for an enterprise. In K. Kosanke & G. Nell (Eds.), *Building International Consensus* (pp. 420-430). Berlin: Springer-Verlag.

Kuzel, A. J. (1999). Sampling in qualitative inquiry. In B. F. Crabtree & W. L. Miller (Eds.), *Doing Qualitative Research* (pp. 33-45). London: Sage Publications.

Lamoreux, M. (2005). *Improving agile team learning by improving team reflections [agile software development]*. Paper presented at the agile conference, 2005. IEEE Computer Society (pp. 139 - 144).

Langley, A. (1999). Strategies for theorizing from process data. *Academy of Management Review, 24*(4), 691-710.

Laplante, P. A. (2003). Stand and deliver: Why I hate stand-up meetings. *Queue 1* (7), 7-9.

Larman, C., & Basili, V. R. (2003). Iterative and incremental developments: a brief history. *Computer, 36*(6), 47-56.

Larson, W., & Gobeli, H. (1987). Matrix management: contradictions and insights. *California Management Review, 29*(4), 126-138.

Larson, W., & Gobeli, H. (1988). Organizing for product development projects. *Journal of Product Innovation Management, 5*(2), 180-190.

Latham, G. P., Winters, D. C., & Locke, E. A. (1994). Cognitive and motivational effects of participation: a mediator study. *Journal of Organisational Behaviour, 15*(1), 49-63.

Law, A., & Ho, A. (2004). *A study case: evolution of co-location and planning strategy.* Paper presented at the agile development conference, 2004. IEEE Computer Society (pp. 56 - 62).

Law, A., & Learn, S. (2005). *Waltzing with changes [agile software development]*. Paper presented at the agile conference, 2005. IEEE Computer Society (pp 279 - 285).

LeCompte, M. D., & Goetz, J. P. (1982). Problems of reliability and validity in ethnographic research. *Review of Educational Research, 52*, 38-54.

Lee, A. (1989). Case studies as natural experiments. *Human Relations, 42*(2), 117-137.

Lee, A., & Baskerville, R. (2003). Generalizing generalizability in information systems research. *Information Systems Management, 14*(3), 221-243.

Lee, F. (1997). When the going gets tough, do the tough ask for help? Help seeking and power motivation in organisations. *Organisational Behaviour & Human Decision Processes, 72*(3), 336-363.

Lee, G., & Xia, W. (2010). Toward agile: An integrated analysis of quantitative and qualitative field data on software development agility. *MIS Quarterly, 34(1),* 87-114.

Leedy, D. P., & Ormrod, E. J. (2001). *Practical research: planning and design* (7th Ed.). New Jersey: Merrill Prentice Hall.

Lehto, I., & Rautiainen, K. (2009) *Software development governance challenges of a middle-sized company in agile transition.* Paper presented at the Software Development Governance, 2009. IEEE Computer Society (pp. 36 - 39).

Lehto, I., & Rautiainen, K. (2009). Software development governance challenges of a middle-sized company in agile transition. http://dx.doi.org/10.1109/SDG.2009.5071335 . In *proceedings of the 2009 ICSE workshop on software development governance* (pp. 36-39): IEEE Computer Society.

Lehtola, L., Kauppinen, M., & Kujala, S. (2005). *Linking the business view to requirements engineering: long-term product planning by roadmapping.* Paper presented at the 13th IEEE international conference on requirements engineering, 2005. IEEE Computer Society (pp. 439 - 443).

Lei, D., Hitt, M. A., & Bettis, R. (1996). Dynamic core competences through meta-learning and strategic context. *Journal of Management, 22*(4), 549.

Leitheiser, R. L. (1992). MIS skills for the 1990s: a survey of MIS managers' perceptions. *Journal of Management Information Systems, 9*(1), 69-91.

Leithiser, R., & Hamilton, D. (2008). Agile versus CMMI - process template selection and integration with Microsoft team foundation server. http://doi.acm.org/10.1145/1593105.1593153. In proceedings *of the 46th*

annual southeast regional conference on XP (pp. 186-191). Auburn, Alabama ACM.

Leonard-Barton, D. (1988). Implementation as mutual adaptation of technology and organisation. *Research Policy, 17*(5), 251-267.

Leonard-Barton, D. (1992). Core capabilities and core rigidities: A paradox in managing new product development. *Strategic Management Journal, 13*(S1), 111-125.

Levine, J. M., & Moreland, R. L. (1990). Progress in small group research. Annual *Review of Psychology, 41*(1), 585.

Levinthal, D. A., & March, J. G. (1993). The myopia of learning. *Strategic Management Journal, 14*, 95-112.

Lewicki, J., Barry, B., & Saunders, D. (1995). *Negotiation*. Boston: McGraw-Hill/Irwin.

Lewis, T. L., Smith, W. J., Bélanger, F., & Harrington, K. V. (2008). Are technical and soft skills required: the use of structural equation modeling to examine factors leading to retention in the CS major. http://doi.acm.org/10.1145/1404520.1404530. In *proceeding of the fourth international workshop on computing education research* (pp. 91-100). Sydney, Australia ACM.

Lincoln, Y. S., & Guba, G. E. (1985). *Naturalistic Inquiry*. Beverly Hills: Sage Publication.

Lincoln, Y. S., & Guba, G. E. (2000). The only generalization is: there is no generalization. In R. Gomm, M. Hammersley & P. Foster (Eds.), c*ase study method*. London: Sage Publication.

Lindgren, M., Land, R., Norstrom, C., & Wall, A. (2008). Key aspects of software release planning in industry. In *proceedings of the 19th Australian conference on software engineering* (pp. 320-329): IEEE Computer Society.

Lindgren, M., Wall, A., Land, R., & Norstrom, C. (2008). *A method for balancing short- and long-term investments: quality vs. features*. Paper presented at the software engineering and advanced applications conference, 2008. IEEE Computer Society (pp. 175-182).

Lindvall, M., Basili, V. R., Boehm, B. W., Costa, P., Dangle, K., Shull, F., et al. (2002). Empirical Findings in Agile Methods. In p*roceedings of the second XP universe and first agile universe conference on extreme programming and agile methods - XP/Agile Universe 2002* (pp. 197-207): Springer-Verlag.

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., et al. (2004). Agile software development in large organisations. *Computer, 37*(12), 26-34.

Little, T. (2005). Context-adaptive agility: managing complexity and uncertainty. *Software, IEEE, 22*(3), 28-35.

Liu, L., Erdogmus, H., & Maurer, F. (2005). *An environment for collaborative iteration planning*. Paper presented at the agile conference, 2005. IEEE Computer Society (pp. 80-89).

Locke, E. A., & Schweiger, D. M. (1979). Participation in decision-making: one more look. *Research in organisational behaviour, 1*, 265.

Lovelace, K., Shapiro, D. L., & Weingart, L. R. (2001). Maximizing cross-functional new product teams' innovativeness and constraint adherence: a conflict communications perspective. *Academy of Management Journal, 44*(4), 779-793.

Lui, K. M., & Chan, K. C. C. (2008). Software process fusion by combining pair and solo programming. *Software, IET, 2*(4), 379-390.

Lui, M., Chan, C., & Nosek, J. (2008). The effects of pairs in program design tasks. *IEEE Transaction on Software Engineering, 34*(1), 197-211.

