UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering, Science, and Mathematics
School of Electronics and Computer Science

Software Quality and Governance in Agile Software Development

by

Noura Abbas

A thesis submitted for the degree of Doctor of Philosophy

December 2009

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE, AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Noura Abbas

Looking at software engineering from a historical perspective, we can see how software development methodologies have evolved over the past 50 years. Using the right software development methodology with the right settings has always been a challenge. Therefore, there has always been a need for empirical evidence about what worked well and what did not, and what factors affect the different variables of the development process. Probably the most noticeable change to software development methodology in the last 15 years has been the introduction of the word “agile”. As any area matures, there is a need to understand its components and relations, as well as the need of empirical evidence about how well agile methods work in real life settings.

In this thesis, we empirically investigate the impact of agile methods on different aspects of quality including product quality, process quality and stakeholders’ satisfaction as well as the different factors that affect these aspects. Quantitative and qualitative research methods were used for this research, including semi-structured interviews and surveys. Quality was studied in two projects that used agile software development. The empirical study showed that both projects were successful with multiple releases, and with improved product quality and stakeholders’ satisfaction. The data analysis produced a list of 13 refined grounded hypotheses out of which 5 were supported throughout the research. One project was studied in-depth by collecting quantitative data about the process used via a newly designed iteration monitor. The iteration monitor was used by the team over three iterations and it helped identify issues and trends within the team in order to improve the process in the following iterations. Data about other organisations collected via surveys was used to generalise the obtained results. A variety of statistical analysis techniques were applied and these suggested that when agile methods have a good impact on quality they also has a good impact on productivity and satisfaction, also when agile methods had good impact on the previous aspects they reduced cost. More importantly, the analysis clustered 58 agile practices into 15 factors including incremental and iterative development, agile quality assurance, and communication. These factors can be used as a guide for agile process improvement. The previous results raised questions about agile project governance, and to answer these questions the agile projects governance survey was conducted. This survey collected 129 responses, and its statistically significant results suggested that: retrospectives are more effective when applied properly as they had more impact when the whole team participated and comments were recorded, that organisation size has a negative relationship with success, and that good practices are related together as when a team does one aspect well, they do all aspects well. Finally, the research results supported the hypotheses: agile software development can produce good quality software, achieve stakeholders’ satisfaction, motivate teams, assures quick and effective response to stakeholder’s requests, and it goes in stages, matures, and improves over time.

Table of Contents

List of Figures

List of Tables

Declaration of Authorship

I, *Noura Abbas* declare that the thesis entitled:

Software Quality and Governance in Agile Software Development

and the work presented in it are my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been published as:

Abbas, N., Gravell, A. and Wills, G. (2010)Using Factor Analysis to generate Clusters of Agile Practices. In Agile 2010 Conference, *Orlando, Florida.*

Abbas, N., Gravell, A. and Wills, G. (2010). The Impact of Organisation, Project and Governance Variables on Software Quality and Project Success. In Agile 2010 Conference, *Orlando, Florida.*

Abbas, N., Gravell, A. and Wills, G. (2008) An Empirical Comparison of Two Agile Projects in the Same Organisation. In Research in Progress Workshop at the Agile 2008 Conference, *Toronto, Canada.*

Abbas, N., Gravell, A. and Wills, G. (2008) Historical Roots of Agile Methods: Where did "Agile Thinking" Come from? In: Agile processes and eXtreme programming in Software Engineering*, 10-14 June 2008, Limerick, Ireland. pp.94-103.*

Abbas, N., Gravell, A. M. and Wills, G. B. (2007) Agile Software Assurance. In: PhD Symposium in the 8th International Conference on Agile Processes in Software Engineering and eXtreme Programming (XP 2007)*, June 18-22, Como, Italy. pp. 165-166.*

Abbas, N. (2007) Agile Software Assurance: An Empirical Study (Poster) In the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 20-21 September, Madrid, Spain. p. 499.*

Signed: *Noura Abbas*

Date: *18/12/2009*

Acknowledgments

First and foremost I would like to express my sincere gratitude and thanks to my supervisor Dr. Andrew M Gravell, not only for the continuous feedback, guidance, and reviewing every chapter in this thesis, but also for his unlimited support, patient, encouragement, and for challenging me to always deliver the best throughout the whole period of my study, since day one to the day of the final submission.

Also, I would like to thank my advisor, Dr Gary B Wills, for his support, guidance and suggestions.

My thanks also go to Dr. Paul W Garratt for his feedback, suggestions and sharing his industrial experience.

I would like to express my thanks to all my interviewees for taking the time to participate in my research. In particular, I would like to thank Mr. Simon Rachman from IBM for his time, discussions and the valuable feedback, and his team for participating in the interviews and filling the iteration monitor. In addition, my thanks go to Mr. Colin Thorne for his input and his team from IBM for participating in the interviews.

I would like to thank all the people in the agile community for the discussions and sharing their experiences. In particular, I would like to thank everybody who took the time to fill the agile projects governance survey.

I would like to acknowledge Mr. Scott Ambler for making the raw data of his surveys available for reuse, and for providing comments on the agile project governance survey design.

I would like to express my thanks to my friends and colleagues in the School of Electronics and Computer Science and the University of Southampton who made my stay in the University such a great and enjoyable experience an for being there when I needed them.

Finally, I would like to thank my family: my parents for their endless love, support and for always believing in me, my sisters and their families, and my brother for all their support and encouragement. I would also like to thank my husband Mansour for his love, understanding, and for putting a smile on my face even in difficult times.

Thank you all, for helping me achieving my dream.

To My Parents

Definitions and Abbreviations

ACM: Association of Computer Machinery

ASD: Adaptive Software Development

Agilists: People who support agile methods and its philosophy

BVD: Business Value Delivered

CMM: Capability Maturity Model

Consumability: a description of customers' end-to-end experience with technology solutions. This concept was introduced by IBM.

DSDM: Dynamics Systems Development Methods

EAP: Early Access Program

FDD: Feature-Driven Development

IBM®: Registered trademark of International Business Machines Corporation in the United States, other countries, or both.

IBM FDS: IBM Federal Systems Divisions

IDD: Incremental and Iterative Development

IEEE: Institute of Electrical and Electronics Engineers

ISO: International Organisation for Standardization

IT: Information Technology

LOC: Lines of Code

NASA: National Aeronautics and Space Administration

OR/I: Obstacles Removed per Iteration

PAR: Problem Analysis Report

PMR: Problem Management Reports

QA: Quality Assurance

RAD: Rapid Application Development

RIPP: Rapid Iterative Production Prototyping

RTF: Running Tested Features

RUP: Rational Unified Process

SDC: Software Development Corporation

SWEBOK: Software Engineering Body of Knowledge

SQA: Software Quality Assurance

TDD: Test Driven Development

TQM: Total Quality Management

TTOR: Time to Obstacle Removal

XP: Extreme Programming

Introduction

1.1 Development of the Research

Agile software development is gaining interest from both academia and industry. Researchers expect to see increasing use of agile methods for projects such as financial services, E-commerce, and air traffic control (Boehm 2002). Although many papers, articles, and books have been published about agile methods, little work has discussed their impact on software quality and stakeholders' satisfaction. More importantly, evidence about this impact was needed. This was the starting point of this research and it was used to form the initial research questions: what is the impact of agile software development on software quality and stakeholders' satisfaction? and what factors affect quality, stakeholders' satisfaction and project success when using agile software development approaches?.

In order to answer the research questions we had to fully understand what is agile software development, and how did the idea of agile software development evolve? In addition, we had to understand what is quality, what is quality assurance, and what is quality for agile software development?

The literature review showed that the available quality models were based and designed for traditional approaches to software development, mainly the sequential or the waterfall model. Therefore, we argue that there is no systematic way to integrate quality assurance within an agile method. Also, this review gave us a deeper understanding of the existed research in the area, and the used research methods. Furthermore, we were able to identify the gaps in the research area. In a systematic review about the empirical studies of agile software development (Dyba et al. 2008), the authors concluded that that there "*is a need for more and better empirical studies of agile development within a common research agenda*". Although the paper showed that a good number of empirical studies about agile development has been done, still these are mainly focused on one agile method, namely XP.

The literature review refined our research questions to go beyond investigating the impact of agile software development on the different aspects of quality, and to develop a model or framework or checklist or recommendations for agile teams wishing to enhance their quality.

As software development is a human-based activity, the best way to study this activity and to get valid applicable results is to apply empirical approaches within real world settings. Therefore, this approach was chosen for our investigation.

It was therefore decided to find a case study. Two project managers from IBM agreed to be interviewed for the purpose of the research. This led to more interviews with their team members. The empirical study was a great opportunity to explore quality in two agile projects. Moreover, the analysis of the interviews produced 31 hypotheses about the impact of agile software development on the different aspects of software quality. This list was refined and reduced by asking practitioners during the Agile 2008 conference to indicate which would be most interesting to confirm or reject. This allowed 13 of the hypotheses to be selected as the focus for the dissertation. In addition, the empirical study enabled us develop and trial a new technique for agile quality: The iteration monitor. This monitor was designed to first test the produced hypothesis, and to collect data about the iteration to understand how things are changing over the iterations and more importantly, the team members' opinions about the process, the quality of the product, and the support provided to the different stakeholders. The iteration monitor collected data from the team over three iterations. Analysing the collected data produced interesting results, which supported four of the hypotheses. In addition, analysing the data yielded in statistically tested relationships between the different aspects of the iteration. Although studying IBM experience was very valuable, it was limited to one organisation. Therefore, we decided to explore the experience of other organisations using a survey to collect as much data as possible about other organisations' experience. It was moreover decided to explore existing surveys to avoid repeating questions which had previously been asked. Agile adoption surveys that were conducted since 2006 (Ambler 2006) were available with their raw data. Therefore, we decided to review these surveys and investigate how we can further analyse their data. The survey results supported two of the hypotheses. Applying different statistical analyses on these data produced more interesting significant results about the impact of different aspects such as organisation size, productivity and cost on software quality, success rate, and stakeholder satisfaction. More importantly, it generated 15 factors or clusters of agile practices; which can be used as a guide for agile process improvement. The results of applying the iteration monitor and analysing the agile adoption surveys led us to think about agile projects governance; also we needed to know how organisations are governing agile projects. In addition, we conducted our survey, which covered not only software quality, but also agile projects governance. The survey collected 129 responses, and its results illustrated the state of the art in agile projects governance, the use of retrospective and reflection meetings, and metrics within agile software development. The findings from this survey supported two of the hypothesis and it provided statistically tested evidence about how quality and success rate are affected by organisation, project, retrospective and metrics variables.

1.2 Research Contributions

I. An empirical study of two agile projects was carried out, focussing on quality in agile projects. It was notable that both projects were on-time, through multiple releases, achieving high level of customer satisfaction and low defect rates. A list of grounded hypotheses was generated and refined. The 13 remaining hypotheses were organized in three main groups: the impact of agile software methods on software quality, stakeholder's satisfaction, and process quality. Six of these hypotheses in particular were supported throughout the research:

   H5: Agile software development assures quick and effective response to stakeholders' requests (Chp.6 – P.113)

   H6: Agile software development can produce good quality software (Chp.6 – P.113) – (Chp.7 – P.115-118) – (Chp.8 – P.140)

   H1: Agile software development can achieve customer satisfaction (Chp.7 – P.115-118)

   H21: Team members are happy and motivated when using agile software development (Chp.6 – P.113)

   H11: The quality of the code increases as the number of iterations increases (Chp.6 – P.113)

   H26: The adoption of agile methods goes in stages and it improves over time, releases, and projects (Chp.8 – P.147)

II. An iteration monitor was designed to identify issues and trends within a team in order to improve the process and understand changes between iterations. One of the IBM teams used the iteration monitor over three iterations.

III. Three existing agile adoption surveys were re-coded and reanalysed. New and statistically significant results were obtained which suggest that:

   a. When agile methods had good impact on one aspect, they also had good impact on others. Good impact on quality, customer satisfaction, and productivity were positively correlated, so that as productivity improves, quality and satisfaction improve, and cost is reduced.

   b. 58 techniques used by agile teams were clustered into 15 factors which can be used as a guide for agile projects process improvement.

    c. Agile quality assurance practices and iterative and incremental development have a positive, statistically significant, relationship with project success.

IV. A new survey to study agile projects governance was conducted. The results presented the state of art of agile project governance including the use of retrospectives and metrics in an agile software development environment. The statistical analysis of this survey suggested that:

    a. Organisation size has a negative, statistically significant relationship with project success (also supported in contribution III)
    b. Retrospectives are more effective when applied properly as they had more impact on the project when the whole team participated, everybody had their say, and comments were recorded.
    c. Project success has a positive and statistically significant relationship with a number of agile metrics such as team velocity, business value delivered, running tested features, as well as more traditional metrics such as number of test cases, and defect count after testing.
    d. Good practices are related together: (good quality, high productivity, high customer satisfaction, and low cost) and (performing retrospective, team participation, comments recording, collecting metrics). In other words when a team or an organisation does one aspect well, they do all aspects well.

V. Our review of the literature on traditional and agile methods generated new insights and understanding into the nature of agile methods and their roots.

    a. The reasons behind the development and introduction of agile methods are identified, as a reaction to traditional methods, as a result of people's experience, and in particular focusing on reusing ideas and techniques from the history of software development.
    b. A new definition of agile methods is given whereby they are defined as adaptive, iterative and incremental, with a people oriented process.

As with all empirical research, there are a number of threats to the validity of the previous conclusions. They are based on interviews and questionnaires, so the data collected is subjective and based on the subjects’ perception of quality rather than direct measures. As adaptivity and people-orientation are key components of agile methods, it is not possible to come up with definitive recommendations: instead, each project and team needs to select and refine those techniques which work well for them.

1.3 The Thesis Structure

The Thesis is organised as follow:

Chapter 2 Historical Overview: This chapter discusses the historical development of software methodologies, the waterfall, the V-Model, the spiral model, and the Rational Unified Process. Particularly, it investigates the roots of the waterfall, and alternative approaches such as iterative, incremental, and evolutionary approaches. Furthermore, it provides evidence of what did work well in practice and what did not. Finally, it explains our understanding of these different approaches.

Chapter 3 Agile Methods Review: This chapter reviews the existing agile methods, and it focuses on what was behind the agile movement, how these methods are different from traditional approaches. In addition, the chapter reviews the existing agile Surveys, and provides our definition of agile methods.

Chapter 4 Software Quality: This chapter reviews the available definitions of quality, software quality and software quality assurance as well as the available software quality models, standards, and metrics. In addition, the chapter investigates quality in agile world: including the conducted research around the topic, agile metrics, stakeholders' satisfaction, and the impact of agile on software quality. Finally, the chapter discusses the gaps in literature and proposes the initial research goals.

Chapter 5 Empirical Study: This chapter presents two case studies to explore quality in two agile projects within IBM using semi-structured interviews. The chapter discusses the following: why such empirical studies are needed, review of related work and studies, the nature of the empirical research and the methodology we used to collect and analyse the data, our results for each project, and a comparison between the two projects. The chapter presents what is bad and what is good about agile methods and a comparison between traditional approaches and agile methods based on the interviewees' perception. Finally, the chapter presents the hypothesis generated based on the two case study results.

Chapter 6 The Iteration Monitor: This chapter introduces the iteration monitor. Which is a web based that can be used a governance visibility tool. This monitor was needed for two reasons; first to support the generated hypotheses and create further ones. Second, to understand how things are changing over the iterations and more importantly what the team members' opinions about the process, the quality of the product, and the support provided to the different stakeholders. This chapter presents the iteration monitor design, and how it was used over three iterations by IBM team. The collected data are analysed and resulted discussed, and a comparison between the three iterations is presented.

Chapter 7 Applying Correlations and Factor Analysis on Existing Agile Surveys: This chapter reviews three existed agile surveys and it presents our new analysis of their results. The correlations between quality, productivity, satisfaction, and cost are studied and finally the factor analysis is used to cluster 58 agile techniques into 15 factors which can be used as a guide for agile process improvement. The relationship between the produced 15 factors is studied and presented.

Chapter 8 Agile Project Governance Surveys: This chapter presents the agile projects governance survey. The main purpose of this survey is to investigate agile projects governance by collecting data about how people are monitoring the progress of projects developed using agile method, practices or principles. The survey is particularly interested in projects using agile retrospectives, reflection meetings, and metrics. This chapter presents this survey, its design, and analysis, and it presents the results that describe the agile projects governance state of the art and the relationship between different aspects of agile governance.

Chapter 9 Conclusions and Future Work: This chapter concludes the thesis by presenting the conclusions of all previous chapters. In addition, it suggests different ways to carry out future work.

Chapter 2

Historical Overview

2.1 Introduction

In the last 25 years, many new methodologies have been introduced to the software engineering field. The iterative and incremental software development approaches form the foundation of most of these new methodologies, and of modern software engineering in general. However, many sources still recommend the single pass software development lifecycle, which is known as the “Waterfall”.

This chapter will discuss the historical development of software methodologies, the waterfall (Boehm 1981), the V-Model (Johansson et al. 1999), the spiral model (Boehm 1988), and the Rational Unified Process (Kruchten 2001). Particularly, it will investigate the roots of the waterfall model, and alternative approaches such as iterative, incremental, and evolutionary approaches. Furthermore, it will provide evidence of what did work well in practice and what did not. Finally, it will present our understanding of incremental and iterative development.

2.2 The Waterfall Model: Historical View

It is difficult to define “waterfall” development precisely as this word has been used in different ways as shown below.

As early as 1956, Benington proposed the first version of the “waterfall”. This nine phases’ model was used in MIT’s Lincoln Laboratory to produce programs for the SAGE air-defence system (Benington 1956; Benington 1983; Benington 1987). As this model was a sequence of phases, later, it was referred to as the “sequential waterfall” (Brooks 1995). Winston Royce introduced the next version of the “waterfall” in his article “Managing the Development of Large Software Systems”. In this article, Royce argued that the implementation of the “sequential waterfall” is “*risky and invites failure*” due to its limitation in dealing with requirements change.

In 1976, Barry Boehm wrote a paper to provide a definition of the term “Software Engineering” and to survey the state of the art in the field. In this paper Boehm used the term “software lifecycle” to refer to an improved version of Royce’s model (Boehm 1976). Actually at this point neither Royce nor Boehm had mentioned the word “waterfall”. Arguably, Boehm was the first who presented the “waterfall model” in his book Software Engineering Economics in 1981 as a second version of the software lifecycle. According to Boehm, the major features in this form of the model were that each phase was culminated by a verification and validation activity in order to reduce the problems of this phase.

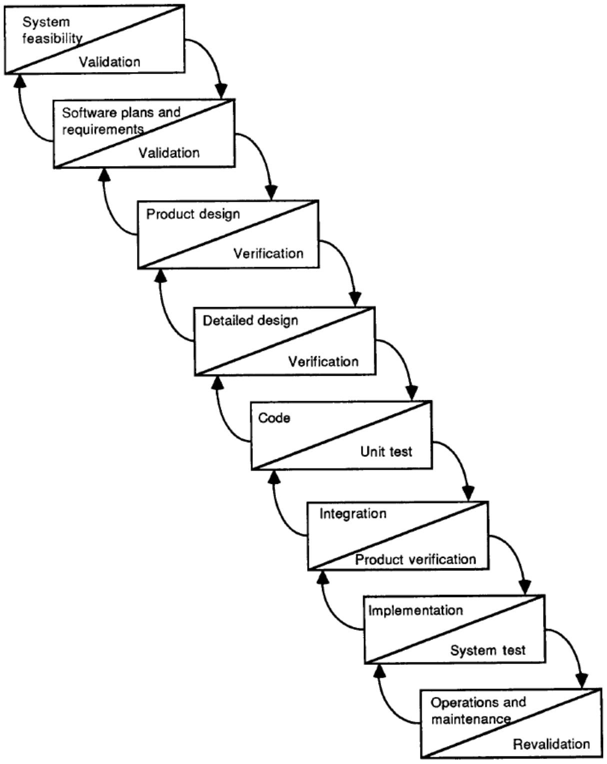


Figure 2-1 The waterfall model of the life-cycle (Boehm 1981)(See Appendix A for other versions)

Verification means, “*are we building the product right*?” where validation means, “*are we building the right product*”. In addition, Boehm recommended performing as much as possible iterations of earlier phases in the next phase (Boehm 1981). In his book, Boehm introduced two refinements of the idealized waterfall model: the incremental development and the “advancemenship” which takes two main forms in a software project: anticipatory and software scaffolding.

I tried to find out who first used the term “waterfall”. Some resources used the term “Software life-cycle” to refer to that model (Boehm 1981; Gladden 1982; McCracken et al. 1982; Pressman 1987). During a personal conversation with Barry Boehm (Boehm 2007) he stated that the term “waterfall” was in use when he joint the TRW. Interestingly most books cited Royce’s paper and not Boehm’s book when they mentioned the term “waterfall”.

2.2.1 Did the “Waterfall” work well?

Although both Royce and Boehm recommended additional features in order to adopt the basic software lifecycle or the waterfall, most of the later sources presented the basic model as one of the software development methods. Actually, the waterfall did not invariably work well. Here we will provide evidence from both practice and literature.

Evidence from Practice

- The USA Department of Defence DoD standard, DoD-STD-2167, which was released in the 1980s, was based on the waterfall model and a document-driven approach (Larman 2004). In 1988, the DoD abandoned the 2167 and replaced it with an iterative, incremental development friendly standard, the DOD-STD-2167A. However, the original single-step waterfall diagrams remained in the updated 2167A, and this was because “*the military logistics people would not agree with my assessment that they would continue to foster the waterfall mindset*” (according to Firesmith who was involved in improving the 2167A standard) (Larman 2004). Therefore, the need for a new standard began to surface. In 1994, the MIL-STD-498 was completed and approved. This standard removed the “*waterfall bias*”, recommended developing software in incremental builds, added more flexibility to the development process, and decreased the emphasis on documentation (Newberry 1995; Radatz et al. 1995). In addition, the MIL-STD-498 proposed an alternative to formal reviews and audits which used to cause “*tremendous expenditure of time and energy*”. Instead, MIL-STD-498 recommended informal discussions and ongoing communication between the acquirer and the developer (Radatz et al. 1995).
- The attempted USA-ATC (Air Traffic Control) project started in 1983 as a massive big-bang waterfall project. In 1994, the government cancelled the project after spending $2.4 billion USD. The project was restarted in the late 1990s with an incremental approach (Government Accounting Office 1998).
- A similar project, the Canadian ATC System (CAATC) started in 1989 as waterfall project following the DoD-STD-2167A. In 1992, and after spending several hundred millions dollars, the Canadian government restarted the project using an iterative approach (Toth et al. 1993).

Evidence from Literature

- Royce considered the single pass waterfall a risky model. Furthermore, he was in favour of a throwaway prototype which he called "*a pilot model*" (Royce 1970). In addition, Winston's son Walker Royce stated that his father was always a proponent of iterative, incremental, evolutionary development (Larman et al. 2003)

- In his famous book the Mythical Man-Mouth, Brooks criticised both versions of the waterfall. In the first edition of the book, he criticised the sequential version of the model when he recommended to "*plan to throw it away*" (Brooks 1979). This is similar to Royce's advice "*build it twice*". However, in the anniversary edition of the book in 1995 Brooks stated "*now I perceive to be wrong*" and his advice became "*do not build one to throw away- the waterfall model is wrong!*" (Brooks 1995). So, in this book he is neither in favour of the sequential waterfall, nor of the one with the iterative flavour. He mentioned that the problem with the waterfall is that it "*assumes that all the mistakes will be in the realization*" and they can be easily repaired during component and system testing. In his opinion, the "*plan to throw away*" advice fails to solve the problem.

- McCracken and Jackson published a paper in 1982 where they presented three groups of criticism to the lifecycle concept, which assumed that the systems development consists of 10 sequential steps. Their main points were that the life cycle concept:

  - can not be applied to all system development;
  - eliminates the need for communication, ignores the need for the end-user heavy involvement in all phases of application development, and does not take into account that the user and his/her needs change during the process;
  - "*rigidifies thinking*", and it is very poor in response to change.

  They stated "*The lifecycle concept is simply unsuited to the needs of the 1980's in developing systems*" (McCracken et al. 1982).

- In his paper, "*Stop the life cycle, I want to get off*", Gladden considered the concept of a lifecycle may be "*harmful*" to the software development profession. Actually, he mentioned the word "*waterfalling*" to describe the sequence of tasks in the lifecycle. Similar to McCracken and Jackson, he argued that the main problem of the lifecycle approach has been its limitation in dealing with changing and new requirements (Gladden 1982).

- In 1988, Boehm presented the spiral model in his paper "A Spiral Model of Software Development and Enhancement" in order to overcome the problems in the current models. In this paper, Boehm stated that the waterfall model, even with all it revisions and refinements, has some "*fundamental difficulties and these have led to the formulation of alternative process models*" (Boehm 1988). Boehm argued that the primary problem of the waterfall model has been its emphasis on fully documented requirements and design in early stages of the project (Boehm 1988).

The reason why the waterfall model is still recommended is that it might worked with some cases were the project is well understood, when quality requirements dominate cost and schedule requirements, or when the staff are technically weak or inexperienced (McConnell 1996). During an interview with a consultant and project manager who worked in industry for more than 10 years, the interviewee stated that he was involved with three waterfall projects that were successful and the customer was happy. The first project had a team of 5 people and lasted for 9 months with 10% overrun. The second project had a team of 80 and lasted for two years and a half with 25% overrun. The third team had a team of 100 and lasted for two and a half years with 25% overrun. The first project had naive customer and stable requirements, where the last two had complicated unpredictable requirements (Garratt 2007).

2.3 The V-Model

The V-Model is an extension of Boehm's waterfall where each phase was culminated by a verification and validation activity. The oldest source we could find about the V-Model or the V-chart was Boehm's paper "Guidelines for Verifying and Validating Software Requirements and Design Specifications". He stated, "*This figure [Figure 2-2] represents a "V-chart" which shows the context of verification and validation activities throughout the software lifecycle [personal communication from J.B. Munson, System Development Corporation, 1977]*" (Boehm 1979). Obviously, the V-Model has almost the sequence of phases as the waterfall model. However, instead of going down in a liner way, for each requirement and design phase, a testing phase will take place to ensure the system validation and verification (Johansson et al. 1999).

The criticism about the V-Model is that it divides system development with firm boundaries between them; this is the same problem with the waterfall. More importantly, it discourages people from carrying testing information across the different phases as some tests are done earlier than required and others are executed to be done later (Marick 1999; Liversidge 2005).

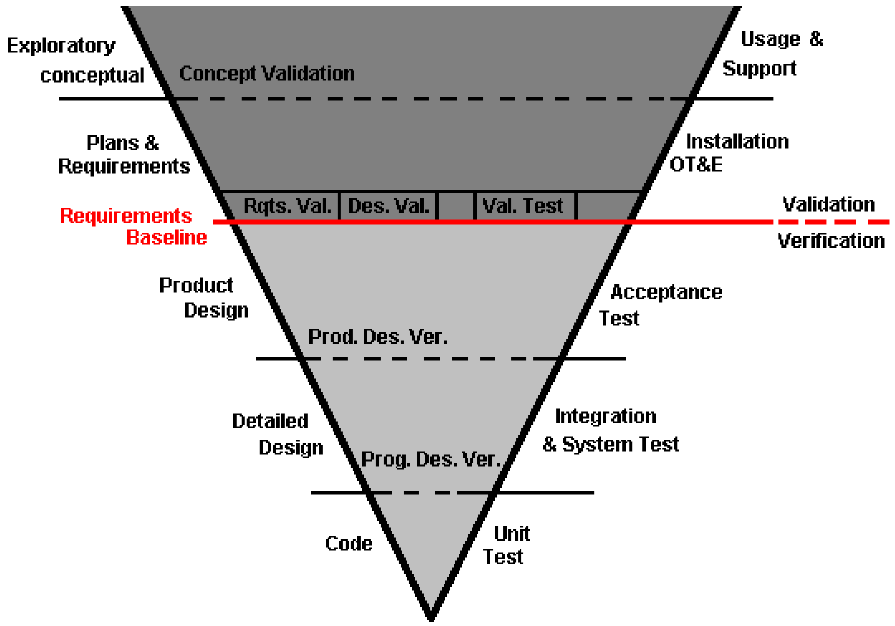


Figure 2-2 The V-Model (Boehm 1979)

2.4 The Spiral Model

The spiral model is a risk-driven approach originally proposed by Boehm 1988 (see figure 2-3). This model was presented as a "*candidate for improving the software model situation*" (Boehm 1988). In addition, it provided the potential for rapid development of incremental versions of the software. In this model, the software is developed in a series of incremental versions. The development starts on a small scale in the middle of the spine, according to Boehm; each cycle of the spiral begins with the identification of:

1. The objectives of the current portion of the product
2. The alternative methods to implement this portion
3. The constraints on the implementation of these alternatives

In other words, we start with exploring and determining the risks of the current portion of the product. In the next step, we evaluate the risks and try to find a strategy to resolve them. This strategy could be prototyping or simulation. After the evaluation, the remaining risks will determine the next step. McConnell suggested that it is possible to combine the spiral model with other life cycles. For example, if we reduce the risks to an acceptable level, we can apply a non-risk approach for the rest of the project (McConnell

1996). In his paper, Boehm stated that the most important advantage of the spiral model is that its range of options accommodates the good features in the existing process models, while the risk assessment avoids their difficulties. However, it relies on the ability of software developers to identify the risks (Boehm 1988).