Lukanuski, M., Milano, M., Bruin, J. d., Rochford, M., & Bosman, R. (2008). Agile or awkward: surviving and flourishing in an agile/scrum project. http://doi.acm.org/10.1145/1358628.1358662. In *CHI '08 extended abstracts on human factors in computing systems* (pp. 2253-2256). Florence, Italy ACM.

Lukens, S. (2007). The agile CFO: change agent. *Financial Executive (April)*, 30-32.

Lyytineen, K. (1987a). Different perspectives on information systems: problems and solutions. *ACM Computing Surveys, 19*(1), 5-46.

Lyytineen, K. (1987b). A taxonomic perspective of information systems development: theoretical constructs and recommendations. *In critical issues in information systems research* (pp. 3-41).

Maddox, M. E. (1990). Communication skills needed by first-line managers. *Manage, 42*(2), 11-13.

Madison, J. (2010). Agile Architecture Interactions (was: Agile Architecture Insertion Points). *Software, IEEE, pp*(99), 1.

Mahaney, R. C., & Lederer, A. L. (1999). Runaway information systems projects and escalating commitment. http://doi.acm.org/10.1145/299513.299719. In *proceedings of the 1999 ACM SIGCPR conference on computer personnel research* (pp. 291-296). New Orleans, Louisiana, United States ACM.

Majchrzak, A., Malhotra, A., & John, R. (2005). Perceived individual collaboration know-how development through information technology-enabled contextualization: evidence from distributed teams. *Information Systems Research, 16*(1), 9-27.

Mankin, D., Susan, C., & Bikson, T. (1996). *Teams and technology: fulfilling the promise of the new organisation*. Boston: Harvard Business School Press.

Manz, C. C., & Sims Jr., H. P. (1991). Superleadership: beyond the myth of heroic leadership. *Organisational Dynamics, 19*(4), 18-35.

Maples, C. (2009). *Enterprise agile transformation: the two-year wall.* Paper presented at the agile conference, 2009. IEEE Computer Society (pp.90 - 95).

Maqsood, M., & Javed, T. (2007). Practicum in software project management: an endeavour to effective and pragmatic software project management education. http://doi.acm.org/10.1145/1295014.1295022. In *the 6th joint meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering: companion papers* (pp. 471-479). Dubrovnik, Croatia.

March, J. (1991). Exploration and exploitation in organisational learning. *Organisation Science, 2(*1), 71-86.

March, J. G. (1981). Footnotes to organisational change. *Administrative Science Quarterly, 26*(4), 563-577.

Marino, K. (1982). Innovation, adoption, and structural dilemma. *Journal of Management, 8*(1), 75-93.

Marino, K. E. (1980). A preliminary investigation into the behavioral dimensions of affirmative action compliance. *Journal of Applied Psychology, 65*(3), 346-350.

Markham, D. B. (2009). *Agile won't work: implementing agility in non-standard teams.* Paper presented at the agile conference, 2009. IEEE Computer Society (pp. 338 - 343).

Marks, M. A., Mathieu, J. E., & Zaccaro, S. J. (2001). A temporally based framework and taxonomy of team processes. *Academy of Management Review, 26*(3), 356-376.

Markus, L., & BjØrn-Anderson, N. (1987). Power over users: its exercise by system professionals. *Communication of the ACM, 30*(6), 498 -504.

Markus, M. L. (1997). The qualitative difference in information systems research and practice. In *proceedings of the IFIP TC8 WG 8.2 international conference on information systems and qualitative research* (pp. 11-27). Philadelphia, Pennsylvania, United States.

Marsh, C. H. (1999). The engineer as technical writer and document designer: the new paradigm. http://doi.acm.org/10.1145/311147.311159. *SIGDOC Asterisk Journal of Computer Documentation, 23* (2), 57-61.

Marshall, C., & Rossman, B. G. (1999). *Designing qualitative research*. London: Sage Publications.

Martin, A., Biddle, R., & Noble, J. (2003). *Being Jane Malkovich: A look into the World of an XP Customer*. Paper presented at the XP2003., Genoa, Italy.

Martin, A., Biddle, R., & Noble, J. (2004). *The XP customer role in practice: three studies.* Paper presented at the agile development conference, 2004. IEEE Computer Society (pp. 42 - 54).

Martin, A., Biddle, R., & Noble, J. (2009). *The XP customer team: a grounded theory*. Paper presented at the agile conference, 2009. IEEE Computer Society (pp. 57-64).

Martin, A., Noble, J., & Biddle, R. (2006). Programmers are from Mars, customers are from Venus: a practical guide for customers on XP projects. http://doi.acm.org/10.1145/1415472.1415496. In *proceedings of the 2006 conference on pattern languages of programs* (pp. 1-9). Portland, Oregon.

Martin, R. C. (2005). The test bus imperative: architectures that support automated acceptance testing. *Software, IEEE, 22*(4), 65-67.

Mathiassen, L., Pries-Heje, J., & Ngwenyama, O. (2002). *Improving software organisations: from principles to practice*. Upper Saddle River, NJ: Addison-Wesley.

Mathieu, R. (1997). *Manufacturing and Internet*. Paper presented at the American production and inventory control society, VA.

Matusik, S. F., & Heeley, M. B. (2005). Absorptive capacity in the software industry: identifying dimensions that affect knowledge and knowledge creation activities. *Journal of Management, 31*(4), 549-572.

Maximilien, E. M., & Williams, L. (2003). Assessing test-driven development at IBM. *In proceedings of the 25th international conference on software engineering* (pp. 564-569). Portland, Oregon IEEE Computer Society.

Maxwell, A. J. (2005). *Qualitative research design: an interactive approach* (Vol. 41). London: Sage Publications.

Maznevski, M. L., & Chudoba, K. M. (2000). Bridging space over time: global virtual team dynamics and effectiveness. *Organisation Science, 11*(5), 473 - 492.

McChesney, I. R., & Gallagher, S. (2004). Communicating and co-ordination practices in software engineering projects. *Information and software Technology, 46,* 473-498.

McConnell, S. (2004). *Professional software development*. Boston: Addison-Wesley.

McDowell, C., L, W., Bullock, H. E., & Fernald, J. (2003). *The impact of pair programming on student performance, perception and persistence*. Paper presented at the 25th International conference on software engineering. IEEE Computer Society (pp. 602-607).

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). The effects of pair programming on performance in an introductory programming course. *ACM SIGCSE Bulletin, 34*(1), 38-42.

McGuire, M. (1986). Team software development techniques. http://doi.acm.org/10.1145/317210.317234. In *proceedings of the twenty-second annual computer personnel research conference on computer personnel research conference* (pp. 163-168). Calgary, Canada ACM.

Medhat, S. S., & Rook, J. L. (1997). *Concurrent engineering-processes and techniques for the agile manufacturing enterprise*. Paper presented at the fifth international conference on factory 2000 - the technology exploitation process. IEEE Computer Society (pp. 9-14).

Melnik, G., & Maurer, F. (2004). *Direct verbal communication as a catalyst of agile knowledge sharing.* Paper presented at the agile development conference, 2004. IEEE Computer Society (pp. 21 - 31).

Mendes, E., Al-Fakhri, L. B., & Luxton-Reilly, A. (2005). Investigating pair-programming in a 2nd year software development and design computer science course. In *proceedings of the 10th annual SIGCSE conference on innovation and technology in computer science education* (pp. 296-300). Caparica, Portugal ACM.

Meredith, S., & Francis, D. (2000). Journey towards agility: The agile wheel explored. *The TQM Magazine, 12*(2), 137-143.

Miles, M., & Huberman, A. (1984). *Qualitative data analysis: a sourcebook of new methods*. Beverley Hills: Sage Publication.

Millar, V. (1994). Outsourcing trends. Paper presented at the outsourcing, cosourcing and insourcing conference, University of California - Barkeley.

Miller, D. (1993). The architecture of simplicity. *Academy of Management Review*, 18(1), 116- 138.

Miller, D., & Chen, M.-J. (1996). The simplicity of competitive repertoires: an empirical analysis. *Strategic Management Journal, 17*(6), 419-439.

Miller, D., & Friesen, P. H. (1980). Momentum and revolution in organisational adaptation. *The Academy of Management Journal, 23*(4), 591-614.

Miller, D., Lant, T. K., Milliken, F. J., & Kom, H. J. (1996). The evolution of strategic simplicity: exploring two models of organisational adaptation. *Journal of Management, 22*(6), 863-887.

Miller, K. I., & Monge, P. R. (1986). Participation, satisfaction, and productivity; a meta-analytic review. *Academy of Management Journal, December*, 727-752.

Miller, K., & Crabtree, B. (1999). *A multimethod typology and qualitative roadmap* (2nd Ed.). London: Sage Publications.

Milliken, J., & Lant, T. (1991). The impact of an organisation's recent performance history on strategic persistence and change. *Advances in Strategic Management, 7,* 129-156.

Mirbel, I., & Ralytë, J. (2005). Situational method engineering: combining assembly-based and roadmap-driven approaches. *Requirements Engineering, 11*(1), 58-78.

Misic, V. B. (2006). Perceptions of extreme programming: an exploratory study. *SIGSOFT Software Engineering Notes, 31*(2), 1-8.

Moe, N. B., & Aurum, A. (2008). *Understanding decision-making in agile software development: a case-study.* Paper presented at the 34th Euromicro Conference on Software Engineering and Advanced Applications, 2008. IEEE Computer Society (pp. 216 - 223).

Moe, N. B., Dingsoyr, T., & Dyba, T. (2008). *Understanding self-organising teams in agile software development.* Paper presented at the 19th Australian conference on software engineering, 2008. IEEE Computer Society (pp. 76 - 85).

Moir, S. (2008). The delicate business of nurturing a better culture. *People Management, 14*(18), 19.

Molokken-Ostvold, K., & Furulund, K. M. (2007). *The relationship between customer collaboration and software project overruns.* Paper presented at the agile development Conference, 2007. IEEE Computer Society (pp. 72 - 83).

Moløkken-Østvold, K., & Haugen, C. (2007). *Combining estimates with planning poker--an empirical study*. Paper presented at the software engineering conference, 2007. IEEE Computer Society (pp. 349-358).

Mookerjee, V. S., & Chiang, R. I. (2002). A dynamic coordination policy for software systems construction. *IEEE Transactions on Software Engineering, 28*(6), 684 - 694.

Moore, E. (2009). *Influence of large-scale organszation structures on leadership behaviours.* Paper presented at the agile conference, 2009. IEEE Computer Society (pp. 309 - 313).

Moore, E., & Spens, J. (2008). *Scaling agile: finding your agile tribe.* Paper presented at the agile conference, 2008. IEEE Computer Society (pp. 121 - 124).

Morien, R. (2005). *Agile management and the Toyota way for software project management.* Paper presented at the 3rd IEEE international conference on industrial informatics, 2005. IEEE Computer Society (pp. 516 - 522).

Morien, R. (2005). *Agile management and the Toyota way for software project management*. Paper presented at the industrial informatics conference, 2005. IEEE (pp. 516-522).

Mortensen, M. (2004.). *Antecedents and consequences of team boundary disagreement*. Retrieved March 12, 2005, from http://www.aom.pace.edu/amj/.

Muller, M. M., & Hagner, O. (2002). Experiment about test-first programming. Software, *IEE Proceedings, 149(5)*, 131-136.

Murawski, M., Olds, B. M., & Miller, R. L. (1996). *The department of energy's technical leadership development program: lessons learned from assessment and evaluation*. Paper presented at the frontiers in education conference, 1996. IEEE Computer Society (pp. 306-209).

Myers, D. M., & Avison, D. (2002). *Qualitative research in information systems*. London: Sage Publications.

Nadler, D. A., & Tushman, M. L. (1989). Organisational frame bending: principles for managing reorientation. *Academy of Management Executive, 3*(3), 194-204.

Nagel, N., & Bhargava, P. (1994). Agility: The ultimate requirements for word-class manufacturing performance. *National Productivity Review, 13*(3), 331-340.

Nathan-Ragu, B. S., Apigian, C. H., Nathan-Ragu, T. S., & Tu, Q. (2004). A path analytic study of the effect of top management support for information systems performance. *The International Journal of Management Science, 32*, 459-471.

Nemuraite, L., & Paradauskas, B. (2004). From use case to well structured conceptual schemas. In O. Vasilecus, A. Caplinskas, W. Wojtkowski, W. G. Wojtkowski, J. Zupanèiè & S. Wrycza (Eds.), *information system development advances in theory, practice and education* (pp. 303-485): Springer.

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communication of ACM, 48*(5), 72-78.

Ngo-The, A., & Ruhe, G. (2009). Optimized resource allocation for software release planning. *IEEE Transactions on Software Engineering, 35*(1), 109-123.

Nguyen, P., & Henderson-Sellers, B. (2003). *Towards automated support for method engineering with OPEN approach*. Paper presented at the seventh IASTED international conference on software engineering and application, Cambridge, USA.

Nidumolu, S. (1995). The effect of coordination and uncertainty on software project performance: residual performance risk as an intervening variable. *Information Systems Research, 6*(3), 191 - 219.

Nonaka, I., Toyama, R., & Byosiare, P. (2001). A theory of organisational knowledge creation: Understanding the dynamic process of creating knowledge. In *handbook of organisational learning & knowledge* (pp. 491-517): Meinolf Dierkes.

Nosek, J. (1998). The case for collaborative programming. *Communication of ACM, 41*(3), 105-108.

Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: a roadmap. http://doi.acm.org/10.1145/336512.336523. In *proceedings of the conference on the future of software engineering* (pp. 35-46). Limerick, Ireland.

Oinas-Kukkonen, H. (1996). A proposal for context-specific method engineering. In S. Brinkkemper, K. Lyytineen & R. Walke (Eds.), *method engineering: principles of method construction and support* (pp. 191-208). London: Chapman and Hall.

Olle, A., Hagelstein, J., Macdonald, C., Rolland, C., Sol, H., Assche Van, F., et al. (1988). *Information systems methodologies : a framework for understanding*. Workingham: Addision Wesley.

Olle, W. T., Hagelstein, J., Macdonald, G. I., Rolland, C., Sol, G. H., Assche, J. M. F., et al. (1991). *Information systems methodologies: a framework for understanding* (2nd Ed.). Wokingham, England: Addison-Wesley.