The spiral model suggests a customer communication where the customer will address all the requirements. Unfortunately, in reality this is not the case, therefore, the spiral model was extended to include a negotiation process. The result was the Win-Win spiral model (Boehm et al. 1998). In this model a set of negotiation activities will be defined at the beginning of each pass around the spiral (Pressman 2001).

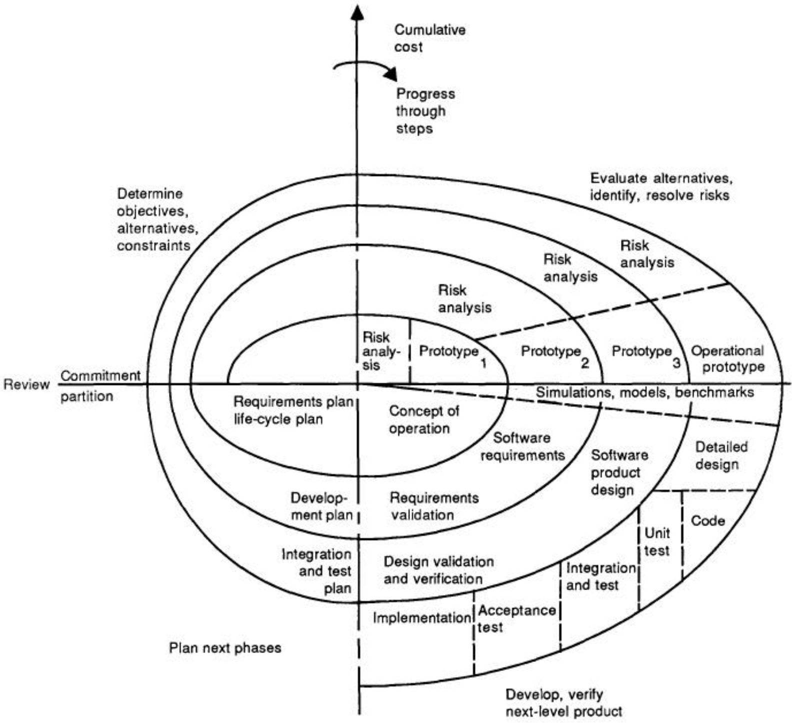


Figure 2-3 The spiral model (Boehm 1988)

2.5 The Rational Unified Process

The Rational Unified Process (RUP), which was released in 1998, was a result of merging the rational approach, which was developed at Rational Software in the 1980s, and 1990s, with the Objectory Process, which was developed by Ivar Jacobson. Actually, the RUP is more than a lifecycle, it is a process framework and it can be adapted to accommodate the organisations' needs (Kruchten 2001). In the RUP architecture (Figure 2-4), we can recognize two dimensions: a horizontal dimension to represent time and to show the dynamic aspect of the process which describes phases and iterations; and a vertical dimension to represent the workflows of the process, and to show the static aspect of the process which describes activities and roles.

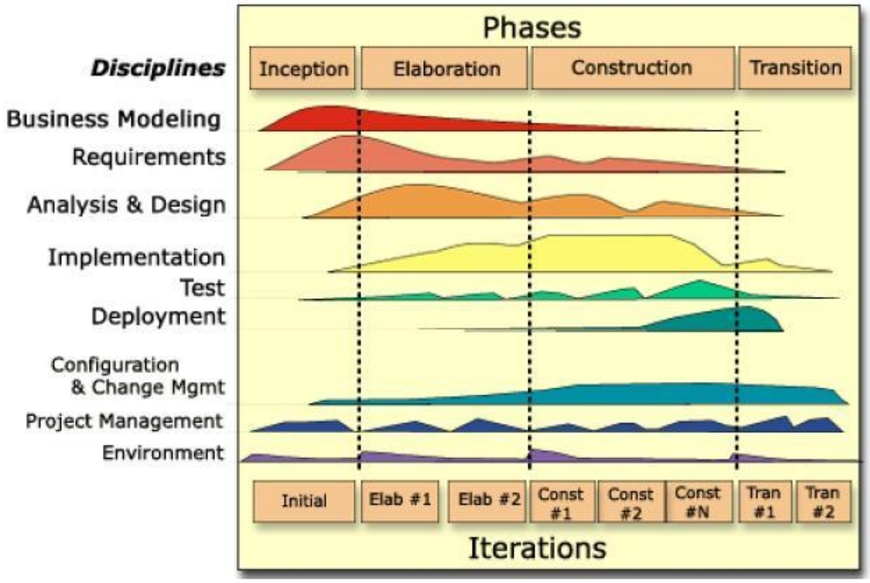


Figure 2-4 The Rational Unified Process architecture (Rational 1998)

Each phase in the RUP can be enacted in an iterative way, and the result will develop incrementally. The whole set of phases can be enacted in an iterative way as well, and at the end of each development loop, a new release of the system will result.

Recently, Ivar Jacobson introduced the Essential Unified Process or "EssUP". According to him, it is "*a fresh new start*" and it integrates the successful process from three process campus: the Unified Process, the agile methods and the Process Maturity (Jacobson 2006). Dave Thomas, the managing director of Object Mentor and founding director of the Agile Alliance (www.agilealliance.org) looked at the EssUp and concluded that it is "*much simpler and more flexible than previous expressions of UP*" and it is a lightweight UP process (Thomas 2006).

2.6 Iterative, Incremental and Evolutionary

The waterfall model was not the only available approach to software development. People were using other approaches successfully in the 70s and the 80s. Iterative and incremental concepts formed the basics for these approaches. Larman and Basili found early roots for iterative and incremental development (IID) since the 1950s in NASA and IBM FSD (Larman et al. 2003). According to them, NASA's 1961-63 Project Mercury was run with "*short half-day iterations*". In addition, the Extreme Programming practice of test-first development was applied as tests were planned and written and then the code was written to pass the tests. Furthermore, the project used continuous integration as each mini-iteration required integration of all code. Gerald Weinberg, who wrote the Project Mercury description, was an IID proponent. According to (Larman et al. 2003) he stated:

"We were doing incremental development as early as 1957, in Los Angeles, under the direction of Bernie Dimsdale at IBM's Service Bureau Corporation.[...] All of us, as far as I remember, thought water-falling of a huge project was rather stupid or at least ignorant of the realities"

Furthermore in 1970, Winston Royce who criticised the sequential model, recommended "*five additional features that must be added to the basic approach to eliminate most of the development risks*" (Royce 1970). These steps had the favour of iterative development. In step two, he recommended an early development pilot model for a 30-month project. This model might be scheduled for 10 months. In addition, in step five, he stated that the customer should be formally involved and he/she has to commit himself/herself at earlier points before the final delivery. Table 2-1 illustrates eight projects in the 70s and 80s, where IID was the reason behind the success of large, critical projects.

Date/Reference	Company	Project Description	IID Favour and Key Ideas
1972 (O'Neill 1983)	IBM FSD	-Trident Project -The command and control system for the first USA trident submarine-high-visibility -life-critical system -Over one million lines of code	-Up-front specification effort with feedback-driven evolution -Four iterations (six months each)
1972 (Williams 1975)	TRW	- Army site defence software project for ballistic missile defence - Large, real time reliable software	- Royce waterfall with the 5 features - Incremental approach to detailed design, code and test - Used a combination of top down concepts and incremental approach
Mid 1970 (Mills 1980)	IBM FSD	- USA Navy Helicopter-ship system - Four years 200 person-year effort - Developing over three millions and integrating over seven millions words of program and data	- 45 Incremental delivery over four years - Two months increments

Date/Reference	Company	Project Description	IID Favour and Key Ideas
1977-1980 (Madden et al. 1984)	IBM FSD	- NASA Space Shuttle software	- 17 Increments over 31 months - Feedback driven refinement of specification
1977-1980 (Wong 1984)	SDC	- Modern air defence system - 25 month schedule	- Incremental build, implementation and test
1984-1988 (Firesmith 1987)	Magnavox Electronic Systems	- Large field artillery command and control system for the USA Army - 1.3 million line Ada project	- Five iterations - Two times the productivity, three times the quality comparing to other Magnavox projects
1987 (Royce 1998)	TRW	- Command Centre-Processing and Display System Replacement (CCPDS-R) - Four years project	- Six iteration, six months each

Table 2-1 Projects with iterative and incremental flavour

From table 2-1, we can see that all the projects are major, government, life-critical systems, involving large numbers of people. This invalidates the claim that only the single-pass waterfall is suitable for large-critical systems (Pressman 2001; Sommerville 2004). In addition, most of the projects used a combination of top down concepts and incremental development. The projects used different iterations' lengths, which were longer than the range recommended by today's iterative methods. Interestingly, only one project mentioned that they used Royce's approach as it was presented in his paper. Furthermore, the word incremental was used more frequently than the word iterative, in the sense of "adding onto" rather than improving the previous implementation in the next iteration. In all references, the authors mentioned the word "approach" rather than "method".

2.6.1 Definitions

Although people were using iterative and incremental approaches successfully since the 1950s, we could not find a common or clear understanding of the terms "evolutionally", "incremental", and "iterative", neither in industry, nor in literature. Therefore, it is worth understanding these approaches. In this section, the available meanings will be reviewed and evaluated in order to give a clear understanding of those terms.

Evolutionary

Tom Gilb introduced the word evolution in 1976 in his book "Software Metrics". He stated "*evolution is a designed characteristic of a system development which involves gradual stepwise change*" and "*evolution is a process characteristic*" (Gilb 1976). In a later paper, Gilb distinguished between incremental development (break up the system into small deliveries), throwaway prototyping (which does not have to produce a real result), and evolutionary where he proposed a similar idea to iterative development, based on system refinement. Gilb recommended to use "*prototyping, successive refinement, and incremental (evolutionary) delivery in complementary combination*" (Gilb 1981). This is almost what most people mean by iterative and incremental development these days.

Incremental

According to The Compact Oxford English Dictionary (Increment 2009), the word increment (noun) is: "*an increase or addition, especially one of a series on a fixed scale*". Derivatives: incremental (adjective), incrementally (adverb). Noticeably, the word increment was well-known and used in software development since 1957 (Larman et al. 2003). McConnell defined incremental development as breaking the project into a series of small subprojects (McConnell 1996). Pressman presented incremental model as an evolutionary process model, and he considered it iterative in nature (so the two words were equal for him). In Pressman's model, the first increment is a core product in which basic requirements are addressed but many supplementary features (some known and some unknown) remain undelivered. He stated, "*It delivers software in small but usable pieces called 'increments'. In general, each increment will be built on those that have already been delivered*" (Pressman 2001).

Alistair Cockburn described incremental development as a "*staging and scheduling strategy*" where different parts of the system are designed, tested and then integrated as they are complete (Cockburn 1997).

In incremental development, a plan of several feature deliveries is defined, so feedback is not driving the delivery plan. However, it is possible to re-plan after each increment as the conditions may be different after each one, but re-planning is not strictly necessary (Cockburn 2007).

Iterative

According to Oxford English Dictionary (Iterate 2009), the word iterate (verb) means to "*make repeated use of a mathematical or computational procedure, applying it each time to the result of the previous application*". Derivatives: iteration (noun), iterative (adjective). The first paper that presented the iterative approach was in 1975 by Basili and Turner. In this paper the authors recommended the "iterative enhancement" technique, a top-down, stepwise refinement approach to software development (Basili et al. 1975). They suggested starting with a "*simple initial implementation of a subset of the problem and iteratively enhance existing versions until the full system is implemented*". Obviously, this is the current understanding of iterative development. Furthermore, the authors stated that at each step of the process, design modification and system extensions can be made (Basili et al. 1975). This is the description of iterative and incremental development as we will see in the following sections. Cockburn explained that iterative development is a "*rework scheduling strategy*", where the best first guess of a piece of design or code is produced, evaluated, and then improved (Cockburn et al. 2005; Cockburn 2007). The evaluation can be done by the suitable stakeholders according to the overall process. In iterative development, there is no fixed plan of future deliveries; the plan for the next iteration is created based on emerging information.

2.6.2 Discussion

We can see that iterative and evolutionary development approaches are almost the same as both emphasise developing a piece of the system, reviewing it for improvements and feedback either by the customer or by the team. Thus after each iteration the customer will see an improved system and not always new features. In this case, we prioritise quality over features. In incremental development, after each increment, the customer will see the system growing. In this case we prioritise features over quality.

2.6.3 Iterative and Incremental Development

From the previous discussion, we can say that incremental means "add-onto" where iterative means "repeating". Development can be iterative without being incremental by applying the activities repeatedly without growing the system. It also can be incremental without being iterative by breaking the system into a number of parts without a repetitive application of the development activities (Spence et al. 2005).

We argue that the most effective strategy is to be iterative and incremental by developing part of the system, testing it, and improving this section while developing a new part. After a number of iterations, a release will be delivered to the customer. In addition, after each iteration there will be a delivery which could be a piece of code delivered to test groups, a demo, or a presentation to get feedback from the customer.

Arguably, each increment includes a number of iterations, and the result of the increment will be the release. However, in order to stop the confusion with the real meaning of iterative and incremental development, we suggest the following: when we say iterative and incremental development we mean that we develop, refine and add-onto, and we will have a delivery after each iteration. This delivery could be internal or external. (See Figure 2-5)

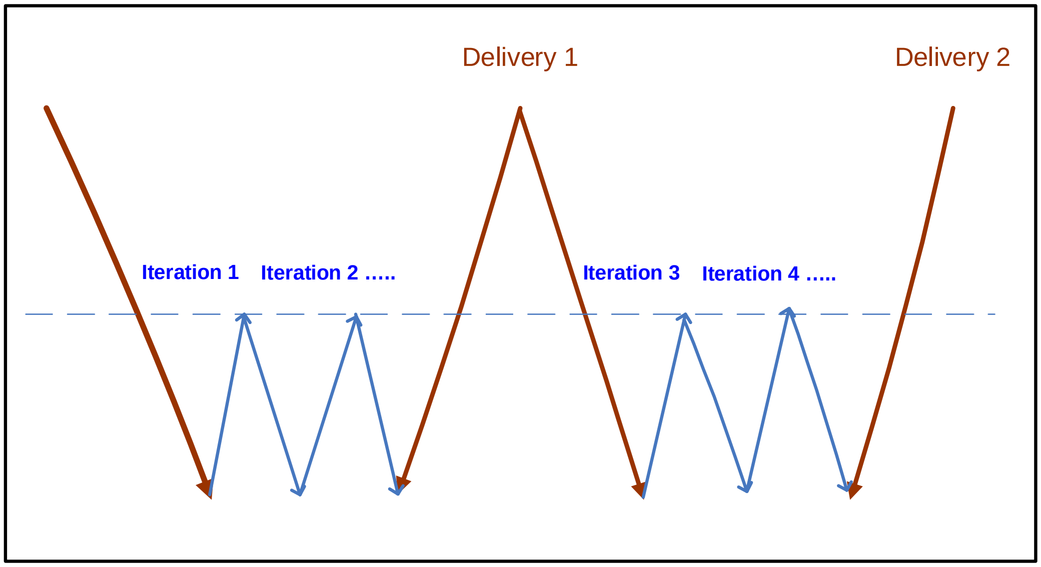


Figure 2-5 One way to apply iterative and incremental development (adapted from (Cockburn 1997))

In a personal contact, Alistair Cockburn stated that "*these days most people just say iterative development to mean both and use the term iteration to mean any period of time in which they increment or iterate*" (Cockburn 2007).

2.7 Summary

This chapter has given a historical overview of the most recognizable software methodologies. It has started with the root of the software life-cycle or the waterfall model. Furthermore, it has provided evidence from literature and industry that the waterfall model did not work well unless with stable and predictable requirements. Then the chapter showed that the iterative and the incremental approaches were used successfully since the 70s. So it was worth understanding these approaches which form the foundation of agile methods. Finally our understanding of incremental and iterative development was explained as developing a part of the system, testing it, and improving this part while developing a new one.

Chapter 3

Agile Methods Review

3.1 Introduction

The appearance of agile methods has been the most noticeable change to software process thinking in the last fifteen years (Fowler 2005). Many reviews, studies, and surveys have been conducted on agile methods (Abrahamsson et al. 2002; Highsmith 2002; Paulk 2002; Cohen et al. 2004; Larman 2004; Williams 2004; Levine 2005). Therefore, this chapter will not only review the existing agile methods, but will cover some aspects that were not been covered in previous studies. The chapter will focus on what was behind the agile movement, how these methods are different from traditional approaches. In addition, the chapter will review the existing agile surveys, and will provide our definition of agile methods.

3.2 Background

The term “agile” refers to a philosophy of software development (Fowler 2005). This term was agreed on during a big gathering when seventeen of the proponents of the “lightweight” approaches to software development came together in a workshop in early 2001 (Highsmith et al. 2001). Prior to this workshop, a number of different groups have independently developed methods and practices to respond to the changes they were experiencing in software development (Cohen et al. 2004; Fowler 2005).

According to Martin Fowler, one of the participants, the workshop was organised by Jim Highsmith and Bob Martin. The idea was to invite people who shared the same ideas about software development to get together in order to build better understanding of each others approaches and to agree on an umbrella name for these various approaches. The result of this gathering was the Agile Manifesto (Highsmith et al. 2001). In OOPSLA (ACM Conference on Object-Oriented Programming, Systems, Languages, and Application) 2001, where most of the seventeen people met again, there was a suggestion that they should continue the agile movement, but they did not want to claim the leadership of the agile community. The next step was the formation of the Agile Alliance, which is a non-profit web-centre which encourages and researches agile software development (Fowler 2005).

3.2.1 Agile Manifesto

The Agile Manifesto gathered representatives from Extreme Programming (XP), Scrum, Dynamics Systems Development Methods (DSDM), Adaptive Software Development (ASD), Crystal Methods, Feature-Driven Development (FDD), Pragmatic Programming, and others who saw the need for an alternative to documentation driven, heavyweight software development processes.

The manifesto reads as follows (Highsmith et al. 2001):

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

Individuals and interactions	*over*	*Processes and tool*
Working software	*over*	*Comprehensive documentation*
Customer collaboration	*over*	*Contract negotiation*
Responding to change	*over*	*Following a plan*

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

The previous four values have been further defined by twelve principles (see Appendix B). According to Fowler, the principles section was started at the workshop, however, it was developed in a wiki afterwards (Fowler 2005).

The participants strengthened the idea that "*the agile movement is not anti-methodology*". They embraced modelling, but not to write diagrams that will never be used. They embraced documentation, but not hundreds of pages that will rarely be read or even maintained. They embraced planning, but they understood that a changing environment needs planning that is limited in scope.

We have to keep in mind that the manifesto is "symbolic" (Highsmith et al. 2001) and its "*values are relative statements, not absolute*" (Boehm et al. 2003). In other words, it represents an attitude, a philosophy, or a goal. Alistair Cockburn, one of the participants in the manifesto, stated, "*We're still not there. There is really only would-be-agile development*" (Cockburn 2002b).

Although the manifesto provides some specific ideas, and a focusing statement that helps concentrate these ideas, most of the participants have their own approaches, ideas, and communities. The point we want to make here is that all these people agreed on the same overall approach for software development, despite the fact that they all have their own approaches. This does not happen frequently in the software engineering world. UML (Unified Modelling Language) is probably the only available example (Booch et al. 1996). When UML was created by the three amigos; Grady Booch, Jim Rumbaugh, and Ivar Jacobson, it was the result of the unification of their three modelling languages Booch, OMT (Object Modelling Technique) and OOSE (Object-Oriented Software Engineering). Alistair Cockburn was surprised that this group of "agilists" agreed on something substantive. He stated, "*I was surprised that the others appeared equality delighted by the final phrasing [of the manifesto]. So we did agree on something substantive*" (Highsmith et al. 2001).

3.3 What does it Mean to be Agile

The understanding of the word agile varies in practice. In addition, it is difficult to define agile methods, as it is an umbrella for well-defined methods, which also vary in practise. This section will show how this word was explained in literature by its proponents, as well as by other researchers.

According to Cambridge Advanced Learner's Dictionary (agile 2009), the word "agile" has two meanings:

- (Physically quick) able to move your body quickly and easily
- (Mentally quick) able to think quickly and clearly

Although this definition addresses the response to change feature in agile methods, it missed a lot of their real meaning.

Some researchers tend to define agile as a philosophy. Alistair Cockburn's definition is "*agile implies being effective and manoeuvrable. An agile process is both light and sufficient. The lightness is a mean of staying manoeuvrable. The sufficiency is a matter of staying in the game*" (Cockburn 2002a). Barry Boehm described agile methods as "*an outgrowth of rapid prototyping and rapid development experience as well as the resurgence of a philosophy that programming is a craft rather than an industrial process*" (Boehm et al. 2003).

Another way to describe agile methods is by stating the basic practices various methods share. Craig Larman stated, "*It is not possible to exactly define agile methods, as specific practices vary. However short timeboxed iterations with adaptive, evolutionary refinements of plans and goals is a basic practice various methods share*" (Larman 2004). Boehm gave more practice-oriented definition, "*In general, agile methods are very lightweight processes that employ short iteration cycles; actively involve users to establish, prioritize, and verify requirements; and rely on tacit knowledge within a team as opposed to documentation*" (Boehm et al. 2003). In an eWorkshop on agile methods organised by the Centre of Experimental Software Engineering (CeBASE), the participants defined agile methods as iterative, incremental, self-organising, and emergent. In addition, they stated that all agile methods follow the four values and twelve principles of the Agile Manifesto (Cohen et al. 2004). Boehm provided similar definition as he considered that a truly agile method must include all of the previous attributes (Boehm et al. 2003).

3.4 What was Behind Agile Methods

Some interesting questions are:

- What was behind agile methods?
- Where agile methods were introduced?
- What are the origins of agile thinking?

We will answer these questions through three different points: reaction to traditional methods and business change, reusing ideas from history, and people's experience.

3.4.1 Reaction to Traditional Approaches and Business Change

The previous chapter discussed in detail the problems with the waterfall model. In addition, it showed that iterative approaches were in use a long time ago. Although the V-Model, the spiral model and then the RUP tried to solve the waterfall problems, they are still heavyweight, document and plan driven approaches. Fowler referred to these approaches as engineering methodologies which may work perfectly for building a bridge but not for building a software, as building a software is a different kind of activity and it needs a different process (Fowler 2005). Another reason behind agile methods was the increasing change in the business environment. According to Highsmith and Cockburn, agile methods were proposed from a "*perspective that mirror today's turbulent business and technology change*" (Cockburn et al.

2001a). The traditional approaches could not cope with this change as they assumed that it is possible to anticipate a complete set of the requirements early in the project lifecycle. In reality, most changes in requirements and technology occur within a project's life span.

3.4.2 Reusing Ideas from History

Many agile ideas are hardly new. Furthermore, many people believed that this is the most successful way of building software. However, these ideas have not been treated seriously, in addition, presenting them as a whole is new (Larman 2004; Fowler 2005).

The previous chapter presented examples of successful use of iterative and incremental approaches since the 70s. In addition, it provided evidence that the waterfall did not work well. Actually, people who criticised the waterfall suggested alternative approaches. In his paper "Stop the Life-Cycle, I Want to get off", Gladden suggested a new view of the development process and he called it the "Non-Cyclical Hollywood Model". According to Gladden, this model satisfies three propositions (Gladden 1982):

- *System objectives are more important than system requirements*: this meets the agile idea of having a general understanding of the system rather than having detailed requirements which will change over the project
- *A physical object conveys more information than a written specification*: this is noted as the agile manifesto values: *Working software over Comprehensive documentation*
- *System objectives plus physical demonstrations will result in a successful product:* by successful project he meant that a product that performs the function intended and satisfies the customer's need.

Gladden believed that most users do not have a clear idea about their needs, and he raised the problem of missing and changing requirements.

Another suggestion was from McCracken and Jackson in their paper "Life Cycle Concept Considered Harmful". They suggested two scenarios of system development processes (McCracken et al. 1982):

- Prototyping: McCracken and Jackson suggested building a prototype extremely early in the development process as a response to the early statements of the user. A series of prototypes or a series of modifications to the first prototype will gradually lead to the final product. This is exactly how development in agile is meant to be, short iterations in each we improve the system. In addition, they recommended a close relation with the user: "*development proceeds step-by-step with the user, as insight into the user's own environment and needs is accumulated*".

- The second suggestion was a process of system development done by the end-user and an analyst in this sequence: implement, design, specify, redesign, re-implement. Again, starting with implementing the system is the idea of modern iterative development. In addition, the authors suggested providing the user with an implementing tool and one version of a system which is a similar idea of the CASE tools, which were used in Rapid Application Development (RAD) the early version on DSDM. RAD and DSDM will be discussed later in this chapter.

Agile ideas appeared in old development processes as well. In 1985, Tom Gilb wrote "Evolutionary Delivery versus the 'Waterfall model'". In this paper, Gilb introduced the EVO method as an alternative of the waterfall which he considered as "*unrealistic and dangerous to the primary objectives of any software project*".

Gilb based EVO on three simple principles (Gilb 1985):

1. *Deliver something to the real end-user*
2. *Measure the added-value to the user in all critical dimensions*
3. *Adjust both design and objectives based on observed realities*

In addition, Gilb introduced his "personal list" of eight critical concepts that explained his method. When he discussed the early frequent iteration, he emphasised the concept of selecting the "*potential steps with the highest user-value to development–cost ratio for earliest implementation*" (Gilb 1985). Another important concept in EVO method is "*Complete analysis, design and test in each step*" where he stated that the waterfall is one of the great time wasters with too many unknowns, too much dynamic change and systems complexity. Gilb stressed being user oriented:

"With evolutionary delivery the situation is changed. The developer is specifically charged with listening to the user reactions early and often. The user can play a direct role in the development process"

and being results oriented, not process oriented, *"Evolutionary delivery forces the developers to get outside of the building process for a moment, frequently and early – and find out whether their ship is navigating towards that port of call many cycles of delivery away"*

Obviously, many of Gilb's concepts meet agile principles. Not only the frequent delivery and the short iterations, but also he stressed the user role in the development process. In addition, he recommended an adaptive process and he gave the developers the power to change the direction of the process.

After Gilb's EVO, in 1988, the DuPont Company presented a methodology called Rapid Iterative Production Prototyping (RIPP). The main goal was to build working prototypes that could be presented to customers regularly to ensure that the finished product is what they wanted. The company guaranteed "Software in 90 days… or your money back" (Ambrosio 1988).

James Martin expanded this methodology into a large formalized one which became the Rapid Application Development (RAD). The RAD lifecycle has four phases: requirements planning, user design, construction and implementation (Martin 1991). What distinguishes RAD from traditional lifecycles is that in RAD construction phase we do the detailed design and code generation of one transaction after another. Each transaction can be shown to the end users to make adjustments. In addition, the "timebox" applies to the construction phase. The team will be given a fixed timebox within which the system must be constructed. The timebox inputs are the functions and the design framework of the system. The output is the system, which will be evaluated to decide whether to put it in production, or not. Within the timebox, "*continuous iterative development is done*" in order to produce a working system by the end of the timebox (Martin 1991). Martin recommended 60 days length for the timebox, with a 1-5 persons team. The term "timebox" was created by Scott Shultz and was first used in DuPont. Shultz stated that the timebox methodology was successful as all the applications were complete in less time than it would have taken just to write the specification for a COBOL or FORTRAN application (Martin 1991).