Opelt, K. (2008). *Overcoming Brooks' law*. Paper presented at the agile conference, 2008. IEEE Computer Society (pp. 208 - 211).

Orsted, M. (2000). *Software development engineer in Microsoft. A subjective view of soft skills required.* Paper presented at the international conference on software engineering, 2000. IEEE Computer Society (pp. 539 - 540).

Ørvik, T. U., Olsen, D. H., & Sein, M. K. (Eds.). (1999). Deployment of system development methods. In J.Zupancic, W. Wojtkowski, W.G. Wojtkowski and S. Wrycza (eds.), *evolution and challenges in system development*. (pp. 3-31). New York: Kluwer Academic / Plenum Publishers.

Owusu, A. (1999). Importance of employee involvement in world-class agile management systems. *International Journal of Agile Management Systems, 1*(2), 107 - 115.

Paasivaara, M., & Lassenius, C. (2006). *Could global software development benefit from agile methods?* Paper presented at the international conference on global software engineering, 2006. IEEE Computer Society (pp. 109 - 113).

Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2009). *Using scrum in distributed agile development: a multiple case study*. Paper presented at the global software engineering conference, 2009. IEEE (pp. 195-204).

Padberg, F., & Muller, M. M. (2003). *Analyzing the cost and benefit of pair programming.* Paper presented at the ninth international software metrics symposium, 2003. IEEE Computer Society (pp. 166 - 177).

Palmer, R., & Felsing, M. (2002). *A practical guide to feature-driven development*. Upper Saddle River, NJ: Prentice-Hall.

Paré, G., & Elam, J. J. (1997). Using case study research to build theories if IT Imlementation. In S. A. Lee, J. Liebenau & I. J. DeGross (Eds.), *information systems and qualitative research* (pp. 542-568). London: Chapman & Hall.

Park, S. S., & Maurer, F. (2008). The benefits and challenges of executable acceptance testing. http://doi.acm.org/10.1145/1370143.1370148. In *proceedings of the 2008 international workshop on scrutinizing agile practices or shoot-out at the agile corral* (pp. 19-22). Leipzig, Germany.

Parkinson, S. (1999). Agile manufacturing. *Work Study, 48*(4), 134-137.

Parrish, A., Smith, R., Hale, D., & Hale, J. (2004). A field study of developer pairs: productivity impacts and implications. *Software, IEEE, 21*(5), 76-79.

Parsons, D., Ryu, H., & Lal, R. (2007). *The impact of methods and techniques on outcomes from agile software development projects*. Paper presented at the IFIP 8.6 conference: organisational dynamics of technology-based innovation: diversifying the research Agenda, Manchester, UK.

Patton, Q. M. (1987). *How to use qualitative methods in evaluation*. Newbury Park: Sage Publications.

Patton, Q. M. (1990). *Qualitative evaluation and research methods* (2nd Ed.). London: Sage Publications.

Pelz, D. C. (1981). *Use of innovation in innovating processes by local government.* University of Michigan.

Pettichord, B. (2002). *Agile testing, what is it? Can it work?* Retrieved 12th May, 2006, from http://www.io.com/~wazmo/papers/agile_testing_20021015.pdf.

Phaal, R., Farrukh, C. J. P., & Probert, D. R. (2005). Technology roadmapping - a planning framework for evolution and revolution. *Technological Forecasting and Social Change, 71*, 5-26.

Phalnikar, R., Deshpande, V. S., & Joshi, S. D. (2009). *Applying agile principles for distributed software development.* Paper presented at the international conference on advanced computer control, 2009. IEEE Computer Society (pp. 535 - 539).

Pillai, R., & Meindl, J. R. (1998). Context and charisma: A "meso" level examination of the relationship of organic structure, collectivism, and crisis to charismatic leadership. *Journal of Management, 24*(5), 643-671.

Pitman, B. (1994). Stop wasting training dollars: Train for outcomes, not outputs. *Journal of Systems Management, 45*(6), 25.

Plonka, S. (1997). Developing a lean and agile work force. *Human Factors and Ergonomics in Manufacturing, 6*(4), 324-349.

Podsakoff, P. M., & MacKenzie, S. B. (1997). Impact of organisational citizenship behaviour on organisational performance: a review and suggestion for future research. *Human Performance, 10*(2), 133.

Poppendieck, M. (2004). The role of the customer in software development: the XP customer - fad or fashion?. http://doi.acm.org/10.1145/1028664.1028723. In *companion to the 19th annual ACM SIGPLAN conference on object-oriented programming systems, languages, and applications* (pp. 148-150). Vancouver, BC, CANADA.

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: an agile toolkit*. Boston: Addison-Wesley.

Powell, W. (1987). New form or transitional development. *California Management Review, Fall*, 67-87.

Prahalad, K., & Hamel, G. (1990). The core competence of corporation. Harvard Business Review, May-June, 1-15.

Pressman, S. (2005). *Software engineering: a practitioner's approach* (6th edition). New York: McGraw-Hill.

Pulakos, E. D., Arad, S., Donovan, M. A., & Plamondon, K. E. (2000). Adaptability in the workplace: development of a taxonomy of adaptive performance. *Journal of Applied Psychology, 85*(4), 612-624.

Qureshi, M. R. J., & Kashif, M. (2009). *Seamless long term learning in agile teams for sustainable leadership.* Paper presented at the international conference on emerging technologies, 2009. IEEE Computer Society (pp 389 - 394.).

Raatikainen, M., Rautiainen, K., Myllärniemi, V., & Männistö, T. (2008). Integrating product family modeling with development management in agile methods. http://doi.acm.org/10.1145/1370720.1370728. In *proceedings of the 1st international workshop on software development governance* (pp. 17-20). Leipzig, Germany: ACM.

Ralyte, J. (1999). Reusing scenario based approaches in requirement engineering methods: CREWS method base. Paper presented at the database and expert systems applications 1999 conference. IEEE Computer Society (pp. 305-309).

Ralyte, J., & Rolland, C. (2001). An assembly process model for method engineering. In Proceedings of the 13th international conference on advanced information systems engineering (pp. 267-283): Springer-Verlag.

Ramesh, G., & Devadasan, S. R. (2007). Literature review on the agile manufacturing criteria. *Journal of Manufacturing Technology Management*, 18(2), 182-201.

Randolph, A., & Posner, B. (1992). *Getting the job done: managing project teams and task forces for success*. Englewood Cliffs, NJ: Prentice-Hall.

Rautiainen, K., Vuornos, L., & Lassenius, C. (2003). *An experience in combining flexibility and control in a small company's software product development process*. Paper presented at the 2003 empirical software engineering conference. IEEE Computer Society (pp. 28-37).

Raymond, L. (1990). Organisational context and information systems success: a contingency approach. *Journal of Management Information Systems, 6(*4), 5-20.

Reed, K., & Blunsdon, B. (1998). Organisational flexibility in Australia. *The International Journal of Human Resource Management, 9*(3), 457 - 477.

Reel, J. (1999). Critical success factors in software projects. *IEEE Software, May/June*, 18-22.

Rehesaar, H., & Beames, E. (1998). *Project plans and time budgets in information systems projects*. Paper presented at the 1998 software engineering: education & practice conference. IEEE Computer Society (pp. 120-124).

Reid, M., & Hammersley, R. (2000). *Communicating successfully in groups: a practical guide for the workplace*. London: Routledge.

Reifer, D. J. (2002). How good are agile methods? *Software, IEEE, 19*(4), 16-18.

Reilly, R. R., Chen, J., & Lynn, G. S. (2003). *Power and empowerment: the role of top management support and team empowerment in new product development.* Paper presented at the international conference on management of engineering and technology, 2003. IEEE Computer Society (pp. 282 - 289).

Remenyi, D., & Sherwood-Smith, M. (1998). Business benefits from information systems through an active benefits realisation programme. International *Journal of Project Management, 16*(2), 81-98.

Ren, J., Yusuf, Y. Y., & Burns, N. D. (2003). The effects of agile attributes on competitive priorities: a neural network approach. *Integrated Manufacturing Systems, 14*(6), 489.

Rendell, A. (2008). *Effective and pragmatic test driven development*. Paper presented at the agile conference, 2008. IEEE Computer Society (pp. 298-303).

Rising, L., & Janoff, N. S. (2000). The scrum software development process for small teams. *Software, IEEE, 17*(4), 26-32.

Rivard, S., Raymond, L., Bergeron, F. o., & Aubin, M-C. (1999). Project manager's influence tactics and authority: a comparison across project structures. http://doi.acm.org/10.1145/568498.568499. *SIGCPR Computer Personnel, 19/20* (4/1), 6-20.

Rosenberg, D., Stephens, M., & Collins-Cope, M. (2005). *Agile development with ICONIX process*. Berkeley, CA: Apress.

Ross, J. W., Beath, C. M., & Goodhue, D. L. (1996). Develop long-term competitiveness through IT assets. *Sloan Management Review, Fall*, 31-42.

Rubart, J., & Freykamp, F. (2009). Supporting daily scrum meetings with change structure. http://doi.acm.org/10.1145/1557914.1557927. In *proceedings of the 20th ACM conference on Hypertext and hypermedia* (pp. 57-62). Torino, Italy ACM.

Ruhnow, A. (2007). *Consciously evolving an agile team.* Paper presented at the agile conference, 2007. IEEE Computer Society (pp.130 - 135).

Runnker, G., & Brache, A. (1995). *Improving performance: how to manage the white space on the organisational chart*. San Francisco: Jossey-Bass.

Russell, R. D., & Russell, C. J. (1992). An examination of the effects of organisational norms, organisational structure, and environmental uncertainty on entrepreneurial strategy. *Journal of Management, 18*(4), 639-656.

Saliu, O., & Ruhe, G. (2005). *Supporting software release planning decisions for evolving systems*. Paper presented at the software engineering workshop, 2005. IEEE Computer Society (pp. 14-26).

Salo, O., & Abrahamsson, P. (2005). *Integrating agile software development and software process improvement: a longitudinal case study.* Paper presented at the international symposium on empirical software engineering, 2005. IEEE Computer Society (pp.193-212).

Salo, O., & Abrahamsson, P. (2008). Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of extreme programming and scrum. *Software, IET, 2*(1), 58-64.

Sanders, G. L., & Courtney, J. F. (1985). A field study of organisational factors influencing DSS success. *MIS Quarterly, March*, 77-93.

Sarin, S., & Mahajan, V. (2001). The effect of reward structures on the performance of cross-functional product development teams. *Journal of Marketing, 65*(2), 35-53.

Satzinger, J., Jackson, R., & Burd, S. (2005). *Object-oriented analysis*. Boston: Thomson.

Savolainen, J., Kuusela, J., & Vilavaara, A. (2010). *Transition to agile development - rediscovery of important requirements engineering practices.* Paper presented at the 18th IEEE international requirements engineering conference (RE), 2010. IEEE Computer Society (pp. 89 - 294).

Sawyer, S. (1997). Supporting the social processes of software development. *Information Technology & People, 10*(1), 46-62.

Sawyer, S. (2004). Software development teams. *Communications of the ACM, 47*(12), 95-99.

Schach, S. (2004). *Introduction to object-oriented analysis and design with UML and the unified process*. Boston: McGraw Hill.

Schatz, B., & Abdelshafi, I. (2006). *The agile marathon.* Paper presented at the agile conference, 2006. IEEE Computer Society (pp. 289 - 294).

Schilling, M. A. (2009). Understanding the alliance data. *Strategic Management Journal, 30*(3), 233-260.

Schmitt, J. W., & Kozar, K. A. (1978). Management's role in information system development failures: A case study. *MIS Quarterly, June*, 7-16.

Schwaber, K., & Beedle, M. (2002). *Agile software development with scrum*. Upper Saddle River, N.J.: Prentice Hall.

Schwandt, T. (1997). *Qualitative inquiry: a dictionary of terms*. Thousand Oaks: Sage Publications.

Seger, T., Hazzan, O., & Bar-Nahor, R. (2008). *Agile orientation and psychological needs, self-efficacy, and perceived support: a two job-level comparison.* Paper presented at the agile conference, 2008. IEEE Computer Society (pp 3 - 14).

Seiler, W. J., & Beall, M. L. (2005). *Communication: making connections*. Boston: Pearson/Allyn & Bacon.

Sfetsos, P., Angelis, L., & Stamelos, I. (2006). Investigating the extreme programming system: an empirical study. *Empirical Software Engineering, 11*(2), 269-301.

Shane, S., Venkataraman, S., & MacMillan, I. (1995). Cultural differences in innovation championing strategies. *Journal of Management, 21*(5), 931-952.

Sharifi, H., & Zhang, Z. (1999). A methodology for achieving agility in manufacturing organisations: an introduction. *International Journal of Production Economics, 62*(1-2), 7-22.

Sharp, J. M., Irani, Z., & Desai, S. (1999). Working towards agile manufacturing in the UK industry. *International Journal of Production Economics, 62*(1/2), 155-169.

Shaye, S. D. (2008). *Transitioning a team to agile test methods*. Paper presented at the agile conference, 2008. IEEE Computer Society (pp. 470-477).

Sherehiy, B., Karwowski, W., & Layer, J. K. (2007). A review of enterprise agility: concepts, frameworks, and attributes. *International Journal of Industrial Ergonomics, 37*(5), 445-460.

Sherwood, J. J. (1988). Creating work cultures with competitive advantage. *Organisational Dynamics, 16*(3), 5-27.

Shinkle, C. M. (2009). *Applying the dreyfus model of skill acquisition to the adoption of kanban systems at software engineering professionals (SEP).* Paper presented at the agile conference, 2009. IEEE Computer Society (pp 470 - 477).

Shweder, R. (1980). *Fallible judgment in behavioural research*. San Francisco: Jossey-Bass.

Sison, R. (2009). *Investigating the effect of pair programming and software size on software quality and programmer productivity*. Paper presented at the 2009 software engineering conference. IEEE Computer Society (pp. 187-193).

Smits, H. (2006). 5 levels of agile planning: From enterprise product vision to team stand-up. Retrieved 10th April, 2007, from http://www.sao.corvallis.or.us/drupal/files/Five_levels_of_agile.pdf.

Smits, H. (2007). *The Impact of scaling on planning activities in an agile software development centre.* Paper presented at the 40th annual Hawaii international conference on system sciences, 2007. IEEE Computer Society (pp.274-280).