It is obvious that RAD has almost all agile ideas. Actually, it formed the base for DSDM, one of agile methods (Stapleton 1997). RAD recommended quick delivery, iterative development, small team of highly trained developers who work together at high speed, and user's involvement at every stage. Clearly, these ideas are the heart of agile methods. However, the term "timebox" is used differently in agile. In RAD, it is the whole construction phase and it consists of many iterations, where in agile the timebox means a fixed iteration. In a fixed iteration, if the requests of the iteration can not be met within the timebox, the scope will be reduced (Stapleton 1997; Larman 2004).

3.4.3 People's Experience

As it has been already mentioned, the manifesto gathered people who needed an alternative to traditional approaches. Importantly, most people involved in the manifesto had experience in software development. Furthermore, they had their own well-defined methods such as Extreme Programming (XP), Crystal and Scrum.

Ken Schwaber, one of the developers of Scrum, described his experience in the early 1990s when he was running a software company. He mentioned that their requirements were always changing and their customer's methodology did not help, instead it slowed them down. In order to solve the problem, he presented his methodology to process theory experts at the DuPont Experimental station in 1995. He stated that they were amazed that his company was using an inappropriate process. In addition, they said that systems development had so much complexity and unpredictability that it had to be managed by an "empirical" process control model (Schwaber et al. 2001). Ken's company and other organisations asked another question, why empirical development approaches deliver productivity while defined processes such as Capability Maturity Model (CMM) do not. They passed the question to scientists at DuPont Chemical's Advanced Research facility, and the answer was that CMM is treated as a well-understood defined process while it is not, and it is performed without control and therefore it gives unpredictable results (Schwaber 1996).

Kent Beck, founder of XP, also had a story. In April 1996, he was hired to help with Chrysler, a payroll system. The project was in a state where two months away from production, the development team were not "*computing the right answers yet*". With the CIO of Chrysler, they decided to start from scratch with a smaller team. With Ron Jeffries, who became the first XP coach, and with the help of Martin Flower with analysis and testing, the first XP project took off. They worked on the base of three weeks iteration, where they implemented stories chosen by the domain expert. In April 1997, the system was live, and it was resalable, cheap and easy to maintain and extend. Beck stated "*it was a technical and business success*" (Beck et al. 2004).

Another story is from Alistair Cockburn, one of the Agile Manifesto authors. In 1991, IBM Consulting Group asked him to write a methodology for object-technology projects (Cockburn 2005). He decided to interview the project team. He found out that their stories were different from what was mentioned in methodologies books. He found that "*close communication, morale, and access to end users separated in stark contrast the successful projects [he] visited from the failing ones*". Cockburn tried these ideas on a $15 million, fixed-price, and fixed-scope project of forty-five people. He was the lead consultant of the project and he wrote up the lessons learned from the project interviews, and from the project itself. Using these ideas, Cockburn built his agile method Crystal (discussed in more details later). Interestingly, unlike most of other authors of the manifesto he stated that he came to agile principles "*through the need for efficiency, not the need to handle rapidly changing requirements*".

3.5 How Agile is Different from another Approaches

Iterative development, customer involvement, frequent delivery and other principles and values of the Agile Manifesto support the argument that agile is different from traditional heavyweight or plan-driven approaches. This section focuses on more detailed differences between agile methods and traditional approaches.

3.5.1 Process

Traditional approaches aim to make software development predictable. Therefore, they try to impose a defined process with a lot of planning. Agilists see that a defined process is suitable for predictable manufacturing domains. They believe that developing software is an unpredictable activity. Therefore, they try to focus on an adaptive (or empirical) process (Schwaber 1996; Fowler 2005). An adaptive process is a process that can give control over unpredictability. To survive in unpredictable world we need iterative and incremental development in order to control the unpredictable process.

3.5.2 People and Communication

One aim of traditional methodologies is to develop a process that fits everybody, in other words a process where people are replicable parts. However, agilists believe that people have a first–order effect on software development (Cockburn 1999). Furthermore, an agile process requires talented and skilled people and moulds the process to specific people and teams not the other way around (Cockburn et al. 2001b; Fowler 2005). In addition, agile methods promote working in an open plan area rather than separate offices, in order to increase communication between the team members (Larman 2004). In agile development, all team members are involved in a variety of different activities during the development process such as design, coding, and testing, where traditional methodologies try to give people separate tasks according to the lifecycle phases.

3.5.3 Measure of Success

The traditional criteria for a successful project are being the one on time and within budget. However from an agile point of view, the measurement is the business value and the question we should ask is "*did the customer get software that's more valuable to them than the cost put in*" (Fowler 2005).

3.5.4 Requirements

Traditional approaches prefer formal, stable, and complete requirements in advance, where in agile methods the requirements are adjustable, informal stories. These stories will be changed and prioritized through iterations, and this will be done collaboratively by the customers and developers (Boehm et al. 2003). Agilists accept that business people do not understand what they need from the software in the beginning. Furthermore, they understand that "*business environment changes at a dramatically increasing pace*" (Cockburn et al. 2001a; Fowler 2005). Martin Flower stated "*even if the customers fix their requirements the business world is not going to stop for them*" (Fowler 2005).

3.5.5 Customer Involvement

Traditional approaches use documentation, contracts and review boards to communicate with the customer, where agile methods strongly emphasise having a dedicated customer who is involved in the development process (Boehm et al. 2003). Having a customer onsite is crucial in agile development (Deursen 2001).

3.5.6 Management

Generally, traditional approaches follow Theory X in management (McGregor 2006), which assumes that the average human beings prefer to be directed, wish to avoid responsibility, lazy and will avoid work if they can. Therefore, most people must be controlled, directed and threatened with punishment. Agile methods follow Theory Y (McGregor 2006), which assumes that average human beings are not lazy, but able to learn under proper conditions. Moreover, work can be a source of satisfaction. In addition, external control and the threat of punishment are not the only means to force people to work. Instead, self-direction and self-control are more important and can be directed to achieve the organisation's objectives (McGregor 2006). We can see that this philosophy agrees with putting people first in agile development. In agile development, all team members will have a go in leading the team. Martin Fowler raised an interesting point in his paper "The New Methodology", where he stated that the developers must be able to make all technical decisions. This is important because after a few years, the technical knowledge becomes obsolete, and the ex-developers can fill management roles and trust new developers (Fowler 2005).

3.5.7 Quality

Traditional approaches to quality assurance are paper based, heavyweight, measured by conformance to plan (Fowler 2005). SW-CMM (Capability Maturity Model for Software) defines quality assurance as specification and process compliance, where Boehm stated that in agile methods quality is customer satisfaction (Boehm et al. 2003). Scott Ambler stated "*quality is an inherent aspect of true agile software development*" and this is as a result of practices such as iterative development, test driven development, and refactoring (Ambler 2005). However, many questions about agile quality assurance are still without an answer (McBreen 2003; Mnkandla et al. 2006) therefore we will investigate this topic in coming chapters.

3.6 Existing Agile Methods

Under the umbrella of "Agile" term sit more specific approaches such as Extreme Programming (XP), Scrum, Crystal Methods, Adaptive Software Development (ASD), Dynamic Systems Development Method (DSDM), Feature-Driven Development (FDD), and Lean Development. As mentioned earlier in this chapter introduction, many studies have been conducted on agile methods. In addition, many books and articles analysed and compared agile methods in details (Abrahamsson et al. 2002; Boehm et al. 2003; Cohen et al. 2004). Therefore, table 3-1 provides a quick illustration to address the most important features of the existing agile methods, as well as providing references for further reading. Then, three agile methods, XP, Scrum and Crystal will be discussed with more details. The lifecycle associated with each method can be found in Appendix C.

Extreme Programming (XP) by Kent Beck (Beck et al. 2004)

Practices	Values	Principles	Process Phases
• Sit together • Whole team • Information workspace • Energised work • Pair programming • Stories • Weekly cycle • Quarterly cycle • Slack • Ten-minute build • Continuous integration • Test-first programming • Incremental design	• Communication • Simplicity • Feedback • Courage • Respect	• Humanity • Economics • Mutual benefit • Self-Similarity • Improvement • Diversity • Reflection • Flow • Opportunity • Redundancy • Failure • Quality • Baby steps • Accept responsibility	• Exploration • Planning • Iterations • Release • Productionising • Maintenance • Death

Beck, K. and C. Andres (2004). Extreme Programming Explained (2nd Edition), Addison-Wesley Professional.

Scrum by Ken Schwaber and Jeff Sutherland (Schwaber et al. 2001)

Practices		Values	Process Phases
• 30 day iteration (Sprint) • Sprint planning meeting • Self organising teams	• Daily scrum meetings • Sprint review • The scrum master • Product backlog	• Commitment • Focus • Openness • Respect • Courage	• Pre-game • Development • Post-game

Schwaber, K. and M. Beedle (2001). Agile Software Development with Scrum, Prentice Hall PTR.

Crystal Clear by Alistair Cockburn (Cockburn 2005)

Properties	Strategies	Techniques
• Frequent delivery • Reflection improvement • Osmotic communication • Personal safety • Focus • Easy access to expert users • Technical environment with Automated tests, configuration management and frequent integration	• Exploratory 360° • Early victory • Walking skeleton • Incremental re-architecture • Information radiators	• Methodology shaping • Reflection workshop • Blitz planning • Delphi estimation using expertise ranking • Daily stand-up meetings • Essential interaction design • Process miniature • Side-by-side programming • Burn charts

Cockburn, A. (2002). Agile software development, Addison-Wesley Longman Publishing Co., Inc.

Cockburn, A. (2005). Crystal Clear A human -Powered methodology for Small Teams, Addison-Wesley.

Adaptive Software Development (ASD) by Jim Highsmith (Highsmith 2000)

Lifecycle Phases	Lifecycle Characteristics	Management
• Speculate (initiation and planning) • Collaborate (concurrent feature development) • Learn (quality review)	• Mission focused Timeboxed • Risk driven iterative • Change tolerant • Feature based	Adaptive (leadership-Collaboration management model) based on: • Leadership • Collaboration • Accountability

Highsmith, J. (2000). Adaptive software development: a collaborative approach to managing complex systems, Dorset House Publishing Co., Inc.

Dynamic Systems Development Method (DSDM) by DSDM Consortium (www.dsdm.org)

Lifecycle Phases	Principles
• Flexibility study • Business study • Functional model iteration • System design and build iteration • Implementation	• Active user involvement is imperative • DSDM teams must be empowered to make decisions • The focus is on frequent delivery of products • Fitness for business purpose is the essential criterion for acceptance of deliverables • Iterative and incremental development is necessary to converge on an accurate business solution • All changes during development are reversible • Requirements are base-lined at a high level • Testing is integrated throughout the lifecycle • A collaborative and cooperative approach between all stakeholders is essential

Stapleton, J. (1997). Dsdm: The Method in Practice, Addison-Wesley Longman Publishing Co., Inc.

Feature-Driven Development (FDD) by Jeff Deluca and Peter Coad (Coad et al. 1999; Palmer et al. 2001)

Values	Process Description
• A system for building systems is necessary in order to scale to larger projects • A simple, well-defined process works best • Process steps should be logical and their worth immediately obvious to each team member • Process pride can keep the real work from happening • Good processes move to the background so team members can focus on results • Short, iterative, feature-driven life cycle are best	• Develop an overall model • Build a feature list • Plan by feature • Design by feature • Build by feature

Coad, P., J. deLuca, et al. (1999). Java Modeling Color with Uml: Enterprise Components and Process with Cdrom, Prentice Hall PTR.

Palmer, S. R. and M. Felsing (2001). A Practical Guide to Feature-Driven Development, Pearson Education.

Lean Development (LD) by Bob Charette (Charette 2002)	
Principles	Process Phases
• Satisfying the customer is the highest priority • Always provide the best value for the money • Success depends on active customer participation • Every lean development project is a team effort • Everything is changeable • Domain, not point solutions • Complete do not construct • An 80 percent solution today instead of 100 percent solution tomorrow • Minimisation is essential • Needs determine technology • Product growth is feature growth, not size growth • Never push lean development beyond its limit	• Start-up • Steady state • Transition and renewal
Charette, R. N. (2002). Foundation of Lean Development: The Lean Development Manager's Guide. The Foundations Series on Risk Management, (CD). Spotsylvania,Va.:ITABHI Corporation. 2. Poppendieck, M. and T. Poppendieck (2003). Lean Software Development: An Agile Toolkit, Addison-Wesley Longman Publishing Co., Inc.	

Table 3-1 Existing agile methods

3.6.1 Extreme Programming (XP)

XP is the most widely recognized agile method (Boehm et al. 2003). Kent Beck developed this method during his experience with the C3 project (Comprehensive Compensation System). XP practices were originally intended for small, collocated teams. Although some practitioners like Kent Beck and Ron Jeffries may envision that XP can be extended to larger teams, they do not try to convince people that it can work for teams of 200 (Highsmith 2002). XP is based on four values, communication, simplicity, feedback, and courage. XP practices include pair programming, continuous integration, refactoring, test-first programming, and user stories (Beck et al. 2004).

3.6.2 Scrum

Ken Schwaber and Jeff Sutherland developed Scrum when they realized that software development is an unpredictable activity. Scrum, along with XP is one of the most widely used agile methods (Ambler 2006). This method defines a project management framework, managed by the Scrum master. Scrum is one of the few agile methods that has been scaled up to medium projects (Boehm et al. 2003). In Scrum, the iteration length is 30 days and it is called a “sprint”. The sprint will be preceded by pre-sprint planning and will be followed by a post-sprint meeting. Scrum practices include the daily scrum meeting and product backlog (Schwaber et al. 2001).

3.6.3 Crystal Methods

Crystal is a family of methodologies that was developed by Alistair Cockburn. According to him, there is no one Crystal methodology but different Crystal methodologies for different types of projects. The factors that influence the methodology selection are staff size, system criticality, and project priorities. Crystal Clear is an optimization of Crystal family and it is targeted at projects where the team consists of two to eight people sitting in the same room or in adjacent offices. Cockburn stated that Crystal Clear shares some characteristics with XP but it is less demanding (Cockburn 2005).

3.7 Agile Surveys: The State of the Practice

Since the Agile Manifesto was introduced in 2001, the interest in agile methods increased over the years. In a review and analysis of agile software development methods that was conducted in 2002, the reviewers stated that there are not many experience reports available and that scientific studies are hard to find (Abrahamsson et al. 2002). These statements changed over the past 7 years as agile methods became more popular in industry. In this section we will present different agile surveys that were conducted over the years.

In 2003, Shine technologies run a web-based survey to measure the market interest in agile methods. They received 131 responses, from different organisations ranging from an online computer library centre to NASA. The results showed that companies that use agile have 49% lower costs, 93% better or significantly better productivity, 88% better or significantly better quality and 83% better or significantly better business satisfaction (Corporate Report 2003). In February 2006 a survey conducted by CM Crossroads ("www.cmcrossroads.com")has received 400 responses, and the results showed “*a very high level of interest in agile processes and the recognition that responding to changing business requirements and delivering value are the key success factors for development organisations*” (Barnett 2006b). In addition, the results showed that 43% of the respondents were driven to agile seeking quality improvement, and XP was the most widely used method then FDD and Scrum. The next survey is conducted by

Versionone Inc. ("www.versionone.com") an agile project management company. The survey was titled “The State of Agile Development” and it was run in 2006, 2007 2008 and 2009. The survey conducted in 2006 received 700 responses, and the results showed that “*agile practices deliver on their promises and can deliver significant rate of return across many types and sizes of organisations*” (Barnett 2006a). Interestingly, 82% of the respondents work in team of 10 or fewer and Scrum is the most widely used method. The 2007 survey was completed by 1681 individuals. The results stated that 73% of the respondents reported that their organisation is adopting agile software development practices. In addition, the results showed that agile methods have increased quality (90%), reduced defects (85%), accelerated time to market (83%) and reduced cost (66%) (Versionone.com 2007). The 2008 survey received 2319 and it focused on the state of organisations implementing agile methods rather than the adoption of agile methods. The reported results were similar to the survey conducted in 2007 regarding agile methods good impact on quality productivity, team morale and cost (versionone.com 2008). The results of the 2009 survey were not published at the time of writing this thesis.

The final series of surveys were conducted by Scott Ambler since 2006. The first survey received 4232 responses, and showed that XP, Scrum and FDD are the most popular methods; 60% of the respondents reported increased productivity; and 60% reported that the quality is better(Ambler 2006). The 2007 survey received 781 responses and the results indicated that agile techniques have been successfully adopted within the majority of organisations and often at scale. The results showed high success rate as 77% of the respondents indicated that 75% or more of their agile projects were successful. Regarding the effectiveness of agile practices, the high scoring practices were iterative development, regular delivery of working software, and simple design (Ambler 2007). The 2008 Survey received 642 responses. The survey results indicated that agile software development appears to still be growing in popularity. Furthermore, the majority of organisations are successfully adopting agile strategies. The results showed that the average success rate for agile teams was 77%, that agile approaches have good impact on productivity as only 5% indicated that their productivity was lowered. No change in productivity was reported by 13% of respondents and 81% reported increased productivity. Furthermore, agile approaches have good impact on quality, with 77% responding that the quality is higher compared to 66% in 2006 survey. 78% of organisations reported improved stakeholders satisfaction compared to 58%, whereas only 7% reported reduced satisfaction. Finally 37% reported that agile approaches helped reducing cost where 23% reported that it increased their cost (Ambler 2008b).

3.8 Conclusion: Definition of an Agile Method

The previous reviews and discussions were essential to form our understanding of agile methods. This section will provide our definition of an agile method. In other words, what makes a development method agile. An agile method is adaptive, iterative, and incremental, and people oriented

- Adaptive: an agile method welcomes change, in technology and requirements, even to the point of changing the method itself (Fowler 2005). In addition, it responds to feedback about previous work (Larman 2004). Fowler stated that an adaptive process is the one that can give control over unpredictability.
- Iterative and incremental: The software is developed in several iterations, each from planning to delivery. In each iteration part of the system is developed, tested, and improved while a new part is being developed. In each iteration, the functionality will be improved. In addition, the system is growing incrementally as new functionality is added with each release. After each iteration (s), a release will be delivered to the customer in order to get feedback.
- People-oriented: "*people are more important than any process. Good people with a good process will outperform good people with no process every time* (Booch 1995). In an agile method, people are the primary drivers of project success (Cockburn et al. 2001a). Therefore, the role of the process in an agile method is to support the development team to determine the best way to handle work (Fowler 2005). Furthermore, an agile method emphasises on face-to-face communication within the team and with the customer who is closely involved with the development process rather than written documents.

To summarise: Software development is an unpredictable activity; therefore, an adaptive process is needed to control this unpredictability. Iterative and incremental development is the best controller for this process, and creative and talented people are the best way to run it. This is illustrated in figure 3-1.

In this research, the phrase "agile software development" means developing software using an agile method, either a pre-defined one, or team defined, and in the later case it will follow the general definition provided above.

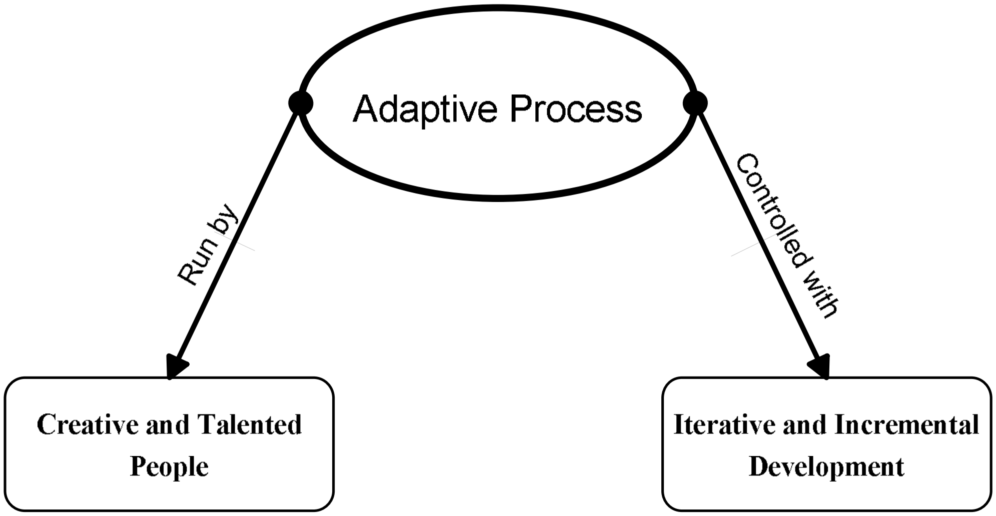

Adaptive Process
Run by
Controlled with
Creative and Talented People
Iterative and Incremental Development

Figure 3-1 Definition of an agile method

3.9 Summary

Agile methods are a reaction to traditional approaches and a result of people's experience as well as using ideas from the history. These methods are different from traditional approaches in terms of process, management, quality, and dealing with people and customer. Many surveys results showed that agile methods are widely used in industry and it seems that they have good impact on quality and productivity. This review was used to form our definition of an agile method as an adaptive process run by talented and creative people and controlled with iterative and incremental development.

Chapter 4 Software Quality Review

4.1 Introduction

The previous two chapters discussed software engineering methodologies and agile methods. As the purpose of this research is to investigate the impact of agile methods on software quality, it is important to review the literature on software quality and software quality assurance. This chapter will go through the available definitions of quality, software quality and software quality assurance. In addition, it will review the available software quality models, standards, and metrics. Furthermore, the chapter will look at quality in the agile world, including the conducted research around the topic, agile metrics, stakeholders' satisfaction and the impact of agile on software quality. Finally, the chapter will discuss the gaps in literature and will propose the initial research goals.

4.2 What is Quality

"*Quality is hard to define, impossible to measure, easy to recognize*" (Kitchenham et al. 1996). The literature provides different definitions for quality; every expert defines quality in a different way somewhat. In addition, quality means different things to different people such as users, customers, and managers. The Cambridge Advanced learner's Dictionary (Quality 2009) defines quality as 1) how good or bad something is, 2) a high standard. In his book Software Quality: Theory and Management, Gillies stated: "*quality is transparent when present, but easily recognized in its absence*". Important perspectives about quality are from the three quality "gurus" Philip Crosby, Joseph Juran, and Edward Deming (Gillies 1992). Crosby defines quality as conformance to requirements. Requirements must be clearly stated, thereby measurements are taken continually to determine conformance to the requirements (Crosby 1980). Juran defines quality as "*fitness for use*" (Juran et al. 1988). In his book "Out of Crisis", Deming defines quality as "*a predictable degree of uniformity and dependability at low cost and suited to the market*" (Deming 1982).

For a customer, quality is the customer's perceived value of the product. In his book "I know it when I see it", Guaspari stated, "*the customers are not buying a product, they are buying your assurance that their expectations for that product will be met*" (Guaspari 1991).

4.3 What is Software Quality?

The IEEE Glossary (IEEE 1990) defines software quality as:

1) The degree of which a system, component, or process meets specified requirements
2) The degree to which a system, component or process meets customer or user needs or expectations.

These two definitions reflect Crosby's (conformance to requirement) and Juran's (fitness for purpose) definitions.

Software quality will mean different things to different people. Each stakeholder has his/her views about quality. The user will see quality as "what I want", "fast response" or "cheap to run" where from the designer point of view it can mean "good specification" "technical correct" or "well documented" (Gillies 1992).

The de facto definition of software quality consists of two levels; the first is product quality, as limited to product defect rate and reliability. (Kan 2002) refers to this narrow definition as the "small q". The broader definition includes product quality, process quality and customer satisfaction, which is referred to as the "big Q". This definition has been used in a number of industries, such as automobile, software, hardware, and consumer electronics. Now although the final product confirms to requirements, it might not be what the customer wanted, therefore, the customer should define the quality as conformance to the customer's requirements.

Another view is from (Gillies 1992) who stated that "quality is people". He explained that this is because quality is determined by people who are facing the problem to be solved by the software, people will define the problem, people will find the solution and people will use the system and make judgments about the quality.

Gillies stated that software quality does appear to be particularly problematical when compared to other arenas. According to a group of professionals, this is because 1) software has no physical existence 2) the clients do not know exactly what they need 3) clients need change over time 4) the change in both hardware and software is very fast and 5) the customers have high level of expectations.

4.4 What is Software Quality Assurance (SQA)

The IEEE Glossary (IEEE 1990) definition of SQA is:

1) A planned systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established functional technical requirements
2) A set of activities designed to evaluate the process by which products are developed or manufactured

4.5 Software Quality Models

Quality models are needed in order to compare quality in different situations. There have been many models suggested for quality, mostly hierarchical such as McCall's and Boehm's models. The idea of hierarchical model of quality in software goes back to 1970s. A hierarchical model of software quality is based upon a set of quality criteria, each of which has a set of measures associated with. When talking about quality criteria, we should think of the following: which criteria should we employ, how they inter-relate, and what are the metrics associated with these criteria and how they may be combined into a meaningful overall measure of quality (Gillies 1992).

A quality model was introduced by McCall (McCall et al. 1977) (also known as the General Electrics Model of 1977) and it was aimed to be used during the development process. This model identifies three areas of software work: product operation, product revision, and product transition. Each area has a number of criteria as shown in figure 4.1. The description of each criterion is illustrated in table 4.1. The criteria in this model were chosen to reflect users' and developers' views in order to build a bridge between the two views (Gillies 1992).

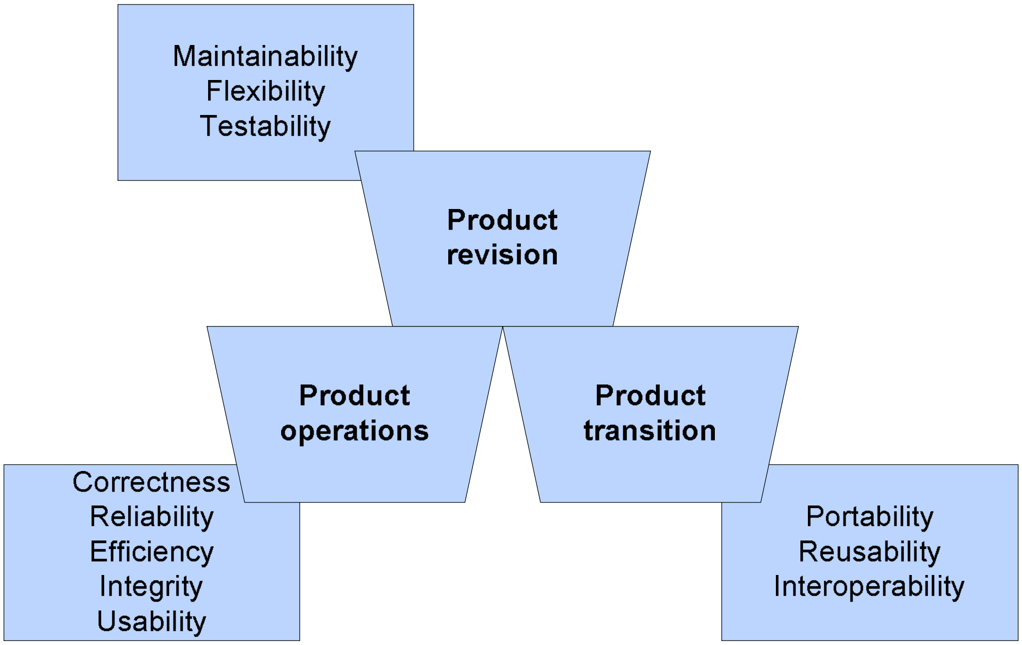


Figure 4-1 McCall's quality model (adapted from (Milicic 2006))

Software Quality Factor	Description
Correctness	The extent to which a program fulfils its specifications
Reliability	The extent to which a system perform its intended function without failure
Efficiency	The computing resources required by a system to perform a function
Integrity	The extend to which data and software are consistent and accurate across systems
Usability	The ease of use of the software
Maintainability	The effort required to locate and fix a fault in the program within its operation environment
Flexibility	The effort required to modify a system
Testability	The ease of testing the program, to ensure that it is error-free and meets its specification
Portability	The effort required to transfer a program from one hardware configuration and/or software environment to another or to extend the user base
Reusability	The ease of reusing software in different context
Interoperability	The effort required to couple the system to another system

Table 4-1 McCall's criteria of quality

Alternative models to the McCall's classic factor model appeared during the late 1980s: the Evans and Marciniak factor model and the Deutsch and Willis factor model. Both models exclude only one factor of McCall's factors, which is testability. The first alternative model consists of 12 factors classified in three categories; the second alternative model consists of 15 factors classified in four categories. Five new factors were introduced in these two models: verifiability and expandability (by both); safety, manageability and survivability (by Dutch and Wills). Further discussion about these models can be found in a number of sources (Evans et al. 1987; Deutsch et al. 1988; Galin 2003)

Boehm proposed another hierarchical quality model in 1976. Although this model shares a number of common characteristics with McCall's, it is based on a larger set of criteria. As seen in figure 4-2, there are three levels of hierarchy in Boehm's model; the high-level structure which reflects the actual uses made by the system. The lower level structure provides a set of primitive characteristics, which combine into sets of necessary conditions for the intermediate-level characteristics (Boehm et al. 1976).

Although McCall's and Boehm's models combine quality with quality metrics, the individual measures of software quality do not provide an overall measure of quality. Therefore, the individual measures have to be combined. However, these measures may conflict each other and the interrelationships between criteria can be direct, neutral, or inverse. This problem was addressed by Perry (Perry 1987) and the relationships are summarised in figure 4-3.

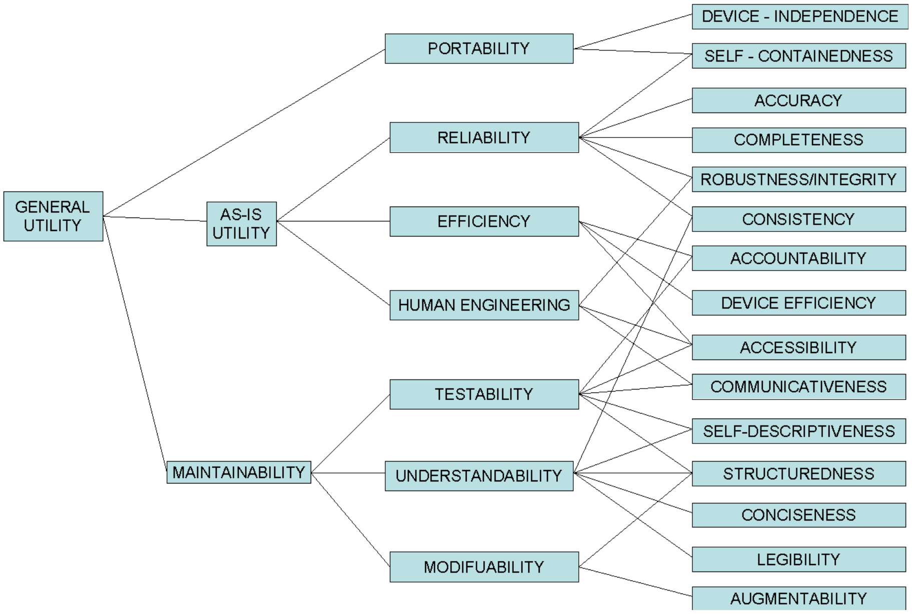
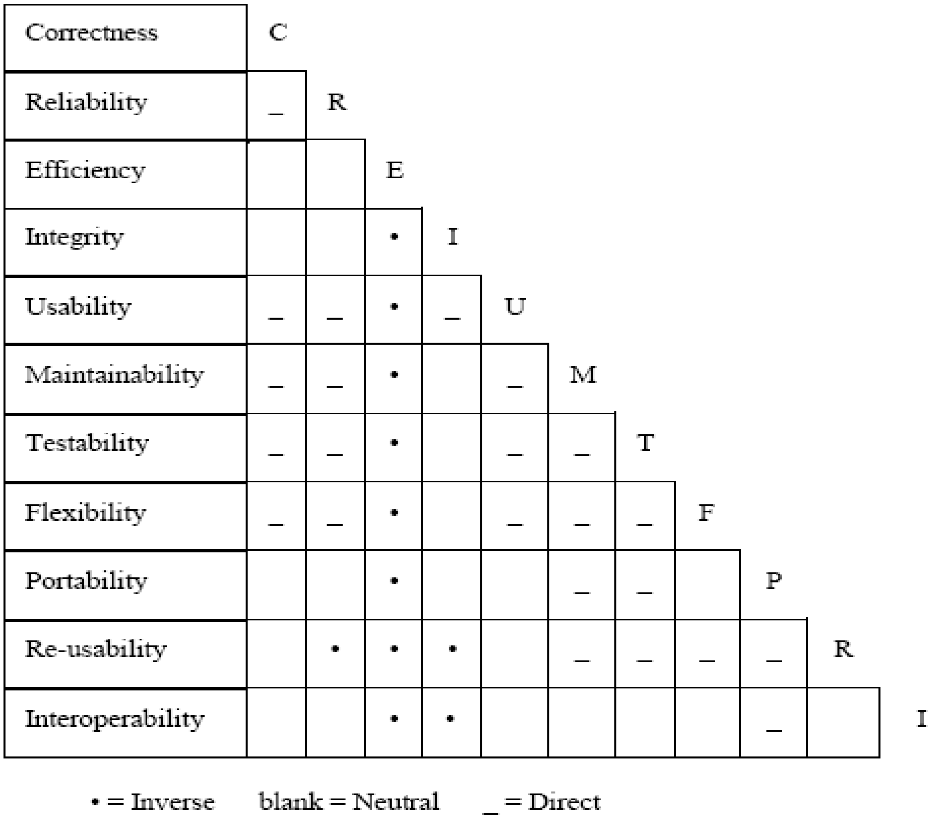


Figure 4-2 Boehm's software quality characteristics tree (Boehm et al. 1976)



Correctness	C										
Reliability	_	R									
Efficiency			E								
Integrity			•	I							
Usability	_	_	•	_	U						
Maintainability	_	_	•		_	M					
Testability	_	_	•		_	_	T				
Flexibility	_	_	•		_	_	_	F			
Portability			•			_	_		P		
Re-usability		•	•	•		_	_	_	_	R	
Interoperability			•	•					_		I

• = Inverse blank = Neutral _ = Direct

Figure 4-3 Relationships between Criteria after Perry (Perry 1987)

Gillies criticized Perry's model because it assumes that the relationships are commutative which is not always the case. For example, the model presents a direct relationship between testability and reliability, but this relationship does not go both ways.

4.6 Software Quality Assurance Activities

According to IEEE, software quality assurance is a set of actions necessary to achieve software quality including product and process quality. In this section, we will review the most common activities and will discuss their impact on software quality (singly and in combination). Table 4-2 summarises the activities we are going to discuss and the quality aspect(s) they are related to. Galin stated that quality assurance activities should be integrated into the development plan that implements one or more software development models. He suggested that the quality assurance plan for a project should include a list of quality assurance activities needed for the project, and for each activity, we need to determine timing, type, performer and resources.

The intensity of quality assurance activities is affected by two categories of factors: project factors (magnitude, technical complexity, amount of reusability, and criticality), and team factors (qualification, acquaintances, availability of support, and team dynamics) (Galin 2003).

Activity	Quality Aspect
Management reviews	Process quality
Technical reviews	Product quality
Inspections	Product quality
Walk-throughs	Product quality
Audits	Product quality & process quality
Refactoring	Product quality

Table 4-2 Software quality assurance activities

4.6.1 *Reviews and Audits*

The IEEE Glossary (IEEE 1990) defines reviews as:

A process or meeting during which a work product or set of work products is presented to project personnel, managers, users, customers, or other interested parties for comment or approval. Types include code review, design review, formal qualification review, requirements review, test readiness review.

According to the SWEBOK guide, there are five types of reviews or audits which are presented in the IEEE1028-97 standard for software reviews (IEEE 1998) . These five types are management reviews, technical reviews, inspections, walk-throughs, and audits. These types are summarised in table 4-3. The definitions are obtained from IEEE Glossary and from IEEE standard for software reviews.

Management reviews	To monitor progress, determine the status of plans and schedules, confirm requirements and their system’s allocation, or evaluate the effectiveness of management approaches used to achieve fitness for purpose
Technical reviews	To evaluate a software product to determine its suitability for its intended use. The result should provide the management with evidence that either confirm (or does not confirm) that the product meets the specification and adheres to standards, and that changes are controlled

Inspections	A static analysis technique that relies on visual examination of development products to detect errors, violations of development standards, and other problems. Types include code inspection, design inspection Purpose: To detect and identify software product anomalies
Walk-throughs	A static analysis technique in which a designer or programmer leads members of the development team and other interested parties through a segment of documentation or code and the participants ask questions and make comments about possible errors, violation of development standards and other problems Purpose: to evaluate a software product Less formal than inspection
Audits	To provide an independent evaluation of the conformance of software products and processes to applicable regulations standards, guidelines plans and procedures

Table 4-3 Software reviews types

4.6.2 *Refactoring*

“Any fool *can write a code that a computer can understand. Good programmers write code that humans can understand*” (Fowler et al. 1999). Refactoring is the process of changing software systems without changing the external behaviours of the code yet improves its internal structure. In other words, during refactoring, we change the design of the code after it has been written. This can be done by moving one field from one class to another, pulling some code out of a method to create a new method, or pushing some code up or down a hierarchy. The cumulative of all these small changes can improve the design. The first long writing on refactoring was from Bill Opdyke’s doctoral thesis (Opdyke 1992). Bill looked at refactoring from a tool builder’s perspective as he investigated how it could be useful for C++ framework (Fowler et al. 1999). Fowler provides the following definition of refactoring: a change made to internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour. As a verb, refactor is to restructure software by applying a series of refactoring without changing its observable behaviour.

There is no claim that refactoring will solve all software problems, yet it is a valuable tool. Martin Fowler provides the answer to why should you refactor:

- Improves the design of software
- Makes software easier to understand
- Helps you find bugs
- Helps you program faster

When should we do refactor? The most common time to refactor is when adding a new function. Refactoring will help understand the code that is going to be modified. In addition, refactoring is useful when we need to fix a bug. Another suggestion is to refactor during code review (Fowler et al. 1999).

4.7 Measuring Software Quality

For many years, measuring software productivity and quality was so difficult that only large companies such as IBM attempted it. The problem today is cultural. There is a cultural resistance due to the natural human belief that measures might be taken against them (Jones 1991).

According to Jones, in order to gain insights into quality problems we need to collect three kinds of data:

1. Hard data: things that can be quantified with little or no subjectivity. The key hard data elements are: number of staff, effort spent on a task, schedule durations, overlap of concurrency, documents, code, test case volumes, number of defects. Although hard data can be measured with high accuracy in theory, companies are inaccurate in the data they collect.
2. Soft data: major source of information that explains variation in productivity and quality. The key soft data elements are: skills and experience of the team, constrains or schedule pressure, stability of requirements, user satisfaction, the expertise and cooperation of users, adequacy of used tools and methods, organisation structure, adequacy of office space, perceived value of the project. Soft data are the most useful kind of information that can be collected, although, collecting soft data is the most difficult intellectual task associated with measurement programs. This is because human opinions must be evaluated, soft data are intrinsically subjective, and absolute precision is impossible.
3. Normalised data: standard metrics used for comparative purpose to determine whether projects are above or below normal in terms of quality and productivity. Examples of normalised data can be lines of code and functional points.

4.7.1 *Software Quality Metrics*

A metric is a measurable property which is an indication of one or more of the quality criteria that we are seeking to measure (Gillies 1992). Metrics can be predictive (to make predictions about software during the lifecycle) or descriptive (the state or the software at the time of measurement). (Kan 2002) classified software metrics into three categories: product metrics, process metrics and project metrics. Furthermore, he stated that software quality metrics are a subset of software metrics that focus on quality aspects of the process, product, and project. Kan discussed product quality metrics based on the de facto definition of software quality. The metrics he suggested are mean time to failure, defect density, customer problems, and customer satisfaction.

(Watts 1987) provided a list of 40 metrics available in the literature. Although this might sounds like a good number, the distribution of the metrics across quality factors is not even. Some quality criteria (such as efficiency, adaptability, interoperability, reusability) have no defined metrics to measure at all, whereas other criteria (such as maintainability, reliability) have many (18, 12 respectively) metrics to measure them (Gillies 1992).

Gillies stated that metrics available from the literature are limited because of the following reasons:

- metrics cannot be validated
- they are not generally objective
- quality is relative, not absolute quantity
- metrics depend upon a small set of measurable properties
- metrics do not measure the complete set of quality criteria
- a metric can measure more than one criterion

The Goal/Question/Metric approach was proposed for defining measurable goals (Basili et al. 1994). The GQM model is a hierarchical structure as seen in figure 4-4. The model starts with a specified goal of measurement. The goal is refined into several questions. These questions break down the issue into its major components. Each question is then refined into metrics that can be objective or subjective. The same metric can be used in order to answer different questions under the same goal.

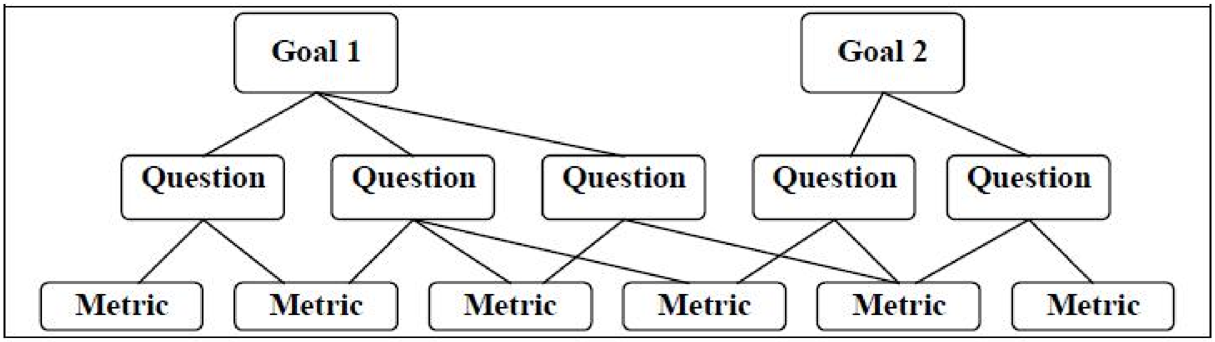


Figure 4-4 The Goal/Question/Metric approach (Basili et al. 1994)

4.8 Total Quality Management

The term “Total Quality Management” (TQM) was originally invented in 1985 by the Naval Air Systems Command to describe its Japanese-style management approach to quality improvements. TQM represents generally a style of management that aimed to achieve long-term success by linking quality and customer satisfaction. This term has taken different meanings and was implemented in different ways depending on how it was applied and who was applying it. Despite the different implementations of the TQM, Kan suggested that it has four key elements as can be seen in figure 4-5 (Kan 2002):

- Customer focus: to achieve total customer satisfaction by studying customer needs and managing and measuring customer satisfaction
- Process improvements: to achieve process continuous improvement with enhancing product quality
- Human side of quality: to achieve company wide quality culture by empowering people, and focusing on human factors
- Measurement and analysis: To achieve a goal-oriented management system that drives process improvements in all quality aspects

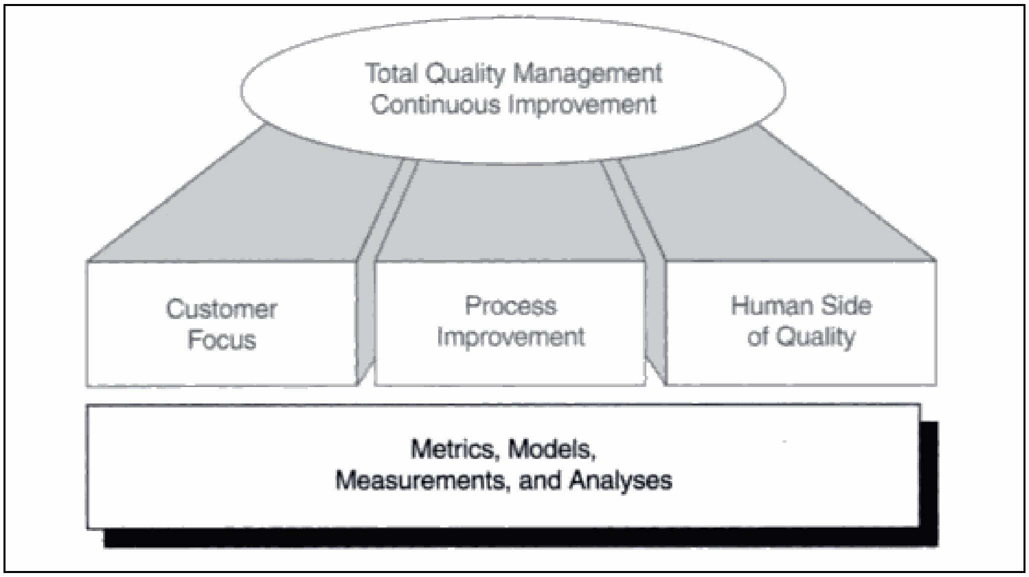


Figure 4-5 Key elements of Total Quality Management (Kan 2002)

In order to better understand the term TQM, we have looked at available definitions. Although TQM is widely used in practice and it is taught as an academic subject, there is a little agreement on what it actually means. This was observed by Boaden who proposed a list of elements for TQM based on the literature (Boaden 1997). We could observe that the proposed list meets the four elements proposed by Kan with more details:

- Customer focus (customer factor)
- The commitment and involvement of everyone to quality improvements (quality culture)
- A focus on processes (process improvement)
- Training and education considered as investment (human factor)
- The use of teams and teamwork (human factor)
- The use of appropriate tools and techniques reviewed regularly (process improvement)
- Goal setting, measurements and feedback for all aspects of the business (measurements/process improvements/customer satisfaction)
- A change in the organisation culture (human factor)
- Including quality principles into product and service design (process improvement)

4.9 Who is Responsible for Assuring Software Quality

It is important to know what are the needed activities to achieve high quality software and to satisfy the customer. It is also important to know who is responsible of these activities and how to achieve these goals. Traditional books on quality assurance such as (Galin 2003), list the actors in a typical quality assurance organisational framework:

- Managers: top management executives, software development managers, software testing managers, project managers, team leaders, testing teams' leaders.
- Testers: members of testing teams
- SQA professionals and interested practitioners: this group refers to staff that dedicate all their work related activities to the SQA, namely SQA unit. In addition, there are bodies such as SQA trustees, SQA committee members and SQA forum members. These actors have part time SQA responsibilities or they are employees who "*volunteer their time to their interest in quality*".

Although Galin would like to assume that most, if not all, of the organisation's staff are expected to contribute their share to quality of the organisational products; we argue that quality assurance should be integrated into the development process and should be part of everybody's job by default. Also we argue that the whole team is responsible of quality assurance, not only managers, testers and SQA professionals.

4.10 Quality in Agile World: The State of the Art

In previous sections, an overview of software quality, software quality assurance, and metrics was presented. It can easily be seen that most of the ideas and the models are based on sequential software development; saying that, some work has been done on agile methods and software quality. This section will review the research that discussed agile methods and the different aspects of quality.

4.10.1 *Agile Methods and Software Quality*

A reference source titled "Agile Software Development Quality Assurance" was published in 2007. This book (Stamelos et al. 2007) puts together 12 chapters from different authors discussing different aspects of agile methods and quality. The first chapter discusses how software quality parameters can be mapped to agile techniques. Chapter 5 describes a process for the "*recurring and sustainable discovery, handling, and treatment of quality defects in software systems*". In addition, the author introduces a tool for assuring the quality in that context. Chapters 10, 11, 12 provide different experience reports and lesson learned about agile software development and quality.

The agile practices' quality assurance abilities and their frequency were analysed in (Huo et al. 2004). The authors concluded that agile methods do have practices that have quality assurance abilities, some are integrated within the development phase, and some others can be separated out as supporting practices. In addition, the authors reported that the frequency with which agile quality practices occurs is higher than in a waterfall development, In addition these practices are available in very early process stages due to the agile process characteristics.

Another paper highlighted the possible activities that can be done to improve the already high quality achievements realised in agile methodologies, and the agile quality techniques for specific agile methods (Mnkandla et al. 2006).

In addition, researchers have studied the relation between agile methods and software quality empirically. Since this research started in 2006 the number of empirical studies that investigated the impact of agile methods on the different aspects of quality has increased. This review will focus on studies that demonstrated high quality research according to (Dyba et al. 2008), studies that were conducted in real settings rather than students' projects, and studies that were based on a well described empirical methodology. A systematic review of empirical studies of agile software development up to and including 2005 was conducted and published in 2008 (Dyba et al. 2008). This review found that five studies examined product quality. For example, one study evaluated the effects of adopting Extreme Programming (XP) and reported 50% increase in productivity, a 65% improvement in pre-release quality and 35% in post-release quality (Layman et al. 2004).

Researchers run experiments to investigate the impact of different agile practices on software quality. One example is a study that compared the use of test-driven development (TDD) in three software development projects. The results showed that the effect of TDD on program design was not as evident as expected, but the test coverage was significantly superior to iterative test-last development (Siniaalto et al. 2007). Another study was based on a post hoc analysis of the results of an IBM team who has sustained the use of TDD for five years. The study reported that TDD practice can aid in production of high quality products (Sanchez et al. 2007).

An empirical analysis that compared quality assurance in XP and spiral model reported that the quality improvement activities are build-in in XP and they happen almost at the same time. Also the frequency of these activities in XP is greater than the spiral model because of the iterative nature to XP. The empirical findings detected no significance superiority of one process to another process. The findings of the case projects did not indicate that the code produced by

XP had more fault density in comparison with the spiral model. In fact, the inverse was true (Hashmi et al. 2007).

4.10.2 *Agile Methods and Quality Standards*

There have been many initiatives to relate agile with quality standards such as CMMI and ISO and Six Sigma. During a workshop in the University of Southern California, the components of CMMI were evaluated for their support of agile methods. As a result, although there are significant differences, there is much in common between the two world views (Turner et al. 2002).

Other initiatives tried to apply a combination of agile methods such as XP and Scrum to satisfy the requirements of CMMI level 2 or 3 (Anderson 2005; Alegria et al. 2007), ISO9000 (Nawrocki et al. 2002a) or the Sommerville-Sawyer model (Nawrocki et al. 2002b), or even to achieve a combination of these standards (Vriens 2003).

The results of these initiatives vary; some of them claim that it is possible to develop an agile lifecycle process that can meet the requirements for all levels of CMMI model (Anderson 2005). Others realized that it is possible to achieve CMMI level 2 if a number of issues were addressed (Alegria et al. 2007). Others had to modify agile methods so they would be acceptable from the standards point of view (Nawrocki et al. 2002a). The authors in (Hashmi et al. 2008) concluded that using Six Sigma with XP can further improve the estimation and accuracy and customer satisfaction of XP products by minimizing the defects and meeting the project schedule.

We can see different opinions and results. Probably this is because the projects and the organisations, where researchers apply their initiatives, vary. An interesting question is whether we really need to modify the method in order to achieve the standards. In other words, if the methodology satisfies our needs, why modify it? So why not create a standard that can be suitable for the new methodology? The main reason for introducing CMMI was to modify CMM to be more suitable for modern iterative development, because few of the modern principles are in conflict with CMM key process areas (Royce 2002).

4.10.3 *Agile Methods and Metrics*

Some work has been done regarding agile metrics and how to measure agile projects progress. However, most of the discussion around agile metrics was found in internet blogs and forums, yet a few published papers are available.

Burn charts are the most visible technique to show a project's progress. They show the progress made against predictions and open the door to discussions about how best to precede, cut scope or extend the schedule (Cockburn 2005). There are two types of burn charts: burn-up charts which show the increasing amount of functionality over time. Burn up charts are similar to the earned-value chart (Let 1998) with one essential difference. The earned-value chart includes the tasks completed even if they did not result in code. In burn-up charts, credit is only given when the code is integrated and tested (Cockburn 2005). The second kind is the burn-down charts, which are used in Scrum. These charts will show the number of tasks remaining for the current sprint or the number of items on the product backlog. Cockburn stated that burn-down chart is emotionally powerful because "*hitting the number zero helps people get excited about completing their work and pressing forward*". One tricky point about the burn up chart is choosing the unit for the vertical axis. The line of code (LOC) code is one option; however the problem with LOC is that the number of lines of code needed will not be known until the end of the project, and if something goes bad, we will only know at the very end. Cockburn suggested choosing a smaller non-expandable unit of measure. This might be a stable number of use cases, or a fixed number of modules to be replaced. Story points are used as unit of measure in the burn-down chart within Scrum.

In respect to agile metrics, Ron Jefferies (Jeffries 2004) suggested a metric for agile projects. He called it running testing features (RTF) metric, which "*shows at every moment in the project how many features are passing all their acceptance tests*". He stated that RTF should increase linearly from day one through the end of the project. According to Jeffries:

- Running means the features are shipped in a single integrated product
- Tested means the features are continuously passing tests provided by requirements givers
- Features means real end-user feature or pieces of the customer-given requirements

Another common metric captured by agile teams is their velocity. Velocity is an agile measure of how much work a team can do during a given iteration. It can be calculated by dividing the completed tasks by time. It seems like velocity is widely used in the agile community as it appears regularly on agile email groups and online discussion groups. Scott Ambler stated that velocity can be used to measure and compare the productivity of two teams by calculating the acceleration of each team which is the change of velocity over time (Ambler 2008a). Other available metrics are the Obstacles Removed per Iteration metric (OR/I) which is the number of obstacles closed as removed in a single iteration, and the Time To Obstacle Removal metric (TTOR) which is the sum of time elapsed for all unresolved obstacles divided by number of unresolved obstacles (WebBlog 2005).

Hartmann and Dymond discussed the appropriate agile metrics and recommended focusing on measuring the outcome rather than the input, using a small, easy to collect, set of metrics that reinforce the agile principles (Hartmann et al. 2006).

4.10.4 *Agile Methods and Stakeholders Communication and Satisfaction*

When the different definitions of quality and software quality were reviewed earlier in this chapter, customer satisfaction appeared as an important attribute of quality. Furthermore, Kan stated that customer satisfaction is the ultimate validation of quality (Kan 2002). In addition, the total quality management (TQM) aimed at long run business success by linking quality with customer satisfaction. The agile manifesto values individuals and appreciates customer collaboration (Highsmith et al. 2001); therefore, we will extend customer satisfaction to cover different stakeholders' satisfaction. This may include employee stakeholder's satisfaction as well as customer stakeholder's satisfaction. As the term “stakeholders” was used loosely in research (Strong et al. 2001), it will be used to refer to customers or employees or both depending on the context or the discussion. In critical situation where it may be confusing, we will be clear about the specific stakeholders.

Existing empirical research focused on stakeholders', communication and satisfaction. An empirical study published in 2008 investigated the impact of five agile practices on stakeholder satisfaction, namely iterative development, continuous integration, collective ownership, test driven development and feedback. The study reported that these agile practices had a positive impact on stakeholders satisfaction (Ferreira et al. 2008). The study however did not clarify what is the type of stakeholder's under investigation.

On the communication with the customer, a study examined the impact of XP and Scrum practices on communication within software development teams. The study showed that agile practices had a positive impact on internal communication between the development team as well as external communication between other stakeholders and the software development team. However, in larger development situations that involved multiple external stakeholders, a "*mismatch of adequate communication mechanisms can sometimes even hinder the communication*" (Pikkarainen et al. 2008).

Finally, a research about communication with stakeholders was conducted using a qualitative case study on a small company that has turned from a waterfall-like process to evolutionary project management (EVO) (Gilb 1985) .The findings reported that close customer engagement does give certain benefits but that it comes with a cost and needs careful attention to management (Hanssen et al. 2006).