Sohal, A. S., Moss, S., & Ng, L. (2001). Comparing IT success in manufacturing and service industries. *International Journal of Operations & Operations Management, 21*(1/2), 30-45.

Sohal, S. (1999). Developing agile manufacturing in Australia. *International Journal of Agile Management Systems, 2*(2), 114-120.

Sol, H. G. (1983). A feature analysis of information systems design methodologies: methodological considerations. In T.W. Olle, H.G. Sol & C.J. Tully (Eds.), *information systems design methodologies: a feature analysis*. (pp. 1-7): North-Holland.

Sommerville, I. (2004). *Software engineering (7th Ed)*. Boston: Addison-Wesley.

Sommerville, I., & Sawyer, P. (1997). *Requirement engineering: a good practice guide*. Chichester: John Wiley & Sons.

Southwell, K. (2002). Agile process improvement. *TickIT International, 3-14*. http://www.tickit.org/ti3q02.pdf.

Spinellis, D. (2005). Version control systems. *Software, IEEE, 22*(5), 108-109.

Srinivasan, A., & Kaiser, K. M. (1987). Relationships between selected organisational factors and systems development. *Communication of the ACM, 30*(6), 556-562.

Srinivasan, J., & Lundqvist, K. (2009). *Using agile methods in software product development: a case study*. Paper presented at the information technology: new generations 2009 conference. IEEE Computer Society (pp. 1415-1420).

Stake, R. E. (1995). The art of case study research. Thousand Oaks: Sage Publications.

Stapleton, J. (1997). *Dynamic systems development method: the method in practice*. London: Addison-Wesley.

Stapleton, J. (2003). *DSDM business focused development* (2nd Ed.). London: Addison-Wesley.

Starbuck, V. (1985). Acting first and thinking later: theory versus reality in strategic change. In *organisational strategy and change* (pp. 336-372). San Francisco: Jossey-Bass).

Stear, E. B. (1997). Ten ways to gain management support for key projects. *ONLINE, May/June,* 103-104.

Steindl, C. (2005). *From agile software development to agile businesses.* Paper presented at the 31st EUROMICRO conference on software engineering and advanced applications, 2005. IEEE Computer Society (pp. 258 - 265).

Stevens, M. J., & Campion, M. A. (1994). The knowledge, skill, and ability requirements for teamwork: implications for human resource management. *Journal of Management, 20*(2), 503.

Stewart, G. L. (2006). A meta-analytic review of relationships between team design features and team performance. *Journal of Management, 32*(1), 29-55.

Sturman, A. (1999). Case study methods. In J. P. Keeves & G. Lakomski (Eds.), *issues in educational research*. Oxford: Pergamon.

Subramaniam, G. H., Klein, G., Jiang, J. J., & Chan, C.-L. (2009). Balancing four factors in system development projects. *Communications of the ACM, 52*(10), 118-121.

Sukkarieh, J. Z., & Kamal, J. (2009). Towards agile and test-driven development in NLP applications. In *proceedings of the workshop on software engineering, testing, and quality assurance for natural language processing* (pp. 42-44). Boulder, Colorado Association for Computational Linguistics.

Sutherland, J. (2001). Agile can scale: inventing and reinventing scrum in five companies. *CUTTER IT JOURNAL, 14*(12), 5-11.

Sutherland, J. (2005). *Future of scrum: parallel pipelining of sprints in complex projects*. Paper presented at the agile conference, 2005. IEEE Computer Society (pp. 90-99).

Takats, A., & Brewer, N. (2005). *Improving communication between customers and developers.* Paper presented at the agile conference, 2005. IEEE Computer Society (pp. 243 - 252).

Tan, R. P., & Edwards, S. (2008). *Evaluating automated unit testing in sulu.* Paper presented at the 2008 software testing, verification, and validation conference. IEEE Computer Society (pp. 62-71).

Tansley, C., & Newell, S. (2007). Project social capital, leadership and trust: a study of human resource information systems development. *Journal of Managerial Psychology, 22*(4), 350-368.

Taylor, H., & Woelfer, J. P. (2009). Critical skills for IT project management and how they are learned. http://doi.acm.org/10.1145/1542130.1542152. In *proceedings of the special interest group on management information system's 47th annual conference on Computer personnel research* (pp. 103-112). Limerick, Ireland ACM.

Taylor, P. S., Greer, D., Sage, P., Coleman, G., McDaid, K., & Keenan, F. (2006). Do agile GSD experience reports help the practitioner? http://doi.acm.org/10.1145/1138506.1138526. In *proceedings of the 2006 international workshop on global software development for the practitioner* (pp. 87-93). Shanghai, China ACM.

Teece, D. J., Rumelt, R., Dosi, G., & Winter, S. (1994). Understanding corporate coherence: Theory and evidence. *Journal of Economic Behaviour & Organisation, 23*(1), 1-30.

Tersine, R., & Wacker, J. (2000). Customer-aligned inventory strategies: agility maxims. *International Journal of Agile Management Systems, 2*(2), 114-120.

Tesluk, P. E., & Mathieu, J. E. (1999). Overcoming roadblocks to effectiveness: incorporating management of performance barriers into models of work group effectiveness. *Journal of Applied Psychology, 84*(2), 200-217.

Thomke, S. H. (1998). Simulation, learning and R&D performance: evidence from automotive development. *Research Policy, 27*(1), 55.

Tolvanen, P., Rossi, M., & liu, H. (1996). Method engineering, principles of method construction and support. In S. Brinkkemper, K. Lyytineen & R. Walke (Eds.), *method engineering: current research directions and implications for future research* (pp. 296-317). London: Chapman-Hall.

Tran, M. Q., & Biddle, R. (2008). Collaboration in serious game development: a case study. http://doi.acm.org/10.1145/1496984.1496993. *In proceedings of the 2008 conference on future play: research, play, share* (pp. 49-56). Toronto, Ontario, Canada ACM.

Ulrich, E., & Weber, W. (1996). Dimensions, criteria and evaluation of work group autonomy. In W. Michael. (Ed.), *handwork of work group psychology* (pp. 247-282). Chichester: John Wiley & Sons.

Ungar, J., & White, J. (2008). Agile user centered design: enter the design studio - a case study. http://doi.acm.org/10.1145/1358628.1358650. In *CHI '08 extended abstracts on human factors in computing systems* (pp. 2167-2178). Florence, Italy ACM.

Vaajakallio, K., & Mattelmaki, T. (2007). Collaborative design exploration: envisioning future practices with make tools. http://doi.acm.org/10.1145/1314161.1314182. In *proceedings of the 2007 conference on designing pleasurable products and interfaces* (pp. 223-238). Helsinki, Finland: ACM.

Vähäniitty, J., & Rautiainen, K. T. (2008). Towards a conceptual framework and tool support for linking long-term product and business planning with agile software development. http://doi.acm.org/10.1145/1370720.1370730. In *proceedings of the 1st international workshop on software development governance* (pp. 25-28). Leipzig, Germany ACM.