4.11 Discussions

The previous literature review helped us understand the general view of software quality and its different aspects. Also, it gave us an idea about the research conducted regarding agile methods, software quality, and stakeholder's communication and satisfaction.

In an article published in 2005 (Ambler 2005), Ambler argued that agile software development techniques lead to much higher quality software than what traditional software teams usually deliver. He stated, "*the greater emphasis on quality implies a changed and perhaps even smaller role for quality professionals on agile development projects*". We agree with Ambler that agile development naturally lead to higher product quality. Agile proponents claim that applying agile methods will improve software quality (Cockburn et al. 2001a; Highsmith 2002; Poppendieck et al. 2003; Larman 2004). Furthermore, the empirical studies discussed previously reported positive impact of agile methods, mainly XP, on product quality. Also the different agile adoption surveys results presented in the previous chapter showed that agile methods had good impact on quality.

A classification of the causes of software errors was presented in (Galin 2003). Assuming that software errors are the cause of poor software quality, it is arguable that agile software development reduces these errors by default. This is mainly because the causes of errors can be solved in agile development. Table 4-4 summarise Galin's causes of errors with our proposed agile solution.

Cause of Software Error	Agile Solution
Faulty definition of requirements	Iteration planning
Client-developer communication failure	Heavy client involvement
Deliberate deviation from software requirements	Customer involvement
Logical design errors	Refactoring
Coding errors	Refactoring
Non-compliance with documentation and coding instructions	Continues integration, share understanding, pair programming
Shortcoming of testing progress	Early testing, test driven development
User interface and procedure errors	Customer/user involvement and continues feedback
Documentation errors	Up to date documentation

Table 4-4 Causes of software errors and their agile solutions

From the previous discussion and the literature review we can form a general hypothesis that the use agile software development improves product quality. However, product quality is one aspect, what about the other aspects of quality, namely process quality, and stakeholder satisfaction.

The systematic review of empirical studies of agile software development up to and including 2005 identified 1996 studies of which 36 were found to be empirical studies. Thirty-three were primary studies and three were secondary studies. The thirty three studies were categorised in four groups: introduction and adoption, human and social factors, customer and developer perceptions and comparative studies. The study showed that 25 (76%) of the considered studies were done on XP, 24 (73%) of the studies dealt with employees who are beginners, and only 4 (12%) studies dealt with mature agile development teams, and found no empirical studies of agile development prior to 2001, (Dyba et al. 2008). Although the systematic review showed that good number of empirical studies about agile development has been conducted, still these are mainly focused on one agile method, namely XP, and there is very little focus on the different aspects of quality.

A preliminary roadmap for empirical research on agile software development was published in 2008 (Dingsøyr et al. 2008). In this roadmap, the authors suggested that researchers in the field should focus on providing more empirical research especially on experienced agile teams and organisations.

Based on what have mentioned above, and on the literature review our initial research questions and goals were formed, firstly to empirically investigate the impact of agile software development on the different aspects of quality, then to form a model, a framework, or a chick list that can capture the different aspects of quality in agile software development.

4.12 Summary

Software quality is a large topic with many subtopics and areas. Most of the work has been done about software quality is based on the sequential model; also, the available models are designed for traditional approaches to develop software. Some work has been done to study quality in agile software development; this included empirical studies, linking agile methods with standards, and suggesting metrics for agile projects. This review resulted in an overall hypothesis that the use of agile software development improves product quality. However, more empirical evidence is needed to support this hypothesis and to study the impact of agile methods on other aspects of quality namely process quality and stakeholders satisfaction.

Chapter 5

Quality in Agile Projects: An Empirical Study

5.1 Introduction

Although many papers, articles, and books have been published about agile methods, empirical studies about how organisations adopt agile methods are still needed. In addition, when a new methodology is introduced, the evidence for what does work and what does not is needed. Most importantly, we need more studies about the impact of agile methods on software quality.

This chapter will present our qualitative empirical study, which included studying two agile projects within IBM® with focusing on quality. The chapter will review the related work and studies, the nature of the empirical research and the methodology we used to collect and analyse the data. Then the chapter will discuss each project's results, and a comparison between the two projects, as well as what is bad and what is good about agile methods and a comparison between traditional approaches and agile methods based on our interviewees' experiences. Finally, the generated hypotheses will be presented and reviewed.

5.2 Background and Related Work

The study of software engineering has always been complex and difficult. This is mainly because of the intersection of machine and human capabilities (Seaman 1999). Therefore, and because software development is a human-based activity (Basili et al. 2007), we need to apply empirical studies in order to understand important problems in the domain. Organisations need to know what are the right processes for their businesses, what is the right combination of methods, and they need answers that are supported with empirical evidence.

The previous chapter reviewed the conducted research regarding agile methods and quality. However, there is a "*need for more and better empirical studies of agile development within a common research agenda*" (Dyba et al. 2008). Therefore, we decided to use empirical methods to investigate the impact of agile software development on the different aspects of quality.

The empirical studies reviewed previously gave us an idea about the different research methods used by different researchers when conducting empirical research. In order to broaden our knowledge regarding empirical software engineering, we took a look at the available references such as (Basili et al. 1986; Basili 1996; Seaman et al. 1998; Seaman 1999; Perry et al. 2000; Wohlin et al. 2000; Kitchenham et al. 2002; Sharp et al. 2004b; Zannier et al. 2006; Basili et al. 2007). We focused on studies that investigated agile methods and agile software development.

The systematic review of empirical studies of agile software development focused on thirty three primary studies. The studies were categorised in four groups: introduction and adoption, human and social factors, customer and developer perceptions, and comparative studies. Different empirical approaches were used to conduct these studies as can be seen in table 5-1.

Research method	Number	Percent
Single-Case	13	39
Multiple-case	11	33
Survey	4	12
Experiment	3	9
Mixed	2	6
Total	33	100

Table 5-1 Agile empirical studies by research methods (Dyba et al. 2008)

Of the 13 single-case studies, nine were done on projects in industrial settings; the other four were conducted on students' projects. Three of these studies took their data from the same project. Regarding maturity, only one single-case study in industry was done on a mature development team. All the 11 multiple-case studies were conducted in industry; only three of these studies were on mature teams. Three of the four surveys were done on employees in software companies, while one survey was done on students. The three experiments were all done on students, with team sizes ranging from three to 16. For the two mixed-methods studies, one reported on a survey amongst students in addition to interviews and notes from discussions. The other study reported on 10 case studies in companies, as well as the findings from discussion groups.

The empirical study conducted in this research is done on mature teams; therefore we took a closer look on the four studies cited by the systematic review that fit this category. The studies had different focus, the first one focused the human factor and it explored the nature of interaction between organisational culture and XP practices (Robinson et al. 2005a). The second paper focused on the social side of XP practices (Robinson et al. 2005b) and the third focused on team characteristics and discussed how these characteristics are embedded in XP practices used in two particular settings (Robinson et al. 2004). The final paper was a study of XP practices and it identified five characterising themes within XP practices (Sharp et al. 2004a). The approach used in these papers was an ethnographic approach (Fielding 2001) which is a non-subjective approach that forces researchers to attend to the taken-for-granted, accepted, and un-remarked aspects of practice, considering all activities as ‘‘strange’’ so as to prevent the researchers’ own backgrounds affecting their observations. The researches applied this approach by using close observation of the day to-day business of XP development, documenting practices using field notes, photograph of the physical layout, copies of documents, records of meetings, and discussions and informal interviews with practitioners.

The systematic review presented the empirical studies up to and including 2005. More recently, two empirical studies about the use of agile methods were published in the Journal of Empirical Software Engineering in 2006. The first one discussed the advantages and difficulties which 15 Greek software companies experience applying XP. The study was conducted using sample survey techniques with questionnaires and interviews. The paper concluded that pair programming and test-driven development were found to be the most significant success factors in addition to interactions, communication between skilled people (Sfetsos et al. 2006). The second paper presented a qualitative case study of two large independent software system projects that have XP for software development within context of stage-gate project management models. The study was conducted using open-ended interviews. The paper concluded that it is possible to integrate XP in a gate model context, and the success factors are the interfaces towards the agile subproject and the management attitudes towards the agile approach (Karlstr et al. 2006).

The previous review convinced us that the best approach to answer the identified research questions is to apply empirical methods including qualitative and quantitative ones.

5.3 The Empirical Study

When studying a human-based activity such as software development, the research must deal with the study of human activities, preferably, within a real world settings (Basili et al. 2007). Qualitative methods are designed to study the complexities of human behaviours (Seaman 1999). Qualitative data can be represented as words and pictures, not numbers. Qualitative research is mainly useful in the early stages of research when no well-known theories or hypotheses have previously been put forth in an area of study. As this is the case for the adoption of agile methods and their impact on the different aspect of quality, we started the research with qualitative methods, namely case studies (Yin 2003).

The next step was finding the suitable case studies. The best option was a mature team that is using agile software development who will agree to participate in the study. We had the opportunity to study two agile projects within IBM. Both did not apply a specific agile method but they adopted agile practice and principles so in effect they had their own agile method. The collaboration with the teams started with a personal communication between the research supervisor and the team lead of one of the projects who introduced us (myself and my research supervisor) to the project manager of the second team.

5.3.1 *Data Collection*

Semi-structured interviews (Wohlin et al. 2000) were used to collect the data. Interviewing people provides insights into their work, their opinions and thoughts (Hove et al. 2005). Two projects were studied within IBM in parallel. As total, 13 interviews were conducted, six with team A and seven with team B. Each interview lasted, on average, one hour. Most of the interviews (10) were conducted from November 2006 to December 2007. The other three interviews were a follow up with the project managers to see how did the project changed over the time. The last three interviews were conducted in October 2008, December 2008 and the last one was in June 2009.

The interviews were informal and conversational. In the early interviews, we asked initial questions about general agile projects experience: number of projects, size of projects, working with agile vs. traditional approaches if any existed, and how they rate the quality of an agile project in terms of code quality and customer satisfaction. Also, we asked about the interviewee experience in the current project: communication within the team, with customers, iteration and incremental development, and how satisfied they are with the whole process. In later interviews, and as research evolved, we added more questions with more focus on product and process quality, as well as stakeholders' satisfaction (See Appendix D for a list of the interview questions).

In each interview, two researchers were present (myself and the research supervisor) and both took notes. In the same day of the interview, the notes were reviewed and written up. Having two researchers taking notes was used in a study of COTS integration within NASA (Seaman 1999). It was difficult to use audio taping because of the company restrictions, for the same reason we could not study the projects documents. Therefore, we decided to use interviews with note taking by both researchers, which was successful in getting the required level of detail with an acceptable level of accuracy.

5.3.2 *Data Analysis*

As mentioned in the previous section, the field notes were written up and reviewed. Each interview produced, on average, eight pages (A4 size). In order to analyse the interviews the constant comparison method, described by Glaser and Strauss (Glaser et al. 1967), was used. This method is one of the theory generation methods, which are generally used to extract a statement from a set of field notes which is supported in multiple ways by the data (Glaser et al. 1967). We were influenced by the guidance from Carolyn Seaman to use this method for software engineering empirical research (Seaman 1999). When using this method, we start with coding the field notes, which means attaching labels to pieces of text that is relevant to a particular theme or topic. Then a list of codes will be generated while reading the data, with a big influence of the research questions. The result is a list of categories and codes (see Appendix D).

The next step was to group the passages of text into patterns and themes according to the codes. The paragraphs or sentences were not cut and paste as this might affect the context of the data, instead the word processor's find feature was used to trace each code. After that, field memos were written to record our observations from the coded data. These field memos are the base for the results presented in the next section, and they will articulate the preliminary grounded hypotheses.

The results will describe the agile adoption in each project as it was described by the interviewees with minimal subjective views from the researchers. The results will be organised in three categories: people, process, and quality. Each category includes sub-categories, which represent the codes. These three categories and their odes will be discussed for each project. We thought that this is the best way to present our data, not only because of the big amount of data we have but because it provides a clearer way to understand agile adoption in each project and the relationship between different codes.

5.4 The Results: Project A

Project A started in January 2007 as an experiment in the organisation: a new way of working. At the beginning, the project was called project 0. The first release was live on the company website in June 2007 with informal support only. Since then the project became part of a larger project that involved different teams in different locations. In May 2008, the first release of the new merged project was out. Figure 5-1 shows the project progress. Version 1.1 was released in December 2008. The results presented here are mainly related to the initial project (before the merge) unless specified. The interviews with team A were conducted between November 2006 and June 2007. We interviewed four people working with the project, a team lead (twice), a tester, a developer, and an architect. An additional interview with the team lead was conducted in December 2008.

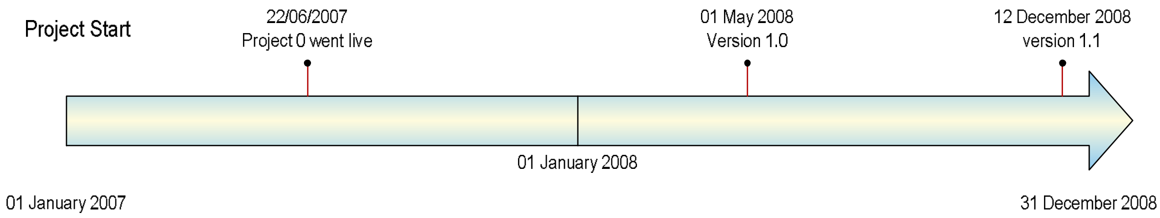


Figure 5-1 Project A progress

5.4.1 *The Process*

- Iteration Planning

When the project started it did not use any specific agile method. The team followed a 2 weeks iteration that begins with a list of priorities (tasks). The team used agile modelling on whiteboard and discussions to refine and tune the plan for the next iteration. Although the senior team is doing the design, the whole team should understand the architecture; therefore, it is reviewed by the whole team and continuously improved over the iterations. Decisions to drop line items are not very strict or formal; the team may roll them over to the next iteration or reword them to close off the iteration.

During the last interview with the team lead, he stated that the process has changed and it became very influenced by Scrum. The team is using iteration planning, review meetings including demos to stakeholders, and web conferences with business partners (external customers). As the team is part of a bigger team, they are using Scrum of Scrums. There is two levels of planning, 2 weeks iteration for each team internally and 6 weeks iteration which is an overall planning phase for the whole product. The 6 weeks iteration long enough to allow time for IBM legal reviews. At the end of the 2 weeks period, the iteration delivery is live on the website. The 6 weeks iteration starts on with a planning meeting for Scrum masters of all teams with the chief architecture who goes through a list of functional requirements and on the next day each team goes back with its tasks. One of the crucial points in the project is to have all the teams are working in an agile way. As working together is more needed in agile, however one challenge was when a team is not delivering as it affects the other teams.

- Estimation

Estimation is in days but the team is moving to story points so velocity can easily be measured and improved.

- Meetings

The team has two meetings for the iteration preparation on two levels, one for the senior team (the team lead and the sub-teams leaders) and one for the whole team. The senior team meeting produces a “straw man” list of items. This list is discussed and refined in the first day of the iteration with the whole team where the tasks are allocated to developers. In this whole team meeting, the team go over the status of the previous iteration and say “well done,” go through each goal for the current iteration and who is responsible, and schedule design sessions which take place during the week. In these sessions, the senior team will be involved and they work on the architecture using UML-like diagrams. During the iteration, the team has daily stand up Scrum meeting for 15 minutes where each team member says a couple of sentences to describe what he/she is working on, which may lead to further communication.

The senior team meets three times a week for half an hour to discuss planning issues, feedback from customer and bugs list. There is an off-site test team and they meet once a week with the test team lead, who is on-site, for half an hour through a formal phone call to agree responsibilities. In addition, a test meeting will take place on the day before the iteration planning meeting.

Every Friday afternoon the senior team meets to discuss what they have done to check that every thing is ready. In earlier iterations, the team lead went around to give feedback to developers. In addition, they have a weekly chalk and talk session; originally, it was for learning purposes, later on they used it for explaining key areas. The team different meetings are presented in table 5-2.

- Tidy up Iteration

An interesting practice was to have an iteration for stabilization, consolidation and to improve code quality. Out of 13 iterations, three were devoted to this purpose. These tidy-up-iterations help to pace the work, fix problems, and allow some breathing space for the team. The consolidation iteration happens every six iterations or as needed. In addition to resolving defects, these iterations can be used to slow the pace and allow some time for reflection.

Duration	When	Whom	Purpose
15 min	Daily 11:45-12:00	Whole team	What they did, what to do
	Thursday before iteration	Senior team	Produce list of line items based on phone calls with customers and a loose idea of what will come up in the future iterations
1 hour	First day of iteration (Monday) 9:30 -10:30	Whole team	Share, discuss and refine ideas from Thursday meeting and assign tasks
1 hour	Every Wednesday 10-11	Whole team	Originally learning, then became technical talks
30 min	(Tue, Thu, Fri) 9:00-9:30	Senior team	Discuss issues (bugs list, feedback, funding, plan changes)
-	Fridays afternoon	Senior team	What we have done to check that everything is ready
-	Monday or Thursday	Senior team	Design sessions – go through these with the whole team once a month
30 min	Every Friday	Test team	Plan next iteration with the off-site test team on the phone

Table 5-2 Meetings in project A

- Testing, Automated Testing, and Relation between Testing and Development Team

Testing started just after writing the project's high-level statement. Developers typically write unit tests using Junit and other frameworks. The test team try to keep ahead of developers in writing functional tests so developers can use them while writing their code. Having separate testers can make them more motivated to find bugs. The team lead reviews, selects the tests, and adds them to the build. When the code is checked in, it needs 30 minutes to build and then they run all the tests. In June 2007, the project had 100 new functional tests written by the test team; these tests are longer and more complicated than the tests written by the developers.

There is a light on the team lead's desk to show when the build is broken. This light will be green when everything is quiet, blue when it is building, and red when the build has failed. When the light is red the person who last checked in the code will be first to investigate. When the requirements are met, the test suite is enabled. Builds pick up test suits and produce a report to show the status of each function, if any test fails, the build fails in which case should be fixed in approximately 30 minutes.

The team found it important to be able to automate tests as part of continues integration. The team has three levels of automation: instant (unit tests), longer tests which can be combined in a suite that completes in less than half and hour and the long running tests that includes the overnight re-build. In the last interview with team lead, he mentioned that the system verification test has not changed.

- Requirements

Validating requirements from product management is very important. The team lead meets the customers to get feedback on product plans. A test is assigned to each requirement, which improves the focus and the result of the testing especially with agile as this will make traceability easier.

Risk was used to priorities the requirements. Some simple ones were picked first to show progress, as well as the most risky ones (to reduce risk). As we mentioned in the communication section, a meeting is held at the beginning of the iteration in order to produce a straw man list of line items, prioritise them, and assign them to team members. At the same day of that meeting, the customer is contacted by the phone to agree on the prioritised requirements. A line item is a term the team uses to refer to requirements description.

After the first two months, the customers become more forceful and started asking for more features. In addition, team members are expecting to have firmer requirements in the future.

- Documentation

Important information can be transferred to the project wiki along with meetings minutes. Keeping the project wiki under control can be problematic, and after a while, it loses structure. Although the team keeps a history of the development (change logs, wikis) they do not have any “static” documents. Even with traditional approaches, however, documentation can easily get out of date.

The architecture is documented as power point slides written by the architect and reviewed by the team. These slides include high-level decisions and some detailed description. The team is using class diagrams, package diagrams, and sequence diagram and there is nothing between these and the code apart from Java doc. An up to date list of features is available for users including how to use them. Also, the team takes photographs of the whiteboards. At the end of each iteration, the team lead writes a report to the senior management. Also, the off-site test team wrote a formal document for test case writing guidelines (around 100 pages).

- Agile Practices

Test-driven development was in use and it worked well for simple tasks. However, it has been helpful to have specialist testers as well as developers in the team, who have the skills to oversee all testing. In addition, the team used pair programming for new team members to help integrate them in the team and assess their competence. Some interviewee found that pair programming could be embarrassing and it needs a special work environment. However, the team members do occasional shared defect debugging. One interviewee indicated that someone cannot force pair programming as it works only with someone you know quite well, "*right combination of people may share a keyboard*".

As mentioned before the team is using stand up Scrum meeting and the team members are happy with it, as it helps having the shared technical understanding. In addition, the team uses continuous integration, which is becoming more difficult as the project is growing. Finally, refactoring is in use and it happens all the way through.

- Management Issues

Project Success: Measuring an agile project will be the same as any other project. The purpose is to fit the expectations of the stakeholders and to meet the actual requirements. One suggestion came from the team, which is to define, at the end of each iteration, a set of measurable functional or non-functional tests or assertions that would pass.

Lessons Learned: The team expressed that lessons learned are better than a formal development manual and important to reflect how things went. In addition, the off-site test team has a lessons learned session every month.

5.4.2 *The People*

- The Team

Size of the Team: The team has 16 members of which 12 are on-site and four off-site. Out of the 16 people, 15 are writing code including the senior team, (12 coders and 3 testers) and one performance person. One developer was off-site in Scotland. The team had one architect, one team lead and two sub teams each had a lead who rotates, one sub team is responsible for the core functionality and the other is responsible for everything else including the add-ons. The team size did not change over the period we studied the project.

The testing team is off-site with a test lead on site. The team lead indicated that he prefers to meet off-site teams members at least once, then they can use chat, emails, wiki, conference calls, webcams, or even recording and transcription phone calls if there is a need.

The test team is located in India with one remote member in Scotland. When interviewing the team lead in December 2008 he stated that having an off-site developer was slowing the team so he moved on-site which worked better. Only one tester remains off-site instead of three as more on-site testing was needed.

The team was put together for this project, two new people were hired, the rest were from the company, the majority already knew each other. Senior management chose the team lead and the test lead to interview people and put the team together. During the interviews, the focus was on finding people that have the ability to deliver, self-directed people, and who will be able to work in a team. In addition, they included negotiation skills in the employing test. Interestingly, the team was put together before deciding on agile, and some people changed their minds and decided not to join when they knew about this new way of working.

Developing Team Skills: In order to learn new techniques, the team had the "chalk and talk sessions" where a topic was divided between the team and each group gives around 30 minutes talk on what they have studied, so everybody will learn about the subject without spending too much time. Sometimes, this slot was used to exchange experience about technical issues, such as debug facilities, or coding standards. In addition, it was a good opportunity for exchanging feedback. During the last interview, the team lead stated that the team also has an "agile education".

Communication within the Team: Communication within agile teams plays an important role. In project A the team used different ways of communication including meetings, whiteboards to record task lists and current progress and to tick complete tasks, however they could not leave information on the whiteboard overnight for security reasons. They also used wikis, meetings minutes, presentations, chalk and talk sessions, and informal discussions in the working area or over lunch.

Seating Plan: The team is seated in an open plan area consists of three bays of four people each. The team lead sits in one of these bays. The layout seems to work quite well; however, as they are sharing the area with other teams, this affects the communication as they have to respect other teams who may use different kind of methods that does not involve high level of communication and interaction between the team members. The seating plan did not change over the time; the team lead stated that the atmosphere has improved over time as well as the communication level.

Shared Understanding: The interviewees indicated that with agile it is easy to have the shared understanding. The architect presents the architecture to the whole team, so everybody will understand what is going on, also, he spends time explaining it to the developers if something is not clear. As a result, everybody should be able to present the architecture.

Ownership: Everyone in the team is writing code with no strict code ownership, only nominal owner, or originator. Yet, during code reviews, they do not do any changes to the code, only suggestions to improve the code and correcting spelling mistakes. The same applies on line items (requirements), as everybody can access and change them at anytime.

Morale of the Team: The team is motivated and hard working, and many comments showed that the team is happy and satisfied with this new way of working. During the last interview with the team lead he reported that "*people are so happy, nobody has asked to leave, we are considered as the happiest team in department, and we even have social events together*". According to the team lead, as time passes the team is more self-directed than the start of the project and even than waterfall days. The team is not waiting for directions form the team lead and things happen automatically. The team lead stated, "*the right people form the right team*".

Roles: The team lead is also a technical lead, so he watches the process day by day and has a full understanding of the process. In addition, he monitors the defects and provides direction and guidance to the whole team. The architect facilitates high-level thinking for requirements design and review. Developers write code and unit test it, and regard testing as part of their role. Testers write functional tests, then the team lead selects the tests and adds them to the build. Since the project became part of a bigger one, the team had service responsibility. Every iteration two people have service as their responsibility, they monitor and respond to forums, make code changes and own the problem. The overall leadership and control for the big project is located in the USA where there is a program director, who attends the Scrum of Scrums, a chief architect and a product manager who have the business view. There is no product owner yet.

Previous experience: The team members' previous experiences are summarised in table 5-3:

Architect	Experience with traditional approaches and a previous experience with a project that used some agile practices
Developer	Previous experience with waterfall, no previous agile experience
Test team lead/tester	Never seen agile to delivery yet, experience in FORTRAN
Team lead	Was involved in a project that used XP as in Beck's book, and from that some things worked and some did not

Table 5-3 Previous experience of Team A

- Customer

Communication with the Customer: At the time of the interviews, the project had an internal customer who was using the delivery of each iteration. At the early stage of the project the customers were not involved in iteration planning, they were happy with what they were getting. Developers expressed that the response to customer requests is very good with agile methods; however, it depends on good customers as in some cases when extra effort is needed to obtain some feedback. The customer provides priorities weekly by phone on the day just before the iteration planning meeting, also they may email their requests to the team members directly. As expected, the customer's demands and requests have increased throughout the project. In 2009, the team had three new internal IBM customers with different requirements and priorities.

Delivery to the Customer: The project started with 2 weeks internal delivery at the end of the iteration, all were on time. Then they moved to weekly (mid iteration) delivery. In the future, the deliveries will be available on demand.

5.4.3 *Quality and Quality Assurance*

Assigning an iteration to improve the quality of the code is an effective practice; in addition, the team is using code reviews. The team expressed that these stabilization and consolidation iterations are important to increase the quality of the code.

There is a quality plan for the big project as a whole to cover what the team is going to do in terms of performance, features, scalability, and defects. This plan must be approved by the quality assurance team and signed off by the director of development before the whole product can be shipped. The quality plan is inflexible and cannot be changed later. The team lead stated that he is looking forward an agile quality plan, which will focus on how well they are applying the process, and on feedback from customer rather than defect count. In addition, this plan has to be more flexible in terms of features. The current quality met the targets for versions 1.0 and 1.1.

- Good Enough

The team lead's drive is to do the right thing at the time based on the current knowledge; the more senior members of the team struggled the most with this change from more traditional processes. Therefore, at the early stage of the project, the team produced alpha code to get quick feedback from the customer and now they can move on as requirements have evolved.

- Defects

"*Agile can achieve high quality*", one interviewee stated, although high quality does not necessary mean zero defects. Still, after six months have passed since the project started, only 30 defects were reported, some are missing features others are internal customer's defects. The project has the same number of defects comparing to previous projects the interviewees worked on. Defect prediction is not easy with agile methods, and current IBM specific techniques that predict cost of defects and time needed for system test are in use although they are not designed for agile software development.

Sometimes fixing a defect may take priority over agreed goals. At later stage of the project, according to the team lead during the interview in December 2008, many problems were fixed without a bug report. The team managed to stay within the bug prediction, which is estimated by IBM. They found that agile software development helps finding problems early, then it is a business decision to fix them or continue adding functions.

- Code Reviews

During the tidy up iteration, the team did code reviews in pairs, each pair worked for two hours individually, and then 45 minutes together. However, they did not change any code, yet they provided suggestions and reported spelling mistakes. Team members stated that code reviews were important to highlight quality gaps that could be solved during the tidy up iteration or could be added to the line items. The use of code reviews has dropped over time but it is coming back as required.

- Quality Assurance

The team lead stated that agile quality assurance is different from traditional development quality assurance as in agile assurance it should be more flexible and adaptable.