Vähänitty, J. (2005). A tentative framework for connecting long-term business and product planning with iterative \& incremental software product development. http://doi.acm.org/10.1145/1083091.1083097. In *proceedings of the seventh international workshop on economics-driven software engineering research* (pp. 1-4). St. Louis, Missouri ACM.

van Oyen, M. P., Gel, E. G. S., & Hopp, W. J. (2001). Performance opportunity for workforce agility in collaborative and noncollaborative work systems. *IIE Transactions, 33*(9), 761.

Vanhanen, J., & Lassenius, C. (2005). *Effects of pair programming at the development team level: an experiment.* Paper presented at the international symposium on empirical software engineering, 2005. IEEE Computer Society (pp.336-345).

Vanhanen, J., & Lassenius, C. (2007). *Perceived effects of pair programming in an industrial context*. Paper presented at the 2007 software engineering and advanced application conference. IEEE Computer Society (pp. 211-218).

Vanhanen, J., Jartti, J., & Kähkönen, T. (2003). Practical experiences of agility in the telecom industry.In *proceedings of the 4th international conference on extreme programming and agile processes in software engineering* (pp. 279-287). Geneva, Italy Springer-Verlag.

Vãzquez-Bustelo, D., Avella, L., & Fernã¡ndez, E. (2007). Agility drivers, enablers and outcomes. *International Journal of Operations & Production Management, 27*(12), 1303-1332.

Vernadat, B. (1999). Research agenda for agile manufacturing. *International Journal of Agile Management Systems, 1*(1), 37- 40.

Vokurka, R. J., & Fliedner, G. (1998). The journey toward agility. *Industrial Management & Data Systems, 98*(3/4), 165.

Wageman, R. (2001). How leaders foster self-managing team effectiveness: Design choices versus hands-on coaching. *Organisation Science, 12*, 559-577.

Wallace, L., & Kell, M. (2004). Software project risks and their effect on outcomes. *Communication of the ACM, 47*(4), 68-73.

Wallach, E. (1983). Individuals and organisations: The cultural match. Training & *Development Journal, 37*(2), 29-36.

Walsham, G. (1995). Interpretive case studies in IS research: Nature and method. *European Journal of Information Systems, 4*(2), 74-81.

Walter, M. (Ed.). (2006). *Social resarch methods: an Australian perspective*. Melbourne: Oxford University Press.

Walz, D. B., Elam, J. J., & Curtis, B. (1993). Inside a software design team: knowledge, acquisition, sharing, and integration. *Communication of ACM, 36*, 63-77.

Wang, X., Wu, Z., & Zhao, M. (2008). *The Relationship between developers and customers in agile methodology.* Paper presented at the international conference on computer science and information technology, 2008. IEEE Computer Society (pp. 566 - 572).

Wang, X., Wu, Z., & Zhao, M. (2008). *The relationship between developers and customers in agile methodology*. Paper presented at the 2008 computer science and information technology conference. IEEE Computer society (pp. 566-572).

Ward, P. T., Bicklord, D. J., & Leong, G. K. (1996). Configurations of manufacturing strategy, business strategy, environment and structure. *Journal of Management, 22*(4), 597.

Wegner, D. M. (1995). A computer network model of human transactive memory. *Sociology Cognition, 13*, 319-339.

Wegner, M. (1986). Transactive memory: A contemporary analysis of the group mind. In B. Mullen & R. Goethals (Eds.), *theories of group behaviour* (pp. 185-205). New York: Springer-Verlag.

Weick, E. (1987). Theorizing about organisational communication. In F. Jablin, L. Putnam, K. Roberts & L. Porter (Eds.), *handbook of organisational communication: an interdisciplinary perspective* (pp. 97-122). Newbury Park, CA: Sage Publications.

Wheelan, S. A. (2009). Group Size, Group Development, and Group Productivity. *Small Group Research, 40*(2), 247-262.

Wheelan, S., & McKeage, R. (1993). Development patterns in small and large groups. *Small Group Research, 29*, 371-393.

Whitener, E. M. (2001). Do "high commitment" human resource practices affect employee commitment? A cross-level analysis using hierarchical linear modeling. *Journal of Management, 27*(5), 515-535.

Whitworth, E. (2008). *Experience report: the social nature of agile teams.* Paper presented at the agile conference, 2008. IEEE Computer Society (pp. 429 - 435).

Whitworth, E., & Biddle, R. (2007). *The social nature of agile teams.* Paper presented at the agile conference, 2007. IEEE Computer Society (pp. 26 - 36).

Wiersma, W. (2000). Research methods in education. Boston: A Pearson Education Company.

Wilby, D. (2009). *Roadmap transformation: from obstacle to catalyst.* Paper presented at the agile conference, 2009. IEEE Computer Society (pp. 229 - 234).

Williams, L. (2003). The XP programmer: the few-minutes programmer. *Software, IEEE, 20*(3), 16-20.

Williams, L., & Kessler, R. (2002). *Pair programming illuminated*: Addison-Wesley.

Williams, L., Kudrjavets, G., & Nagappan, N. (2009). *On the effectiveness of unit test automation at Microsoft*. Paper presented at the 2009 software reliability engineering conference. IEEE Computer Society (pp. 81-89).

Williams, L., Maximilien, E. M., & Vouk, M. (2003). Test-driven development as a defect-reduction practice. In *proceedings of the 14th international symposium on software reliability engineering* (pp. 34): IEEE Computer Society.

Williams, L., McDowell, C., Nagappan, N., Fernald, J., & Werner, L. (2003). *Building pair programming knowledge through a family of experiments*. Paper presented at the international symposium on empirical software engineering, 2003. IEEE Computer Society (pp. 143-152).

Woodman, R. W. (1989). Organisational change and development: New arenas for inquiry and action. *Journal of Management, 15*(2), 205.

Xu, B. (2009). *Towards high quality software development with extreme programming methodology: practices from real software projects*. Paper presented at the 2009 management and service science conference. IEEE Computer Society (pp. 1-4).

Yao, A. C., & Carlson, J. G. H. (2003). Agility and mixed-model furniture production. *International Journal of Production Economics, 81/82*, 95.

Yenduri, S., Munagala, S., & Perkins, L. A. (2007). *Estimation practices efficiencies: a case study*. Paper presented at the 2007 information technology conference. IEEE Computer Society (pp. 185-189).

Yin, K. R. (1994). *Case study research: design and methods* (2nd ed. Vol. 41). London: Sage Publications.

Yin, R. K. (2002). *Case study research, design and methods* (3rd Ed.). Newbury Park: Sage Publications.

Youndt, M. A., Snell, S. A., Dean, J. W., Jr., & Lepak, D. P. (1996). Human resource management, manufacturing strategy, and firm performance. *The Academy of Management Journal, 39*(4), 836-866.

Yusuf, Y. Y., Sarhadi, M., & Gunasekaran, A. (1999). Agile manufacturing: the drivers, concepts and attributes. *International Journal of Production Economics, 62*(1-2), 33-43.

Zaccaro, S. J., & Klimoski, R. (2002). Special issue introduction: The interface of leadership and team processes. *Group & Organisation Management, 27*(1), 4.

Zhang, Z., & Sharifi, H. (2000). A methodology for achieving agility in manufacturing organisations. *International Journal of Operations & Production Management, 20*(4), 496-512.

Zhi-gen, H., Quan, Y., & Xi, Z. (2009). *Research on agile project management with scrum method*. Paper presented at the 2009 services science, management and engineering conference. IEEE Computer Society (pp. 26-29).