- Customer Satisfaction:

The team are doing what the customers want, deliver something to the customers to use and keep business partners happy. However, there are some areas that are not tested yet and they only have 10s of customers not 100s. The team lead expressed that agile methods give a good way to assess customer satisfaction, he stated, "*in waterfall you may find lots of bugs that do not matter to customers*". The project achieved better customer satisfaction compared to other projects. The project uses IBM standard practice to measure customer satisfaction, which involves collecting feedback through customer satisfaction surveys. The team has done these reviews twice (as of December 2008). In addition, forums are used; and most questions are indicating high usage rather than reporting problems

Another customer satisfaction measure is the beta programme which can be used over a release (2 months before the release). In this programme, the team signs up customers who agree to be interviewed in return for extra support. The interviews focus on what expectation the customers have, and what are their thoughts about the project. The interviews identify use cases where customers are not satisfied. The results of Beta programme for release 1.0 showed that customers were using different parts of the product than what is expected which identified gaps, and mismatches between requirements. These results became the focus of the next release (1.1). The team lead stated that this was important, as they were able to spot the problems early.

- Quality measures

The team lead does not prefer formal management metrics as they need to be firmly accurate and do not work well with 3 hours build cycle, instead informal mechanisms work much better. The metrics in use for project A are LOC (Line of Code), number of bugs, and code coverage (79% currently). In December 2008 the team members were applying reflection measures where they measure themselves against practices anonymously based on a self evaluation using 0-10 scale to answer questions about how much a team member is using different practices.

5.4.4 *Project A Summary*

The data analysis gave us an understanding of the adoption of agile software development and how it is impacting the different aspect of quality within a traditional organisation such as IBM. The project did not follow any agile method at the beginning but as the time passes, the used process was influenced more and more with Scrum. The project started with 16 members and the team size was stable throughout the period of the interviews. The team followed 2 weeks iterations, and used iteration planning, TDD, refactoring and continuous integration. Team members were happy and motivated which played a big role in the success of the project. Having off-site members did not work very well so they moved to join the rest of the team on-site. The company culture affected the development in a number of ways including delays in early deliveries because of legal issues, the team was unable of keeping whiteboards overnight, and the quality plan was not flexible enough to fit the agile way of working. However, we argue that the company culture had a more positive impact on the project than negative. This is because the project followed the existing good practices in IBM such as the emphasis on measurements. In addition, although quality plans were inflexible they worked well and they are on the way of producing an agile quality plan. The project delivered on time, defects count was as predicted, and was similar to previous projects, and customer satisfaction was improved.

5.5 The Results: Project B

The purpose of the project B is to produce an integration framework that can simplify existing IT resources and helps deliver a service oriented architecture infrastructure. The project was a challenge as it was required to work on different platforms, a range of databases and it is multilingual. The project started in October 2005, development started in December 2005 as it took 2 months to build the team. The first release was out after 10 months which is quicker than other products within the company. Since then the project has delivered six releases (as in June 2009) and during the last interview with the project manager, the seventh release was taking off. Figure 5-2 shows the progress of project B. The total number of interviews conducted with team B is seven with four team members a tester, an architect, a developer who was a team lead at the time of the interview and the project manager. Five interviews were conducted between January and November 2007. Two additional interviews with the project manager were conducted in October 2008 and June 2009.

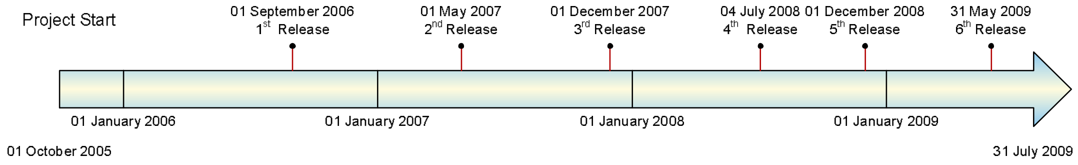


Figure 5-2 Project B progress

5.5.1 *The Process*

The team did not follow any standard agile method; instead, they applied agile practices and principles so they had their own agile development method. The process matured, and changes happened over the period the project was studied.

- Iteration Planning

Each iteration is delivered on a four weeks basis. As each iteration starts, the project manager spends two and a half days for iteration planning as the following:

- On the first day of the iteration, the project manager holds a two hours meeting with the whole team.

- Then each development team leader has a meeting with his or her team on the same day to make sure that they understand everything and to see if they have any questions.
- On the second day of the iteration there is two hours meeting with development teams leaders and the architects
- On the third day, the project manager meets the architects to agree the features list and who's doing what.

The iteration plan changed over the time. During the first four releases, up to two weeks were added to the initial 4 weeks iteration for testing and fixing the code. At the same time, at the end of the fourth week, the next iteration would start, so the two iterations would overlap as we can see in figure 5-3. At the end of week four, the next iteration started and a D-Cut would occur in the current iteration but the testers continued testing the code. The team expressed that defining what they are doing iteration by iteration and building up the code base and test cases has been a good practice.

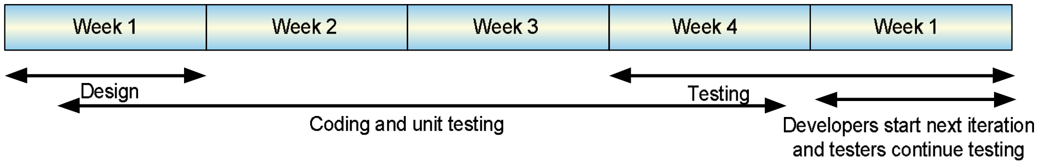


Figure 5-3 Project B iteration planning

As testers continued testing the previous iteration code, developers, however, were under pressure as they start a new iteration but had to respond to testers queries. This approach has changed in release 5 as the one week (or so) overlap has been stopped, and the last week of the iteration will focus on closing the defects that built up during the first 3 weeks of the iteration as shown in figure 5-4. The main reason for this change is that developers did not have enough time to fix defects. In the new approach, they have the last week of the iteration to do so and they are no longer fixing defects and starting the next iteration at the same time. However, development time is reduced to 3 weeks so there is less development time, which means fewer capabilities. In later interviews with the project manager, he stated that this approach is working well so far.

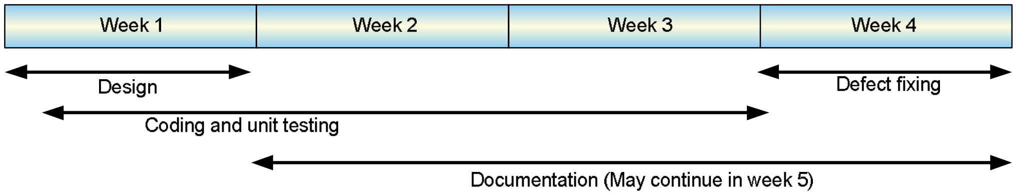


Figure 5-4 Project B Iteration Planning with changes

- Meetings

The iteration starts with two hours meeting for the whole team. After this meeting, each development team lead has a meeting with his or her team on the same day to make sure that they understand everything and to see if there are any questions. On the second day of the iteration, the project manager, the development team leads, and the architects meet for two hours. In the third day, the project manager meets with the architects to agree the feature list and who's doing what.

The senior management team (project manager, team leads, and architects) has three types of meetings during the iteration. These meetings include a weekly meeting to discuss architecture reports, a daily meeting for an hour to focus on the external view and to decide on high-level priorities, and a daily meeting for an hour to discuss architecture, technical decisions, and priorities. This last meeting is open to all team members and its focus is on what should be fixed and which requirements to be deferred to the next iteration or the next release. The iteration meetings are summarised in table 5-4

The project manager is considering introducing or trying a Scrum stand-up meeting as it is easy for implementation to lose sight of other developer's needs, however different people talk for longer or for shorter time, and the question is: is it possible to do a 15 minutes stand up with 30+ people?

Duration	When	Whom	Purpose
2 hours	1st day of iteration	Whole team	General overview of the iteration
-	2nd day of iteration	Each sub-team	Make sure that everybody understands everything and answer questions
-	3rd day of iteration	Project manager & architects	Agree on the list of features and assign tasks
1 hour	Daily	Senior management team	Focus on external view, select customers requests, high level priorities
-	Weekly	Project manager & architects	Go through architecture reports
1 hour	Daily	Senior management team & all welcome	Architecture technical design and priorities, defect prioritizing
-	Daily walk in work area	Project manager	Checking on work status

Table 5-4 Meetings in project B

- Tidy Up Iteration

The team is using tidy up iterations after releases, where they focus on refactoring, defect fixing or testing.

- Testing, Automated Testing and Relation between Development and Testing Team

The project manager indicated that the main success factor in the project is the automated testing as it gives good control, and it is a good measure of the quality of the service. All tests happen overnight and they are all automated including the builds. As mentioned before, at the end of week 4, the next iteration starts and a code cut off occurs in the current iteration, which enters the fifth week where the testers start testing the code written by developers. Test cases and code are written in parallel. Test team writes functional verification tests and system verification tests, where most developers write Junit tests.

In the first release, the team had 1425 tests per environment with 10 environments for that release. In the second release, they had 2000 tests per environment and two more environments. This needed high capacity hardware and this was available in the company so no additional cost was added. However, the heterogeneous platforms increased the difficulty of the testing.

The test team structure mirrored the development team division; this makes communication easier as testers contact developers from the same area when they have an issue. The test team do not fix bugs except in their tests, so when a test fails (with a critical error) the testers go back to developers. There were no cases where bugs led to friction. Testers attend design and brainstorming sessions to understand the design and to suggest testability improvements.

- Requirements

The project manager stated that initially they were prepared to be flexible with requirements. They have new items every month; so they commit to some requirements, and can always change because of customer or sales people requests. The project manager pointed out that requirements management in agile software development is very critical, in order to decide what is important at the time. Requirements' prioritizing happens during the management daily meeting with focus on external view and customer requests. On the third day of the iteration, the project manager agrees on the feature list proposed by architecture and tasks will be assigned loosely.

Customers provide feedback and suggest new requirements along the project. In order to decide on an item, they should ask why it is needed, and if any customer is asking for it. One issue was which customer they should listen to, and the target is to meet the needs of as much customers as possible, so they tried not to be influenced by the size of the customer.

Although the requirements were flexible at the early stages of the project, as the project progressed, the requirements starts to have different level of detail and become more rigid (in release 3). Moreover, during the last interview with the project manager in June 2009, he stated that the requirements for an iteration are fixed now; also, it is difficult to change the agreed requirements for a release.

After release 4 the team started using user stories (Beck et al. 2004) to replace features. The user story structure in use is:

As a <<role>>, I want to be able to <<function>> such that I can <<business value>>

The external requirements are summarised and mapped to the user stories which are available internally. Customers have been encouraged to submit their requirements in user story format. The team is using a bug tracking tool to track user stories, code, documents and tests, and when all three are closed the user story is closed. The challenge now is how to handle user stories across iterations, in other words can an epic user story be broken into smaller user stories (one per iteration)?

- Documentation

A number of approaches are in use for documentation including java doc. However, the team has very little architecture documentation. This works but the architects are not happy with it. The architects provide weekly reports; also, they deliver models using word documents and PowerPoint presentations. In order to improve documentation, the team tried several approaches such as team rooms to document a release or an iteration, daily meetings with the team, and tracking issues against decisions.

During the interview with the project manager conducted in October 2008, he mentioned that the team would spend extra time in the next iteration to review last iterations documentation. This will require creating time for developers so they can focus on documentation.

- Agile Practices

As mentioned before the team did not follow a particular agile method. They used pair programming at the beginning of each iteration, also refactoring was used during the tidy up iteration. After the first release, the team had one iteration to refactor the code and the architecture. They found that refactoring can require a lot of effort and the benefit is not always obvious. In addition, it was hard to judge how much refactoring is enough or when to stop. Besides, they needed to decide between refactoring and functionality. Refactoring can be a theme for an iteration or a release, however, the team is concern because they are under pressure to get functions out, and they need more time for refactoring. In releases 4 the team did not do refactoring due to the very short release schedule. In Release 5 the team is working on changing the data store which is a kind of refactoring or restructuring.

- Project Management

Project Success: The project is a great success. The releases are happening on time so far, and the first release was out after 10 months where with other projects within the organisation, the first release usually takes at least 18 months. The quality is very good comparing to other projects, as the defect rate is incredibly low according to the team members.

Lessons Learned: At the end of the each release, the team learned what they can do better, and that agile software development provides good things and bad things. The agile champion, who is a lab level external person, helps the team to understand the process and to reflect on what are the weaknesses and what are the good practices. He may attend meetings, meet the team, and work with them.

Business Value: The team stated that the project is an example of how agile software development is a very good way to deliver real business value to the market more quickly and therefore ensure stronger earnings. Agile software development also helps with the marketing and political issues that arise when trying to quickly fill a gap in the product portfolio.

Iteration and Release Focus: In this project, releases work on themes. Refactoring can be a theme as any other quality gate such as scalability, performance, portability, and extensibility. However, it is difficult how to agree on high-level views and themes for each release as this depends on market place, customer, development team, and support service organisation.

The focus of the first release was on functionality. In the second release, the focus was on making the product more robust. The third release was about functional features, consumability, and graphical user interface. Thus, as the project processed the iterations provide capabilities rather than individual use case. In addition, the approach has changed over releases. In release 3, the team had very tight timescale and more rigid requirements. However, the fact that each iteration had to deliver something new did not change. For the first release, it was about what the system does, whereas the second made it robust and the third made it consumable.

5.5.2 *The People*

- The Team

Development Team Organisation, Size of the Team, Roles, Development Team Skills, and Developing the Skills of the Team: The project started with 35 members, and the team has grown to 55 members as in October 2008. The team is organised as the following: 17 developers (increased to 24), 20 testers, 2 architects, 2 project managers, and 7 off-site (5 in China and 2 in the US). The team is divided into three groups: UI team, API team and content (database) team, each group has a team lead. The off-site team is writing code to integrate the system with other products in the company and they are using a different process life cycle. Each team lead rotates every month and developers do many rotations so everybody will have experience in different roles. During the last interview in June 2009, the project manager stated that two additional off-site teams have joined the team in May 2009 (10 people in India, and 12 in China) which increased the team size to 77 people.

Regarding people experience, developers stated that iterative development requires experienced people, with high level of communication skills, who are open to change and will not need to be told what they have to do. In addition, they need to have multi-skills, as they are required to do system verification tests as well as coding. This applies to the testers as well, as they are required to have coding skills for automated testing. To improve their capabilities, the team members have an informal reading group, and most team members went to functional testing courses.

Communication within the Team: Brainstorming is used to make a start in difficult areas. The team likes participating in architecture open door sessions. Developers can email architects queries and suggestions and this makes communication quicker and easier.

Seating Plan: All team members are located in the same area, though it is not an open plan area but small offices of two people where a tester and a developer often share the same office.

Share Understanding: The shared understanding was clear during the interviews as the subjects demonstrated the ability to describe the process of working; also all subjects had the same overview picture. An interviewee pointed out that this was very important: "*If everyone understands what is going on, this is what really matters*". In case any problem has arisen, the project manager will call the whole team for a meeting.

Ownership: When asking about code ownership, one answer was "*it is ok to change others code, even testers and developers can do the change and people are comfortable about it*". The same applies to line items where anyone in the team can open one at any time and they can make changes. However, this can cause problems, as anyone who can write to the project will be able to write to any file in the project.

Morale of the Team: when the old iteration plan was in place, developers were under pressure as they started a new iteration and still needed to respond to testers who were testing the previous iteration. This has changed since release 5 to reduce pressure on developers. Team members expressed that they do not get bored with agile software development as constant changing can be creative: in every iteration they have to write new code as well as maintain existing code.

Three of the interviewees were satisfied with the new approach as it is team oriented, with direct feedback from the customer so they can see the value of their work quickly. Furthermore, they have more input to the design and more influence on the architecture. Besides, they are having more fun. This satisfaction was expressed in comments such as "*The team is like a democracy", "current project has more interaction", "in the waterfall days we did not talk to anybody", and "In agile five minutes discussion can solve the problem*". On the other hand, one interviewee had concerns about things going very fast, pressure to get functions out and building new code on unstable code. The project manager is happy with the process, as the developers are sharing thoughts and delivering working code at the end of iteration and give each other demos where they share reflections.

Previous Experience: Table 5-5 summarises team B previous experience

Project manager	Has been involved with many traditional projects, this project is his first experience with agile software development
Developer (team lead)	Previous experience with waterfall/incremental approaches, this project is his first agile experience
Tester	Previous experience with non-iterative projects that did not release, this is his first agile experience
Architect	Hardware background, this project is his first agile experience

Table 5-5 Previous experience of Team B

- Customer (Communication and Delivery)

The first delivery was in iteration 6 (after 6 months). Since then the team delivered every month. According to the project manager, this was for legal reasons. It is very important for the organisation to assure the provenance of all code used in the project and to consider and develop patent submissions. Therefore, the first delivery to the customers required considerable amount of legal paper work. As the customers are expecting something they can use with each delivery, it is important to understand how they are going to use the capabilities they provide.

The customers can install an early version of the product called early access program (EAP) though the website and they can send feedback and questions using a forum. Each developer has access to this forum, so they can read the customers' feedback and questions, and give answers and support. Also, there is an external news group to add comments and questions, and this group can be seen by all customers.

During the interview with the project manager conducted in October 2008, he stated that the number of customers and users increased. However, customers were focusing on using the third release's features, so the fourth release's EAP was not well subscribed, hence less feedback and less agile. The pace of releases maybe therefore a problem as customers were not able to keep up with the releases, however, new customers will always have questions about current release. During the last interview with the project manager in June 2009, he stated that customer interests and demands have increased.

5.5.3 *Quality*

- Quality of Code

The project manager stated that testing is the main factor to assure the quality of the product and iterative development is good to focus on one aspect of quality in each iteration or in each release. For example, iteration or release focus can be on refactoring, performance, portability, or extensibility. During the last interview, the project manager stated that the product is stable, reliable, and available.

One interviewee felt that the quality is slightly less than other products, whereas another two stated that it is no worse than in other products. In this project the team did not use code reviews, but developers expressed that they would like to do code reviews as they have used them before and they were effective.

Making the product good enough was the mantra, “once it worked, move on”, where traditionally a 100% working solution was required. This is a very good way to satisfy the customer as the product is providing what is needed from the customer point of view. Therefore, the team waits for the customer to notice and demand changes and improvements.

In the interview with the project manager in October 2008, he still believes that the quality of the produced code is better than code produced using traditional approaches as they have a working product every month and it is being used by customers.

- Defects

The team members stated that defect rate is very low comparing to other products within the organisation and the customers are satisfied. An interesting comment from a tester was that although the number of reported bugs is bigger in agile projects, they are minor and easier to fix than in traditional projects.

Testing is a crucial factor: when a test fails the bug tracking system will raise a defect. Defects are prioritized in the senior management lead meeting, where it is decided which defects will be fixed and which will be deferred to the next release or the next iteration. The team gives a lot of time and effort to review new defects and to set priorities.

Customers mostly report defects on past releases, and they can report defects on iterations via emails or forums, also defects found internally from the team and internal users. Release 4 had some low priority defects carried forward into release 5 so the project manager needed to monitor how they are fixed. The goal is to reduce the defects in future releases.

- Customer Satisfaction

The team members were not sure if using defects metrics is the best way to measure quality. They thought that a better idea is to look at the feedback, talk to the customers, and collect defects from them. In addition, they examined the feedback from the service organisation. The architect mentioned that this feedback is subjective and qualitative, and needs to be interpreted into a measure of customer satisfaction.

The team used two measures for customer satisfaction: the number of problem management reports (PMR) from customers to the service organisation within the company, and the number of problem analysis reports (PAR) from the service organisation to developers. In post release 4, the number of customers' reports is low and mostly capabilities not coding problems.

- Quality Measures

The team is using a number of defects and quality measures, such as defects raise rate, defect severity, defect fix rate, number of open and deferred defects, code coverage, and defect cover (for functions). In addition, the project has logs for tests and bug reports where existing, management level defect report and graphically open/closed defect reports.

5.5.4 *Project B Summary*

Studying Project B and keeping up with the project changes over three years gave us not only a good understanding of the adoption of agile software development and how it is related to the different aspect of quality, but it helped us understand how agile adoption evolved. The team followed 4 weeks iterations, used iteration planning, pair programming at some stages and also refactoring. There were no problems reported regarding off-site members as there were in project A. The company culture affected the development in a number of ways including delays in the first delivery because of the legal issues, and some times some features were needed as part of the company policy, which increased the load on the team. However, we argue that the company culture had a more positive impact on the project than negative. This is because the project followed the existing good practices in IBM and the measuring culture, including measuring customer satisfaction, and collecting defects and quality measures. With six releases in the market all on time, the project is a success as it is delivering high quality code with low defect rate and the customers are satisfied. In the early stages, it was not clear if the low defect rate was because of the new process or because the project is not mature or heavily used yet. After release 4, however, and with the increase in customers number it is more certain that the used approach has a measurable and positive impact on quality.

The project did not follow any particular agile method and this did not change over the project, however new practices were introduced such as using user stories. More importantly, the iteration plan changed after release 4 to reduce the pressure on developers. The other change is the project team size, which grew to 77 members. One of the important changes we observed and was confirmed by the project manager is the project agility. This was demonstrated with the requirements, although they were flexible at the beginning of the project, this however changed as the project progressed and matured and this meant that customer demands has to wait until the following iteration or even release.

5.6 Comparison

The empirical study results agreed with our previous overall hypothesis that agile methods have improved product quality. They showed that both projects were successful with multiple releases, the product quality in terms of defect rate seems to be as good as other projects in the same organisations (if not better), the time to release is reduced and the differences between the two projects in terms of communication, the iteration length and approach to quality, may have resulted from them having teams of different sizes.

An interesting variable is the team size. We can argue that the team size affected the level of communication in the team. For example in team A, there are more channels of communication within the team. In addition, the whole team is involved in most of the meetings and this is understandable for a team of 12 (on-site). On the other hand, team B has more meetings that involve high level of leadership (project managers, architects, and teams' leads); however, this did not affect the shared understanding and the ownership within the team. The other variable is the iteration length, for the first team it is 2 weeks where it is 4 weeks for the second team with up to 2 weeks of overlap. The question here is do we need longer iterations for bigger teams?

5.7 What is good and what is bad about Agile Software Development?

5.7.1 *The Bad*

During the interview, a few interviewees expressed worries about some aspects of applying agile approaches:

1. Agile will not work for big teams as things may go out of control

2. Agile may require good and collaborative customers as it will be more difficult to get feedback from some customers
3. Possible problems with extending projects as less time was spent on testing, design, architecture and documentations
4. Although agile seems to be good for new products, there are some concerns about using agile for mature products.
5. The focus on what is "good enough" was an issue.
6. Architecture principles can be lost

5.7.2 *The Good*

Although some interviewee expressed some doubts about agile, most of them gave positive comments:

1. Agile gives more control in reality even if it gives illusion of less control over the process
2. Agile methods look better so far, particularly as they can deliver quicker to the market place than any other development approach, but need to wait for final results to make a final judgement
3. Everybody will get the overview picture and the shared understanding
4. Response to customer requirements, changes and feedback is very good, and it is better than in traditional approaches
5. High level of communication and interact within the team
6. The team is very happy, motivated and having fun
7. Agile methods are good for new products
8. With agile the team has more incentive to find out the answer quickly
9. Less useless documentation
10. The functionality of the resulting product is quite good
11. Sales people see that the early delivery is a strong selling point
12. Rotating roles helps people towards promotions where specific responsibilities are required
13. Save testers' time as they can find if their tests make sense at an early stage.

5.8 Compare Agile Approaches with Traditional Approaches

Most of the subjects had experience in traditional approaches before working with agile software development. The differences they expressed are summarised in table 5-6.

Traditional Approaches	Agile Approaches
Give illusion of strong control	Give good control over the process
It takes longer to find out about wrong paths	It takes shorter to find out that a component is not going to do the job and it is not too late to start all over again
Developers do their own thing waiting for architecture to be ready, this could take a year	The whole team is working at the same time on the same iteration. Good coordination between team members
Too slow to get fixes to the users	Provide quick responds to user feedback
Change requirements is difficult in later stages of the project	Can respond to customer requests and changes easier
More time is spent on design so the product will be more maintainable. The "what ifs" arise earlier	There is no time for the what ifs
No communication within the team, novices stay in their rooms and try to understand things	High level of communication and interaction, reading groups, meetings
Restricted access to architecture	The whole team influences and understands the architecture. Everybody will be able to do a design presentation
Documents and review meetings are needed to solve an issue	5 minutes discussion may solve the problem
Everything is up front, everything is big before you start	The focus is on whether customer requirements are met in the current iteration
Normal releases take 18 months	After 10 months the first release was out

Table 5-6 Agile approaches vs. traditional approaches

5.9 Validity Issues

The two projects were developed within one organisation. Therefore, it could be generalised to cover other projects within the same organisation or to similar organisations. However, in order to generalise the results on other organisations we need to expand our study to include projects from different companies. In addition, the collected data was based on the experience of the interviewees and their opinions. On the other hand, the study was done with real software development on two projects of a significant size and duration. Regarding the validly of the collected data, we interviewed four people from each team and the participants mostly agreed with each other. In addition, we had two researchers taking notes, which gave our data higher level of quality and accuracy, however only one researcher analysed the data. It was difficult to obtain more sources of data because of the company restrictions, for the same reason it was not possible to audio record the interviews.

5.10 Generated Hypotheses (Version I)

Analysing the interviews yields in the themes presented in the previous section, which in turn helped extracting statements that will be used as hypotheses. The generated hypotheses are organised in four categories: customer satisfaction, product quality, people quality, and process quality. These hypotheses are mainly based on the interviews conducted from November 2006 to December 2007.

5.10.1 *Customer Satisfaction Hypotheses*

1. Using agile methods improves customer satisfaction
2. Customer involvement/demands/requests increase throughout the project
3. The customer satisfaction has the same level throughout the project
4. Consumability increases when using agile methods
5. Response to customer requests is good when using agile methods

5.10.2 *Product Quality Hypotheses*

6. Using agile methods can achieve high levels of reliability, availability and serviceability
7. When using agile methods, testers receive more minor bugs comparing to traditional approaches where the bugs are fewer but more critical
8. Automated tests can assure high quality code
9. When testers and developers work in parallel, testing will be more effective.
10. The code developed using agile methods has the same (if not lower) defect rate than traditional methods
11. The code developed using agile methods is less maintainable
12. The quality of the code increases as the number of iterations increases
13. Assigning an iteration to tidy up the code improves the quality of the software in terms of defects and maintainability

14. Code reviews can help improving product quality in agile software development
15. Refactoring can help improving product quality in agile software development

5.10.3 *People Quality Hypotheses*

16. Agile software development requires people with high level of communication skills
17. Short iterations have good influence on the morale of the team
18. Iterative development requires developers with high level of experience
19. Integrating new team members is harder with agile methods
20. The smaller the team the higher the communication level between the team members
21. People are happier and more motivated when using agile methods.

5.10.4 *Process Quality Hypotheses*

22. When using agile methods testing is the responsibility of all team members
23. In agile software development governance increases when the team is larger
24. Effectiveness of communication decreases when the team is larger
25. Communication level within the team is higher in agile development than in traditional approaches
26. Agile adoption goes in stages and it improves over iterations, releases and projects
27. In agile development the process matures throughout the project
28. Each release should have a clear focus on one aspect of quality
29. Prioritizing defects is important in agile development
30. Agile methods are more suitable for brand new products
31. In agile development, longer iterations are needed when the team is larger

5.11 Hypotheses Review (Version II)

The generated hypotheses were presented in the Agile Conference in August 2008 during the research in progress workshop. It was a good opportunity to have the hypotheses reviewed and evaluated by the agile conference attendance. The workshop had about 200 people working in groups or 6 to 8. Printed copies of the hypotheses were distributed to the groups. I asked the attendees to evaluate the hypotheses from 1 to 5 according to how much they think the hypotheses are true or interesting to be investigated further. The reason behind this is we have got a long list of hypotheses based on two projects. This survey collected 41 responses (as example of the collected data can be found in Appendix D)

The responses were coded as the following:

1= not interesting

2 = slightly interesting

3=maybe interesting

4=interesting

5=very interesting

In addition, we have some comments and suggestions from the respondents. Some people tend to put number 5 next to some hypotheses without ranking the rest. Others did not give a rank but they gave a comment on the hypotheses such as, already tested, or not agile. We considered the valid percentage only, which is the rankings between 1 and 5. The frequencies and percentages of the responses are included in table D-2 in Appendix D. Table 5-7 presents the means of the valid responses.