Zhu, H. (2006). *Separating designs from implementations: role-based software development*. Paper presented at the 2006 cognitive informatics conference. IEEE Computer Society (pp. 141-148).

Glossary of term

Adaptation factors are the elements which drive adaptation of the agile approach for software development.

Adaptive performance behaviour is the ability to deal with change and aptitude to apply learning from one task to another.

Agile organisations have the ability to create and respond swiftly to both anticipated and unanticipated changes in their business environments, regardless of being in manufacturing or software development industry.

Agility is the ability to sense and respond to business prospects in order to stay inventive and aggressive in an unstable and rapidly shifting business environment.

Behavioural adaptation is the change in the work behaviour of individuals.

Bespoke development relies on the customer to decide what functionalities should be developed.

Case study research is driven by prior development of theoretical propositions to apply, test or generate a relevant theory providing a more scientific approach for the research.

Change agents are the individuals driving agile adoption.

Cognitive adaptation is the individual ability to learn new things, devise problem-focused coping strategies, access information on change and solve problems associated with change.

Communally-based involves collective decision making.

Core competency is a central factor to keep the status as a market leader.

Covert factors enable a software product development environment a good feeling, thinking and functioning well in the quest of delivering software products consistently, regularly and when they are needed.

Cross-functional collaboration is having functional teams working together on projects.

Customer enrichment is providing high performing features for clients.

Customer satisfaction is providing product customisation and customer requested features.

Customer-driven innovation is when the customers identify and drive implementation of new features.

Daily plan shows the story or task an individual will implement for the day.

Dynamic adaptation involves the method-in-action to be created and improved on the fly using un-structured method fragments (new fragments which are untested and unproven) and also using structured method fragments.
Emotional adaptation is the individual ability to cope with change, not resist but allow the change to happen, and have a positive reaction to change and to the new opportunities made available.

Empowerment involves individual involvement in making product related decisions, and planning and managing the day-to-day operations at project level.

Ethnography research does not require a prior model, emphasising that no theoretical model can explain in advance the likely findings in a new contextual setting.

External cooperation is having strategic alliances, customer and supplier collaborations, and outsourcing.

External validity is defined as the extent to which the research findings are applicable to the general population.

Extra-role support involves carrying out cross-functional tasks.

Fluid role performs a wide variety of tasks and motivates joint efforts between the functional units and individuals.

Generalists are flexible individuals having multiple development and business skills in agile software development teams.

Generative behaviour is the individual ability to learn and educate on tasks and responsibilities of other roles while working with them.

Generative rule involves teams creating a rule to make a task or practice compulsory.

Global development involves having outsourcing, subcontracting, in-house development and partnership with others.

Grounded theory requires no literature review prior to research, no taping of interviews and no discussion of theory before being written.

Human factor specialists are individuals who focus on usability practice or usability testing of features and products during development.

In-role support involves carrying out functional tasks.

Internal cooperation is having teamwork and cross-functional collaboration.

Internal validity of the case study approach ensures that the results are seen as accurate and are interpreted with confidence, helping to minimise other plausible and competing explanations of a result.

Intrapersonal competency enables engineers to listen and respond to situations in their work team environment.

Iteration or sprint plan provides a subset of estimated and analysed features that are to be implemented.

Macro orientation involves all the teams adopting agile development.

Macro-manage involves team managers adapting to become outward or customer focussed and no longer direct, control or monitor the day-to-day tasks and work performance.

Market-driven environment have competitors and frequent changes in requirements, customer preferences and technologies.

Mechanistic organisations develop standardised products, emphasise development efficiency and compete based on product cost, quality and delivery.

Method adaptation is the change or modification of a software development method.

Method engineering approach for adaptation has a positivist approach, requiring a systematic means to adapt for having an appropriate method-in-action prior to its application.

Method fragment is a guideline, goal, value, principle, belief, practice, fundamental concept, tool, technique, activity, task, hint, and tip including the product to be delivered.

Methodology-in-action is the adapted or tailored version of the adopted method that is made relevant to the current software development project.

Micro orientation uses one development team only for agile adoption.

Mutual adaptation requires collective decision making, involving all the stakeholders.

On-site customer is a business individual providing support at development level.

Organic organisations are constantly changing and adapting organisations.

Original formalised methodology is the adopted method(s), which the team thinks is relevant for their software development work for the time being.

Overt factors are the obvious reasons as why the methods are used or adopted for development work.

Pair programming is an agile practice in which two engineers work together at one work station.

Performance differential involves developing and making available outstanding features and products.

Political motivation involves having support (executive) for agile adoption.

Positivist study is defined as having a scientific approach for investigating a research problem.

Proactive behaviour is the individual ability to search for new opportunities to contribute to the team's success.

Product backlogs have prioritised requirements ready for implementation in short development cycles.

Product roadmap plan shows features which will be in different releases of a software product.

Profile of development environment consists of organisational factors influencing product development**.**

Qualitative methods enables to study selected issues in-depth and to collect data which are detailed descriptions of situations, events, people, interactions, observed behaviours, and quotations from people about their experiences, attitudes, beliefs, and thoughts.

Quantitative methods involve the use of standardised measures so that the varying perspectives and experiences from a great number of people can fit into a limited number of predetermined response categories.

Reactive behaviour allows learning new things and enabling interpersonal, cultural and physical adaptations.

Release plan describes which features will be delivered in upcoming release.

Research paradigm is a basic set of beliefs that guides how a qualitative or quantitative method and practices relating to data collection, analysis, and interpretation are used for conducting a study.

Resilience behaviour is the ability to cope with change.

Responsive behaviour is the individual and organisational ability to swiftly learn, adapt and respond to deliver the market needs.

Revolutionary approach involves having organisational-wide change for agile development all at once.

Short development cycles are several small phases for implementation in a project, leading to a release.

Social capital is the connections established through social events.

Static adaptation has a repository of tested method fragments from which appropriate or relevant method fragments are selected based on rules to build a project specific method-in-action.

Strategic commitment is having executive support for implementing adaptation decisions made at the development level.

Strategic support is having executive support for agile adoption.

Tacit knowledge is unwritten or unspoken product development knowledge held by individuals, which is gained through their development experiences and observations.

Team composition is defined as individuals who are hired to be part of the development function.

Team effort is sharing of tasks or having a collective effort on tasks.

Team ownership involves the entire team collectively responsible for the delivery of committed features.

Test driven development (TDD) is as an evolutionary approach to development, involving implementing a test before implementing and refactoring the code.

The socio-organisational approach for adaptation has an interpretative approach, requiring the method-in-action to be adapted as needed during its application.

Tolerant behaviour allows coping with stress and uncertainty.

Transactive memory is the team memory system for collective encoding, storing, and retrieving of development knowledge.

Triangulation is the use of different and multiple sources, researchers and theories to provide confirming evidence.

Validity with qualitative studies is defined as how truthful the research results are or if research has truly measured what it intended to measure.

Vision plans have information on target users or customers, a few key features or high-level requirements, product benefits, reasons for clients to buy it, how it differs from other competing products, and how it will bring competitive advantage.

Work team is organised based on fixed development capacity.

Workforce agility is having highly skilled, competent and adaptable individuals capable of dealing special situations.