In addition, after the first version of the hypotheses was generated, one interview with the team lead of project A, and two interviews with the project manager of project B were conducted. The two interviewees reviewed the initial hypotheses list. Considering their feedback and the conference attendees evaluation, we produced the revised list of hypotheses. We have included all hypotheses that got a mean of 3.5 out of 5 on the interesting scale used in the evaluation. The final list of hypotheses was restructured in three categories; stakeholders satisfaction, software quality and process quality hypotheses.

Hypothesis	Number of responses	Mean
H1	33	4.03
H2	31	3.52
H3	29	2.24
H4	27	3.44
H5	33	3.94
H6	34	3.62
H7	31	2.90
H8	34	3.53
H9	32	4.16
H10	30	3.86
H11	35	3.51
H12	27	3.37
H13	29	3.50
H14	31	3.30
H15	31	3.83
16	32	3.38
H17	31	3.45
H18	32	3.00
H19	30	2.70
H20	31	3.10
H21	31	3.50
H22	30	3.57
H23	28	3.30
H24	28	3.26
H25	27	3.56
H26	27	3.41
H27	27	3.25
H28	26	2.92
H29	28	3.43
H30	30	3.17
H31	31	2.87

Table 5-7 The hypotheses ratings means by the agile 2008 conference attendance

5.11.1 *Stakeholders Satisfaction Hypotheses*

H1: Agile software development can achieve customer satisfaction

H2: Customer involvement, demands, and requests increase throughout the project

H5: Agile software development assures quick and effective response to customer's requests

H21: Team members are happy and motivated when using agile software development

5.11.2 *Software Quality Hypotheses*

H6: Agile software development can produce good quality software

H8: Automated tests can assure high quality code

H9: In agile software development, testing is more effective when testers and developers are working in parallel

H10: The code developed using agile software development has the same (if not lower) defect rate than traditional methods

H11: The quality of the code increases as the number of iterations increases

H15: Refactoring can help improving product quality in agile software development

5.11.3 *Process Quality Hypotheses*

H22: Testing should be the responsibility of all team members in agile software development

H25: Communication level between different stakeholders is higher in agile software development than in traditional approaches

H26: Agile adoption goes in stages and it matures over iterations, releases, and projects

5.12 Summary

In this chapter, we presented the empirical study that included two case studies conducted using semi-structured interviews with two teams that were using agile methods within one organisation. Our data was analysed using the constant comparison method. The results were presented to illustrate how the teams adopted agile methods, the team organisation, the approach to quality, the communication within the team and the relation with the customer. The data analysis resulted in a list of 31 grounded hypotheses; this initial list was reviewed and reduced into a list of 13 hypotheses that were organised in three categories: stakeholders' satisfaction, software quality, and process quality.

Chapter 6

The Iteration Monitor

6.1 Introduction

The hypotheses generated in the previous chapter were based on interviewing eight members from two projects. Collecting more data and insights from the teams will help support the hypotheses and answer questions raised during the interviews analysis. The project manager in project B was happy to continue participating in the research. Therefore, an iteration monitor was designed to collect quantitative data about the project. This monitor was needed for two reasons; first to support the hypotheses and second, to identify issues and trends within the team in order to improve the process in the following iterations.

Project B was running since October 2005 and the team size was growing over time. Therefore, we introduced the iteration monitor to understand how things are changing over iterations. More importantly, what are the team members' opinions about the process, the quality of the product, and the support provided to the different stakeholders.

6.2 The Method

The iteration monitor is a web-based questionnaire developed by the author using PHP and MySQL. The questionnaire was uploaded on the author's website provided with a username and a password to assure that only the team members participate in the questionnaire. Before the data collection, the questionnaire was presented to the project manager for suggestions and improvements, then the author presented it to the entire team at the beginning of one of their meetings and their questions and concerns were answered. The data was collected over three iterations during the months of March, April, and May 2009 and the collected data was for the iterations that took place during February, March, and April. A link to the questionnaire was generated and sent to the project manager who distributed the link to the team members at the end of each iteration asking them to fill the questionnaire. Two weeks were given for collecting the data, then we analysed it and sent the results to the project manager so he can reflect to the next iteration as much as possible.

6.3 The Design

The iteration monitor started with asking the team members about their position during the last iteration. So we can understand how different people with different roles responded to the questions. The monitor included six sections: iteration questions, practices, communication, iteration focus, influence on prioritizing requirements, and improvements suggestions.

The iteration questions section included detailed questions about the iteration progress in different stages; during the iteration, at the end of the iteration and supporting previous iterations and releases. We asked the team to state how much they agree or disagree with a set of statements. Each statement was followed by a five-point Likert scale (Likert 1932) ranging from “strongly agree” through “neutral” to “strongly disagree”.

The second section of the questionnaire was the practices section. We listed a set of agile practices and asked the team how effective these practices were in case they were used during the iteration. Each practice was followed with a five-point scale where 5 is most effective and 1 is less effective, as well as a “not applicable” option.

The third section of the questionnaire was about the communication within the team. Each communication method was followed with a five-point scale ranging from “always” through “sometimes” to “never”.

One issue that was raised during the interviews is that the iteration focus is changing over the iterations. Therefore, it was important to ask about the iteration focus to see how it was changing over time. For this section, we had four areas of focus. Each area was followed with three-point scale; “too much”, “just right” and “too little”, as well as a “do not know” option.

The final section in the iteration monitor was about the influence on prioritizing requirements. As we interviewed people from different roles in the team, we noticed that different people had different influence on prioritizing requirement. In order to investigate this issue further, we added this section. We had a set of roles involved in the project; each role had a set of options similar to the iteration focus section. The iteration monitor concluded with suggestions for improvements from the team. The full questionnaire can be found in the Appendix E. We have got different response rate over the three iterations. For the first iteration we have got 24 responses, 13 for the second and only 10 for the third.

After the first iteration, we did not recollect data for the communication sections for iterations 2 and 3, as we were not interested in monitoring how the communication methods will change over the iterations, and hoping that reducing the size of the questionnaire will increase the response rate.

6.4 The Analysis

The data was collected and recoded using Excel and SPSS. SPSS was used as a tool for applying the analysis. First, because the software is provided by the University with introductory training, many books are available for self training, and most importantly it is a well respected tool among statisticians. In order to apply statistical methods on the current data we had to recode it into numbers using SPSS. This was done using a simple syntax that has to be applied on all columns to be recoded. The result is a new set of columns with coded data. The frequencies of the emerging data were compared against the original ones to make sure that the recoding was done correctly.

6.4.1 *Descriptive Statistics*

There are many ways of presenting univariate information about variables including frequencies, graphs, and statistical measures (Nardi 2002). For our data, we will present a frequency table that shows how each response was given by the respondent to each item; frequencies are useful when the variable has a limited number of values such as nominal or ordinal measures. It is less useful when an interval/ratio variable has many values.

In addition to the frequencies, the measure of central tendency provides a quick summary of where the responses are clustered. Depending on whether the variable is nominal, ordinal, or interval/ratio a mode, median or mean is used. For our data, we will use the mean, the sum of the values divided by the number of values. Although the mean is more suitable for interval/ratio, we will use it for our ordinal variables as our scale (Likert -scale) looks like equal appearing interval scales. In order to see how well the mean represents the data we will use the standard deviation. The standard deviation (s) is the square root of the variance which is the average error between the mean and the data points (Field 2005).

$$s = \sqrt{\frac{\Sigma(x_{i\_}\bar{x})^2}{N-1}}$$

Where: x_i is the data point for the ith position, $\bar{x}$ is the mean values and N is the total number of responses.

Small standard deviation relative to the value of the mean indicates that data points are close to the mean. A large standard deviation relative to the mean indicates that the data points are distant from the mean, which indicates that the mean is not an accurate representation of the data. We did not present the mean and standard deviation for some of the data as we did not have an ordinal data that can be treated as interval/ratio. However, when comparing the three iterations the means for the iteration focus and the influence on requirements prioritizing were calculated after considering the “do not know” responses as missing so the calculated means are meaningful.

Different ways were used to present the results depending on the collected data and the suitability of these ways for our results clear presentation. For the iteration questions, we used frequency tables and the measure of central tendency. For the remaining sections, we used the frequency tables and bar graphs as they gave a good quick look at the results. Also in some occasions, we used the filtering feature in Excel to find and work with a subset of the data.

6.4.2 *Statistical Testing*

In order to test the significance of the results we have two options; the first one is to apply the one sample t-test. The t-test investigates the significance of the difference between an assumed population mean and a sample mean. This test assumes normal distribution data. The other option is the one sample chi-square which is a non-pragmatic test. This test also called (goodness of fit) and it can be used to investigate the significance of the differences between observed data arranged in a number of classes and theoretically expected frequencies in the same number of classes (Kanji 2006). The assumptions for this test are; random sampling for the observed frequencies, the expected frequency in each class should be at least 5, and the observed and theoretical distributions should contain the same number of elements (Kanji 2006).

The data collected using the iteration monitor is not normally distributed and transformation did not correct the distribution. Therefore, the better option will be the one sample chi-square; however, the chi-square test requires the theoretical expected frequencies which have to be at least 5 for each class. In case we do not have theoretical expected values, they are calculated by default so the frequencies are equal for all options. This is not suitable for our data as the questions options are Likert scale, and it does not make sense to have equal frequencies for the available options. Therefore, chi-square test is not suitable for the data as it requires to have expected frequencies for each class that are less than 5. Unfortunately, the survey questions are not suitable for statistical testing; therefore, the descriptive statistics will be used to present the results.

6.4.3 *Relationships between Variables*

In order to study any existed relationship between the different variables, correlation was used to analyse the data. Correlation is a measure of the relationship between variables, however, in order to know what type of correlation is more appropriate, we need to explore the data. Screening our data showed that it was not normally distributed. Therefore, Spearman's correlation coefficient *rs* was used, this correlation is non-parametric and it can be used when the data is not normally distributed. The correlation coefficient has to lie between -1 and +1, where a coefficient of +1 indicates a perfect positive relationship and a coefficient of -.01 indicates a perfect negative relationship. A correlation coefficient value of ±.01 represents a small effect, ±.03 is a medium effect, and ±.05 is a large effect. When reporting correlation we have to say how big it is and what its significance value is. Primarily, the most important criterion is that the significant value is less than .05. However if the exact significant value is much lower then we can be much more confident about the strength of the experimentation effect. The letter *rs* represents the correlation type and the letter *p* represents the probability value for its significance. We have to be careful about correlation coefficients interpretation because they give no indication of the direction of causality (Field 2005).

6.4.4 *Comparing Means*

As we have mentioned before, the collected data is not-normally distributed, therefore the suitable statistical tests are the non-pragmatic tests. There is two non-pragmatic tests that can be useful for repeated surveys, if the groups we compare are independent then the suitable test will be the Kruskal-Wallis test (Kruskal et al. 1952) which assumes that each sample size should be >=5, though the samples do not need to be equal. The other test is the Friedman's ANOVA, which could be used to test the differences between several related groups. Now when conducting repeated surveys the samples are generally not overlapping, such that each sample is composed of entirely new individuals from the population (Firebaugh 1997). In our case, the data is not related for sure, which means that we did not collect the opinion of the same person over the three iterations, as we cannot identify our respondents. On the other hand, it is not possible to know whether the samples for the iteration monitor were entirely not overlapping, since personal information were not collected. So for our data there is no perfect solution, however the more suitable test for is the Kruskal-Wallis test, which is used for independent data. The Kruskal-Wallis test is based on ranked data. We start with ordering the collected data from lowest to highest ignoring the group they belong to and then we assign the lowest score a rank of 1, the next highest a rank of 2 and so on, then the data is assigned back to their groups and the ranks are added up for each group. These ranks are used to calculate the test statistic H using an equation calculated using SPSS which does all the previous work automatically. SPSS reports the test statistics which is labelled as chi-square because of its distribution, its degree of freedom df which is one less than the number of groups, and the significance. The significance value will be the crucial thing to look at; if it was less than .05 then the difference between the studied groups is significant. This test will be used to compare the results of the three iterations, to see if there is any significant difference between their results.

6.5 The Results: Iteration 1

For the first iteration we collected 24 responses from the team. The questionnaire was distributed to the whole team (55 people at the time of running the experiment). Table 6-1 shows the SPSS frequency table of the role variable as it was the first question in the monitor.

	Frequency	Percent	Valid Percent	Cumulative Percent
Developer	8	33.3	33.3	33.3
Information developer	2	8.3	8.3	41.7
Manager	1	4.2	4.2	45.8
Team lead	3	12.5	12.5	58.3
Tester	10	41.7	41.7	100.0
Total	24	100.0	100.0	

Table 6-1 The frequency table for the role variable

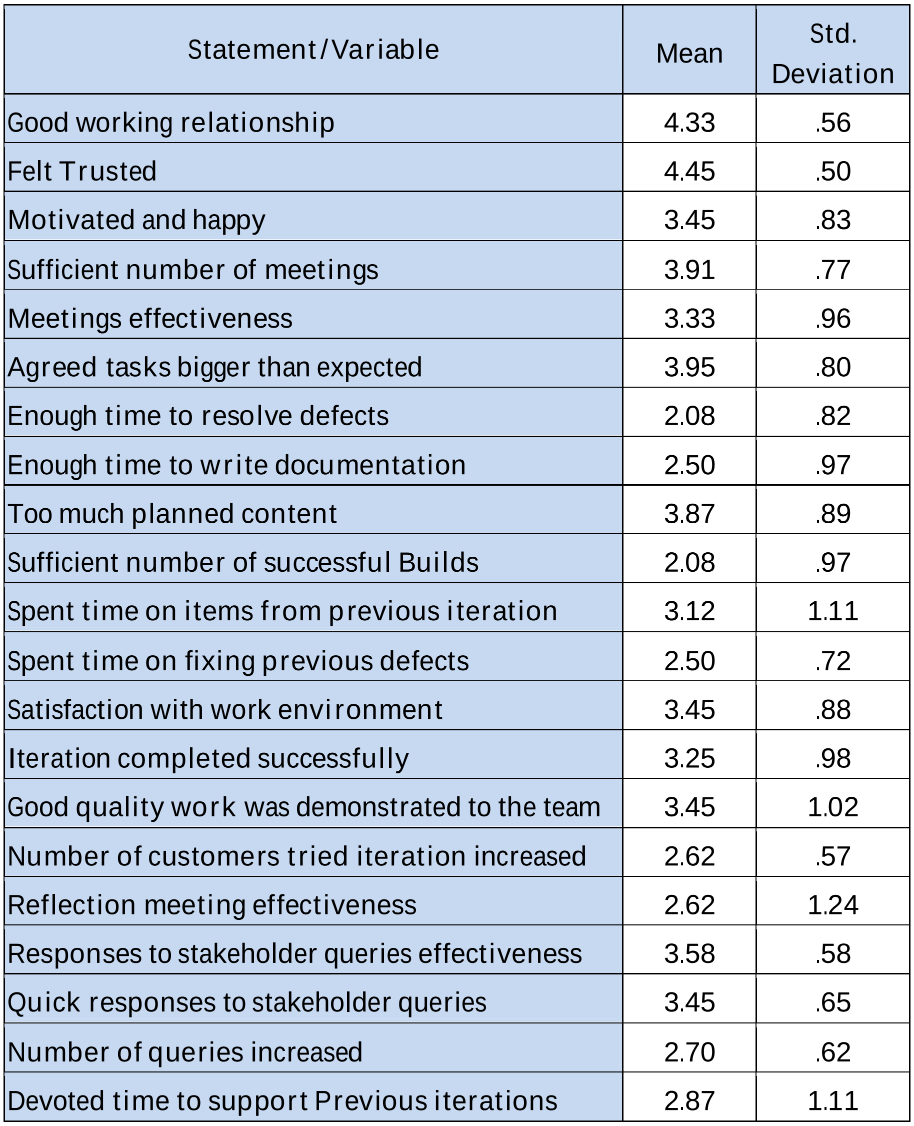
6.5.1 *Section 1: The iteration Questions*

The first section of the questionnaire had 21 statements that can be seen in table 6-2 which shows the frequency distributions (F) of the variables of the iteration section. In addition, each frequency value is expressed as a percentage of the sample (P). For example, 9 members strongly agreed that they had good relationship with other teams and no one disagreed. Also, we can see that 54% agreed that good quality software was demonstrated to the team.

Please State how much you agree or disagree with the following statements										
Statement	Strongly Agree		Agree		Neutral		Disagree		Strongly Disagree	
	F	P	F	P	F	P	F	P	F	P
I had a good working relationship with other teams (test, dev, ID etc)	9	37.5	14	58.3	1	4.2	0	0.0	0	0.0
I felt I was trusted to deliver on my tasks	11	45.8	13	54.2	0	0.0	0	0.0	0	0.0
I was motivated and happy	1	4.2	13	54.2	6	25.0	4	16.7	0	0.0
The number of iteration meetings held was sufficient	5	20.8	13	54.2	5	20.8	1	4.2	0	0.0
I felt that iteration meetings were effective and worth attending	1	4.2	12	50.0	6	25.0	4	16.7	1	4.2
The agreed/planned tasks were bigger than expected	6	25.0	12	50.0	5	20.8	1	4.2	0	0.0
There was sufficient time to resolve defects	0	0.0	2	8.3	3	12.5	14	58.3	5	20.8
There was enough time to write documentation	0	0.0	4	16.7	8	33.3	8	33.3	4	16.7
There was too much content planned for this iteration	7	29.2	8	33.3	8	33.3	1	4.2	0	0.0
A sufficient number of successful builds were delivered during the iteration	0	0.0	2	8.3	6	25.0	8	33.3	8	33.3
I spent time completing work items outstanding from the previous iteration	2	8.3	8	33.3	7	29.2	5	20.8	2	8.3
I spent an excessive amount of time fixing defects from last iteration	0	0.0	1	4.2	12	50.0	9	37.5	2	8.3
I was satisfied with the working environment	1	4.2	14	58.3	4	16.7	5	20.8	1	4.2
The iteration was completed successfully	0	0.0	13	54.2	6	25.0	3	12.5	2	8.3
Good quality working software was demonstrated to the team	2	8.3	13	54.2	4	16.7	4	16.7	1	4.2
The number of customers/internal teams trying the iterations has increased	0	0.0	0	0.0	16	66.7	7	29.2	1	4.2
The reflection/wash-up meeting was effective	1	4.2	5	20.8	9	37.5	2	8.3	7	29.2
I was able to give effective responses to stakeholders queries	1	4.2	12	50.0	11	45.8	0	0.0	0	0.0
I was able to give quick responses to the stakeholder queries	1	4.2	10	41.7	12	50.0	1	4.2	0	0.0
I received more queries from stakeholders compared to the previous iterations	0	0.0	0	0.0	19	79.2	3	12.5	2	8.3
I devoted a lot of time to supporting previous iterations/releases	2	8.3	5	20.8	7	29.2	8	33.3	2	8.3

Table 6-2 The frequency table for the iteration variables (F: Frequency, P: Percentage)

Table 6-3 shows output from SPSS. It presents the descriptive statistics for the iteration variables. As we are using the 5 point Likert-scale for measuring the agreement level with each statement, we were able to apply the mean and standard deviation measures. In this table we refer to each statement with a shorter descriptive text and the text in bold refers to the variable each statement measures, for example, the first statement measures relationship. The second measures trust and so on. We did this so we can relate each statement to a measurable variable. The reason we kept the whole statement in the previous table and the short one in the next table is to prevent confusion.



Statement/Variable	Mean	Std. Deviation
Good working relationship	4.33	.56
Felt Trusted	4.45	.50
Motivated and happy	3.45	.83
Sufficient number of meetings	3.91	.77
Meetings effectiveness	3.33	.96
Agreed tasks bigger than expected	3.95	.80
Enough time to resolve defects	2.08	.82
Enough time to write documentation	2.50	.97
Too much planned content	3.87	.89
Sufficient number of successful Builds	2.08	.97
Spent time on items from previous iteration	3.12	1.11
Spent time on fixing previous defects	2.50	.72
Satisfaction with work environment	3.45	.88
Iteration completed successfully	3.25	.98
Good quality work was demonstrated to the team	3.45	1.02
Number of customers tried iteration increased	2.62	.57
Reflection meeting effectiveness	2.62	1.24
Responses to stakeholder queries effectiveness	3.58	.58
Quick responses to stakeholder queries	3.45	.65
Number of queries increased	2.70	.62
Devoted time to support Previous iterations	2.87	1.11

Table 6-3 The descriptive statistics for the iteration variables

For example the first statement measures relationship, so a high mean stands for a good relationship between the team members which is positive, where in the statement where we ask about the planned content, the high mean is negative as it suggests too much content planned for the iteration.

When reporting the mean, we have to check the SD (Standard Deviation) to see if the mean provides a good representation of the data. From table 6-3 we can see that during the iteration, the team has a very high level of good relationship (mean=4.3) and trust (mean=4.4) between the team members. In addition, the team reported that they were fairly motivated and happy (mean=3.4) which agrees with the interview results. Although we did the interviews with only 5 members of the team, the 24 responses agreed. The team stated that the agreed tasks were relatively larger than expected for the iteration (mean=3.9), yet 54% of the team members agreed that the iteration was completed successfully (mean=3.2, with SD=.98). Also the team did not have enough time to resolve defects or to write documentation, (means are 2.0, 2.5 respectively). For all the previous reported results, apart from the "completed successfully" variable, the SD is small relative to the value of the mean therefore the mean is a good representation of the data.

When asked about the quality of the work demonstrated to the team, the mean was 3.4 with SD=1.02 which is relatively large compared to the value of the mean. Therefore, we had to go back to the frequencies to understand the team input for this variable. We can see that 62.5% of the team agreed or strongly agreed with the statement, and 20.9% disagreed. The same with the reflection meeting effectiveness with mean=2.6 and SD=1.24. From the frequency table we can see that %35.5 of the team did not find the reflection meeting effective, whereas 25% found it effective.

Finally the team gave effective and quick responses when supporting previous iterations and releases as the mean is 3.5 for effectiveness and 3.5 for quickness of the responses with SD=.58 and .65 respectively.

6.5.2 *Analysing the Relationships between the Iteration Variables*

We conducted Spearman correlation on the variables generated from the iteration questions. The correlation matrix can be found in Appendix E. The significant correlations are presented here:

- "time to resolve defects" has a positive relationship with "time to write documentations", rs =.61, (p <0.01).
- "planned content" has a negative relationship with each of
    - o "time to resolve defects", rs = - .43, (p <0.05) , and
    - o "time to write documentation", rs = - .46, (p <0.05)
- "planned content" has a positive relationship with "agreed tasks" , rs =.42, (p <0.05)
- "Agreed tasks" has a negative relationship with "time to write documentation", rs = - .45, (p <0.05)

The previous relations mean that people who found the tasks assigned to them during the iteration of reasonable size had time for defect fixing and documentation writing. Whereas people who either had too much to do or did poor estimations, were short of time and did not have enough time for other tasks.

- "motivated" has a positive relationship with each of
  - "trust", *rs* = .40, (*p* <0.05) ,
  - "environment", *rs* = .,64 (*p* <0.0), and
  - "number of meetings", *rs* = .,55 (*p* <0.01)

These relations tell us that there is a group of happy people within the team who are motivated, felt trusted, and loved the environment.

- "quality work" has a significant positive relationship with each of
  - "number of meetings", *rs* = .59, (*p* <0.01)
  - "effectiveness of meetings", *rs* = .68, (*p* <0.01)
  - "environment" , *rs* = .44, (*p* <0.05)

Also this relation tells us that people who agreed that good quality work was demonstrated to the team members were satisfied with the quality and the quantity of the meeting as well we the working environment.

- "Quick responses" has a positive relationship with "effectiveness responses" *rs* =.77, (*p* <0.01). This means that responses to stakeholder queries where quick and effective.

6.5.3 *Section Two: Effectiveness of Agile Practices*

The second section of the iteration monitor asked about the effectiveness of different agile practices during the iteration. Table 6-4 shows the frequency distributions (F) of the effectiveness of the variables. In addition, each frequency value is expressed as a percentage of the sample (P).

By looking at table 6-4 and figure 6-1, we can see at a glance that most of the practices were not used during the iteration, which was interesting as the project manager reviewed the iteration monitor before collecting the data, yet he did not remove this section. This can be seen in two ways: either the project manager is flexible about what techniques the team members are using and they are free to choose whatever they see appropriate, or the team are using these practices without naming them, because during the interviews some of these practices or at least their description was mentioned by the team members. Only two practices, user stories, and unit testing were reported by more than 75% of the team and they had an average level of effectiveness.

Of the following practices that you may have used during this iteration, how effective were they? (5= very effective, 1= not effective)												
Practice	5		4		3		2		1		N/A	
	F	P	F	P	F	P	F	P	F	P	F	P
Refactoring	0	0.0	1	4.2	0	0.0	1	4.2	0	0.0	22	91.7
Test driven development	0	0.0	0	0.0	0	0.0	0	0.0	1	4.2	23	95.8
Pair programming	0	0.0	2	8.3	0	0.0	0	0.0	0	0.0	22	91.7
Continuous integration	0	0.0	1	4.2	2	8.3	2	8.3	1	4.2	18	75.0
Simple design	1	4.2	1	4.2	0	0.0	2	8.3	3	12.5	17	70.8
Documents reviews	1	4.2	2	8.3	0	0.0	4	16.7	0	0.0	17	70.8
Peer code reviews	0	0.0	2	8.3	0	0.0	1	4.2	1	4.2	20	83.3
User stories	1	4.2	7	29.2	4	16.7	6	25.0	3	12.5	3	12.5
Unit testing	1	4.2	7	29.2	3	12.5	3	12.5	2	8.3	8	33.3
Scrum meeting	1	4.2	2	8.3	4	16.7	0	0.0	2	8.3	15	62.5

Table 6-4 The frequency table for the practices' effectiveness (F: Frequency, P: Percentage)

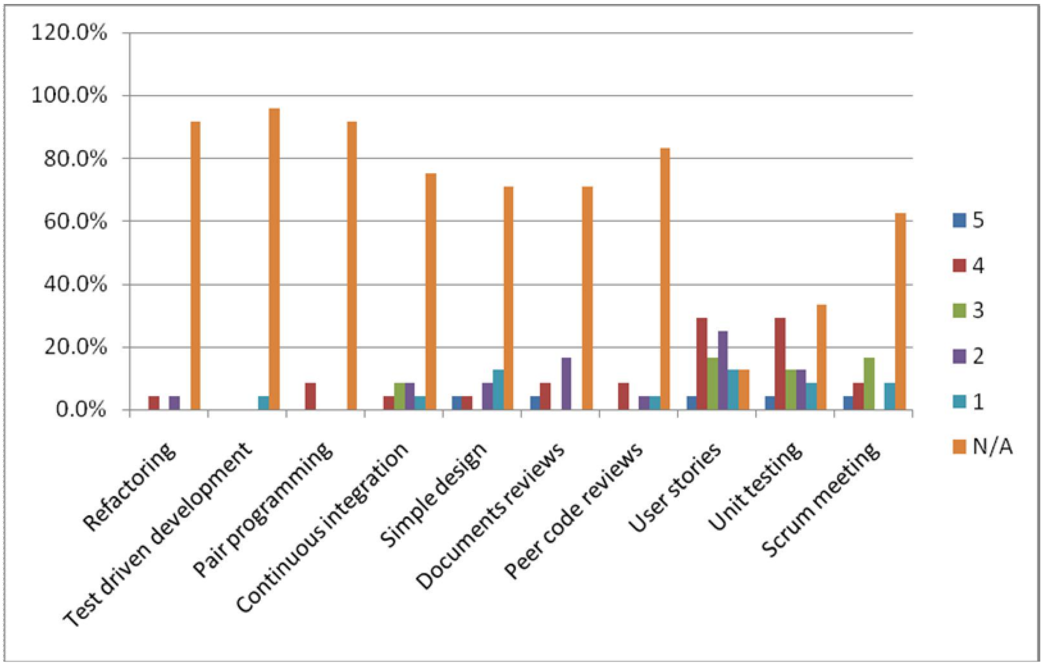


Figure 6-1 The practices' effectiveness

6.5.4 *Section Three: Communication within the Team*

In the third section of the questionnaire, we asked how often the team members used different ways of communication. Table 6-5 shows the frequency distributions (F) of the different ways of communication. In addition, each frequency value is expressed as a percentage of the sample (P). We can see that instant messaging is the most used method with as 75% of the team reported (often) and 12.5% (always). Email, meetings and informal chat are always popular, however the phone was not on the top of the list as 33.3% of the team never used the phone. The communication results are also presented in figure 6-2.

How often did you use the following when communicating with the team?										
	Always		Often		Sometimes		Rarely		Never	
	F	P	F	P	F	P	F	P	F	P
Email	2	8.3	14	58.3	4	16.7	4	16.7	0	0.0
Meetings	0	0.0	14	58.3	8	33.3	1	4.2	1	4.2
Informal chat	4	16.7	15	62.5	4	16.7	1	4.2	0	0.0
Instant messaging	3	12.5	18	75.0	3	12.5	0	0.0	0	0.0
Phone	0	0.0	1	4.2	5	20.8	10	41.7	8	33.3

Table 6-5 The frequency table for the ways of communication within the team

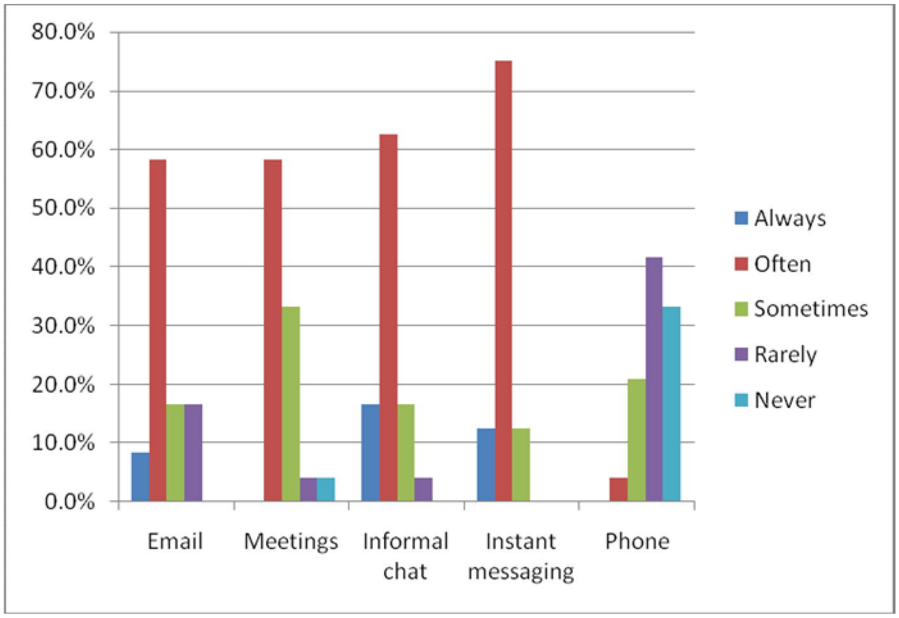


Figure 6-2 The ways of communication within the team

6.5.5 *Section Four: Iteration Focus*

In the next section of the iteration monitor we asked how much time did the team spend on different activities during the iteration. We had four areas of focus in this question: functionality, refactoring, documentation, and defect fixing. These four areas emerged during conducting the interviews, and it was interesting to see how the focus is changing over the iterations. Table 6-6 shows the frequency distributions (F) of the team opinions on the time spend on the different activities. In addition, each frequency value is expressed as a percentage of the sample (P). In addition, the percentages are presented in figure 6-3. From the table and the graph, we can see that 29% of the team stated that too much time was spent on functionality, at the same time 29% stated that the amount of time spent on functionality was just right. Also, the team stated that too little time was spent on documentation (58%) and defect fixing (62%). The responses for refactoring were not very different from the practices section where 91% reported it was not applicable. In this section, 54% of the team did not know how much time they spent on refactoring as an activity. However, 25% said they spent too little time on refactoring as an agile practice.

How much time did the team spend on the following activities during the iteration?								
	Too much		Just right		Too little		Do not know	
	F	P	F	P	F	P	F	P
Functionality	7	29.2	7	29.2	1	4.2	9	37.5
Refactoring	2	8.3	3	12.5	6	25.0	13	54.2
Documentation	0	0.0	3	12.5	14	58.3	7	29.2
Defect fixing	0	0.0	3	12.5	15	62.5	6	25.0

Table 6-6 The frequency table for iteration focus activity (F: Frequency, P: Percentage)

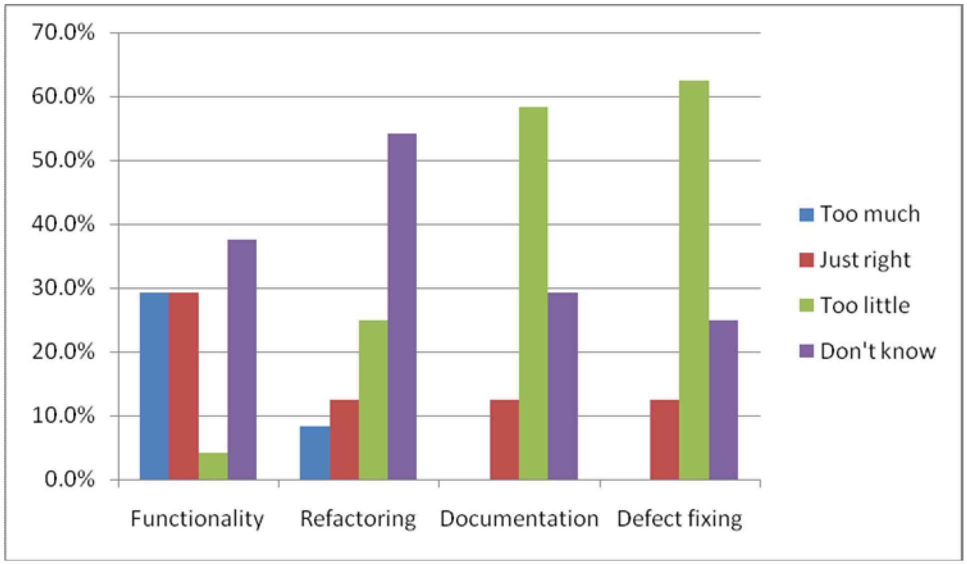


Figure 6-3 The iteration focus

In the iteration section, the team agreed that there was not enough time for defect fixing and writing documentation. This was in harmony with the iteration focus section as the team stated that there was too little time for defect fixing and writing documentation. Also, using the filtering feature in Excel, we could see that the respondents who needed more time for these two activities are the same ones who reported that too little time was spent on defects and documentation.

6.5.6 *Section Five: Stakeholders' Influence on Requirements Prioritizing*

The final section in the iteration monitor asked about the amount of influence different stakeholders had on requirements' prioritization. This issue was raised during the interviews as we had the feeling that some stakeholders had more influence than others did on requirements' prioritization. The results of this question supported our initial interview results.

Table 6-7 presents the frequency distributions (F) of the team opinions on the influence different stakeholders have on requirements' prioritization. In addition, each frequency value is expressed as a percentage of the sample (P). We can see that the architects had the biggest influence with 58.3% of the team stating they had too much. Also over 50% of the team did not know how much influence the sales and marketing, customer size, and software service had on prioritizing requirements. Furthermore, 41% of the team thought the managers had just enough influence where 37% thought they had the right amount of influence. A final observation is that the testers had too little influence on the prioritizing requirements.

How much influence each of the following stakeholders had on requirements prioritization in the iteration?								
	Too much		Just right		Too little		Do not know	
	F	P	F	P	F	P	F	P
Managers	2	8.3	10	41.7	4	16.7	8	33.3
Project managers	0	0.0	9	37.5	4	16.7	11	45.8
Developers	1	4.2	6	25.0	10	41.7	7	29.2
Testers	0	0.0	5	20.8	13	54.2	6	25.0
ID	0	0.0	7	29.2	6	25.0	11	45.8
Service	1	4.2	8	33.3	5	20.8	10	41.7
Architects	14	58.3	3	12.5	0	0.0	7	29.2
Sales and marketing	2	8.3	3	12.5	5	20.8	14	58.3
Customers	1	4.2	4	16.7	8	33.3	11	45.8
Customer size/importance	2	8.3	4	16.7	4	16.7	14	58.3
Senior executives	5	20.8	3	12.5	1	4.2	15	62.5
Software Services	2	8.3	6	25.0	2	8.3	14	58.3
Technical sales support	2	8.3	7	29.2	2	8.3	13	54.2

Table 6-7 The frequency table for the amount of influence on requirement prioritization (F: Frequency, P: Percentage)

When presenting the results to the project manager he was not surprised that the architect has the biggest influence on prioritizing requirements as it is part of their job according to the project manager. In addition, when using the Excel filtering feature, we could see that all testers stated that they have too little influence on requirements, which was not surprising.

The iteration monitor gave the team members the chance to state any suggestions or improvements. The first iteration suggestions were:

- Focus on documentation reviews
- Concentrate on producing more frequent stable builds
- Increase the level of communication between architects
- Required the manager to keep the focus of the meeting on current issues and take raised issues offline.
- Commit to work item that can be tested within the time frame
- Split large user stories into smaller chunks so they can be tracked and tested during the iteration

6.6 The Results: Iteration 2, Iteration 3

The same iteration monitor was repeated over two more iterations. In order not to repeat the results description as it has been done for iteration 1, only the results highlights will be presented here and more details will be discussed when comparing the three iterations.

6.6.1 *Iteration 2 Highlights*

Iteration 2 received 13 responses. The motivation and happiness variables improved slightly in iteration 2 with mean=4.53, 3.46 and SD=.66, .66 respectively. The time spent on functionality looked about right with 38% of the team voted for “too much” and 30% for “just right”. However, most of the team (53% for documentation and 46% for defect fixing) thought that too little time was spent on these two activities.

Improvements focused on asking to limit the planned content for the iteration/release, which is consistent with the iteration focus results (too much functionality). The team asked to focus on important issues during meetings without going into too much detail. In addition, one suggestion was to reflect on previous iterations during the reflection meeting and not during the next iteration kick off.

6.6.2 *Iteration 3 Highlights*

Iteration 3 received 10 responses. Similar to iteration 2, high level of good relationship (mean=4.50, with SD=.70), trust (mean=4.40, with SD=.70) existed within the team. In this iteration the team needed more time to resolve defects as the results showed (mean=1.80, with SD=.91) for the statement “there was sufficient time to resolve defects”. Slightly different answers to the iteration focus question were found as 50% of the responses stated that the iteration had too little focus on refactoring and documentation. The architect influence on prioritizing requirements remained high with 70% voting that it was too much. Finally, the team suggested a couple of improvements. In addition to asking for “more cakes”, the suggestions focused on traceability issues and tracking new functionality to their user stories.

6.7 Comparison

The main purpose of the iteration monitor is to identify issues and trends within the team in order to improve the process in following iterations. The response rate decreased over the three iterations. Even though the survey was shortened for the second and third iterations, still the responses went down. Because of the cut in sections, only the sections that were used for all three iterations are compared. The is done by manually calculating and comparing the means. Then the Kruska-wills test is used to test if the differences between the iterations are significant.

The means will be compared taking the standard deviation into consideration. Regarding the iteration questions section the values ranged from 5 (for strongly agree) to 1 for (strongly disagree), the means over the three iteration did not change substantially as can be seen in table 6-8 (and figure E-1 in Appendix E). Using Microsoft Excel, the variables with differences equal or over 0.5 were filtered by applying a simple IF statement. Figure 6-4 shows these variables which are:

- Sufficient number of successful builds: went up in iteration 2 and down again in iteration 3
- Iteration completed successfully: increased in iteration 2 and went down again in iteration 3
- Good quality work was demonstrated to the team: went down in iteration 2 and again in iteration 3 (the SD was a bit big comparing to the mean for this variable)
- Numbers of queries increased: had the same value in first two iteration, then went up in the third one

Iteration #	Iteration 1 (February 09)		Iteration2 (March 09)		Iteration3 (April 09)	
# Responses	N=24		N=13		N=10	
Statement/Variable	Mean	SD	Mean	SD	Mean	SD
Good working relationship	4.33	.56	4.53	.66	4.50	.70

Felt Trusted	4.45	.50	4.61	.50	4.40	.96
Motivated and happy	3.45	.83	3.46	.66	3.20	1.39
Sufficient number of meetings	3.91	.77	3.84	.55	4.00	.81
Meetings effectiveness	3.33	.96	3.46	.77	3.50	.70
Agreed tasks bigger than expected	3.95	.80	3.53	.96	4.00	1.15
Enough time to resolve defects	2.08	.82	2.15	.68	1.80	.91
Enough time to write documentation	2.50	.97	2.46	.96	2.40	1.07
Too much planned content	3.87	.89	3.61	.86	4.00	1.33
Sufficient number of successful Builds	2.08	.97	3.07	.86	2.10	.87
Spent time on items from previous iteration	3.12	1.11	3.38	.96	3.60	1.17
Spent time on fixing previous defects	2.50	.72	2.69	.85	2.90	1.28
Satisfaction with work environment	3.45	.88	3.92	.49	3.50	.70
Iteration completed successfully	3.25	.989	3.61	.96	2.80	1.22
Good quality work was demonstrated to the team	3.45	1.02	3.15	.80	2.20	1.22
Number of customers tried iteration increased	2.62	.57	2.61	.65	3.00	.47
Reflection meeting effectiveness	2.62	1.24	2.69	.75	2.60	.843
Responses to stakeholder queries effectiveness	3.58	.58	3.46	.66	3.70	.674
Quick responses to stakeholder queries	3.45	.65	3.53	.77	3.70	.674
Number of queries increased	2.70	.62	2.76	.83	3.20	1.13
Devoted a lot of time to support Previous iterations/releases	2.87	1.11	3.15	1.06	2.90	1.28

Table 6-8 Comparing Iteration variables means

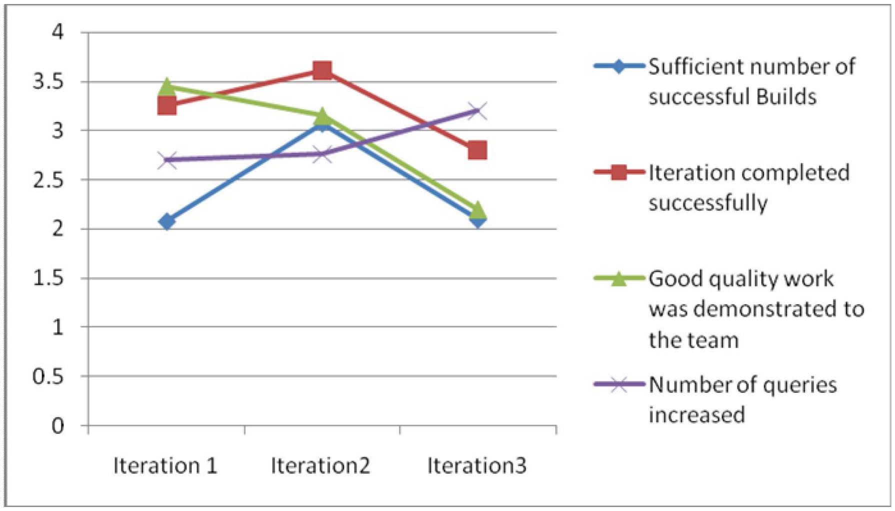


Figure 6-4 Iteration variables that changed over the three iterations (difference considered in means is >=0.5)

The next section is the iteration focus. As mentioned at the beginning of the chapter, when coding the data to calculate and compare the means, the "do not know" option was considered as missing so the mean is more meaningful. The values ranged from 3 (for too much) to 1 (for too little). This also was applied on the influence on requirements prioritizing section. Table 6-9 and graph 6-5 present the data from the three iterations, where it can be observed that the time spent on functionality was relatively the same over the three iterations and it was a little bit over 2 (just right). On the other hand, the team reported that the time spent on refactoring and documentation was not enough as the mean had low values for the two activities over the three iterations. The most interesting result is that the focus on defect fixing was low in the first and second iterations and went up to be just over 2 (just right) in iteration 3. It looks like a lot of defect fixing was done in iteration 3, (when asking the project manager he said that it was a defect fixing iteration).

	iteration 1		iteration 2		iteration 3	
	Mean	SD	Mean	SD	Mean	SD
functionality	2.40	0.63	2.56	0.53	2.43	0.79
refactoring	1.64	0.81	1.57	0.79	1.43	0.79
documentation	1.18	0.39	1.30	0.48	1.44	0.53
defect fixing	1.17	0.38	1.33	0.50	2.13	0.83

Table 6-9 Comparing iteration focus

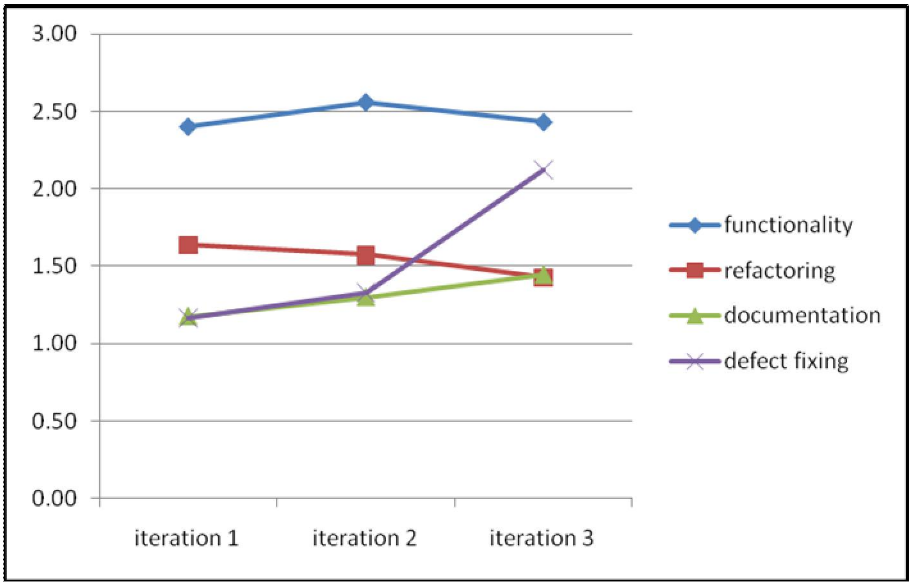


Figure 6-5 Comparing iteration focus

The final section is the stakeholders influence on requirements prioritizing. Table 6-10 presents the means for the three iterations where we can see that for most of the stakeholders the team's opinion did not change much over the three iterations, for example, all three iterations agreed that the testers had too little influence on requirements prioritizing. Figure 6-6 presents the stakeholders where the influence on requirements changed over the three iterations (where differences in mean are >=0.5), also see figure E-2 in Appendix E.

	iteration 1		iteration 2		iteration 3	
	Mean	SD	Mean	SD	Mean	SD
Managers	1.88	0.62	2.29	0.49	2.00	0.53
project managers	1.69	0.48	1.83	0.75	1.29	0.49
developers	1.47	0.62	1.90	0.57	1.38	0.74
testers	1.28	0.46	1.70	0.48	1.29	0.49
id	1.54	0.52	1.57	0.53	1.50	0.55
service	1.71	0.61	1.67	0.52	1.33	0.52
architects	2.82	0.39	2.33	0.50	2.88	0.35
sales marketing	1.70	0.82	2.00	0.89	2.40	0.55
customers	1.46	0.66	1.43	0.53	1.14	0.38
customer size	1.80	0.79	1.80	0.45	1.50	0.84
senior executives	2.44	0.73	2.40	0.55	2.57	0.53
software services	2.00	0.67	1.83	0.75	1.60	0.55
tech sales support	2.00	0.63	1.67	0.52	1.33	0.58

Table 6-10 Comparing stakeholders influence on requirements prioritizing

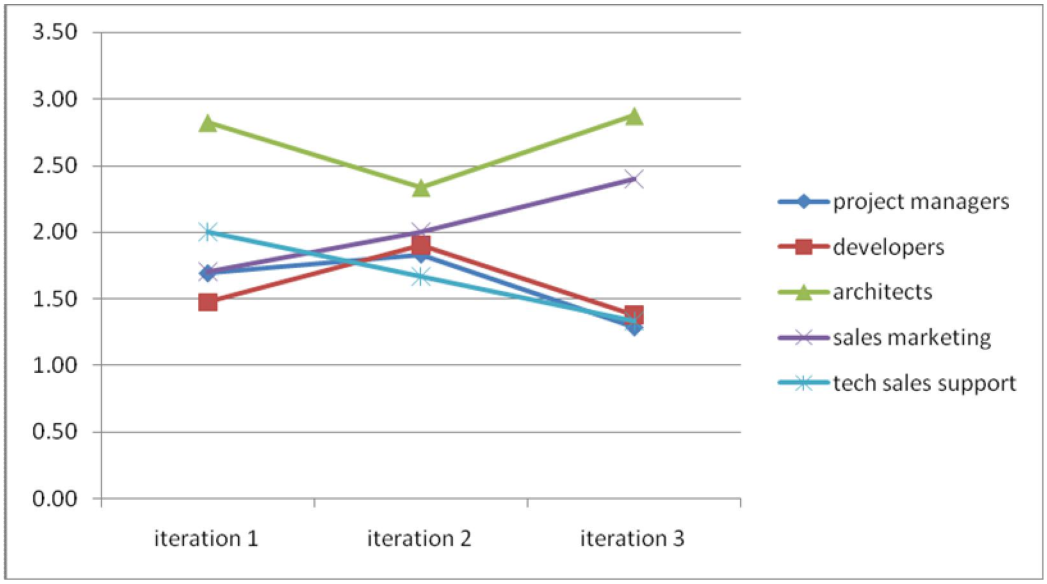


Figure 6-6 Stakeholders whom influence on requirements prioritizing changed over the three iterations (difference considered in means is >=0.5)

Although it was difficult to deeply interpret these results and link them to the team and the process changes as the team has not been observed very closely, yet it can be argued that the results are valuable to the project manager and to the team in general. Mainly because it is a big team (77 and growing), mostly co-located, long term (3 years and still on), so the iteration monitor presented a useful tool to collect data about the iteration, and to communicate with the whole team, understand how they see the development process, and make sure that the whole team has the same understanding. In addition, comparing the results over the three iterations can be used to reflect and change. We believe that the iteration monitor is a useful tool for process improvement especially with the quick rhythm of the iterations in agile software development where there is a need for a simple tool that can produce quick results so the changes can reflect on future iterations.

The data was re-arranged so the Kruskal-Wallis test for the differences between the three iterations can be applied. SPSS output can be seen in table E-2 in Appendix E where we can see each variable and its related results. The results showed that for most of the variables, the difference was not over the three iterations. However, the difference was significant (p<.05) for the following are variables:

- The team opinion on whether "there was too much content planned for this iteration" changed significantly over the three iterations (H(2)=8.81, p <.05) (the significance is .012 which is less than .05 with df=2)
- The team opinion on whether "the iteration completed successfully" significantly changed over the three iterations (H(2)=8.94, p <.05)
- The time assigned for defect fix changed significantly over the three iterations (H(2)=10.19, p<.05)
- The team opinion on the influence of the architects on requirement prioritizing significantly changed over the three iterations (H(2)=8.00, p<.05)

We can see that this analysis gave us some more information about the change over the three iterations; some already have been concluded from the manual comparison of the means. However, this test is more valid as it tested the significance of the difference.

6.8 Limitations, Applicability and Lessons Learned

The iteration monitor presented in this chapter was created to collect further details about the iteration. The iteration monitor is basically a survey, so it has the surveys limitations, mainly, the collected data is self-reported, and poor memory or misunderstanding of the questions can all contribute to inaccuracies in the data (Nardi 2002). This is hopefully did not affect the validity of the iteration monitor, as the iteration was introduced to the team, and its purpose was explained along with each section and its questions. Also, the data was collected after the 4 weeks iteration was finished, so it is unlikely the participants have forgotten the iteration information. Furthermore, the participants were professionals and the questions did not have a personal nature.

One important limitation of the iteration monitor is the response rate, in our case, the number of responses went down over the iterations. So it is important here to mention that the focus of this chapter is the iteration monitor itself as a tool to collect data about the iteration from the team, and how these data can be analysed and interpreted so the results can be used to reflect on the team to improve the development process. The results were also important, yet the low response rate especially in iterations 2 and 3 was a problem in generalising these results on the whole team. The point here is that in order to guarantee a successful application of the iteration monitor, high response rate is recommended.

The data analysis was done by the researcher manually, so if a team is planning to use the iteration monitor, the team should have an analyst who can be one team member, however this means more work load. The other option is to create a tool that analyses and presents reports to the team. The tool can design the monitor and allow changing the questions as needed. This can be done as part of the future work that is going to follow this research.

The iteration monitor is recommended to be used when retrospectives or the reflection meetings are not very effective. This can happen when the team is getting larger and the one hour meeting is not enough so the whole team can have their say in the meeting. In addition, the iteration monitor can be useful when the project is running for a long time so keeping a record for each iteration will be very useful to identify trends and changes over time. This can lead to a quick process improvement that can impact the following iterations Furthermore, these data can be useful for future projects as a reference and a real, easy to collect, case study. The iteration monitor questions were designed to address the needs of the studied team in project B, other teams may adapt the monitor as needed by changing it according to their settings.

The project manager asked to use the iteration monitor for the next release, as he was interested in the collected data after the initial results were presented to him during the last interview. For next iterations, the team should be more encouraged to complete the iteration monitor, and this can be done by making it a part of the process and filling it is considered as one of the team member’s tasks. Also, the data should be analysed before the start of the next iteration so the team can reflect on the process where possible, and it is recommended that the project manager passes on the results to the team, and explain which results will impact the process, and which will not and why. This will encourage the team to fill the monitor when they see its impact on the process. Unfortunately, this means more work load on the project manager (or the team member who will interpret and present the results), at least at the first few iterations, one option can be to rotate this task between the team members if possible, and this will not be much work as the iteration monitor is conducted after each iteration.

6.9 Reflection on Research Hypotheses

Analyzing the iteration monitor data produced a large set of results; some were related to four of the hypotheses presented in the previous chapter. We used the mean of means, i.e. the mean of each variable means for the three iterations; we can use the data in table 6-8 for this purpose. The reader should keep in mind that for the iteration monitor section the values range was (1-5).

- H5: Agile software development assures quick and effective response to stakeholders' requests: this hypothesis was supported by two variables in the iteration monitor: effectiveness of responses, and their quickness, with means: 3.56 and 3.58 (not statistically significant)
- H6: Agile software development can produce good quality software: this hypothesis was also supported by two variables: the iteration completed successfully (mean=3.22), and good quality work was demonstrated to the team (mean=2.98) (not statistically significant)
- H21: Team members are happy and motivated when using agile software development: This hypothesis was supported by three variables: relationship, motivation and happiness which had the following means over the three iterations: 4.45, 4.49, 3.37 (not statistically significant).
- H11: The quality of the code increases as the number of iterations increases: the descriptive showed that the quality went down from iteration 1 to 3 according to the team. However, the statistical test showed that the quality was significantly different among the three iterations, however we could not tell if it increased or not.

6.10 Summary

This chapter presented the iteration monitor, a survey tool that was newly designed by the researcher to understand how things are changing over the iterations and more importantly, what are the team members' opinions about the process, the quality of the product, and the support provided to the different stakeholders. The iteration monitor can be used as a first stage process improvement tool as its main purpose is to diagnose the iteration trends and problems so an immediate action can be taken to improve future iterations. The iteration monitor was used over 3 iterations by a team in IBM. Using the iteration monitor in a real project was very important to identify its limitations, applicability and to explore how it can be improved. The iteration monitor results supported four of the research hypotheses.

Chapter 7

Applying Correlations and Factor Analysis on Existing Surveys

7.1 Introduction

The main purpose of this research is to investigate the impact of agile methods on software quality and customer satisfaction. The conducted empirical study was an exploratory tool that helped us understand how agile approaches are used in industry, and what is their impact on different aspects of quality. Although studying IBM experience was very valuable, it is important to study the experience of other organisations so the results can be more general and not related to one organisation only. Therefore, we decided to explore the experience of other organisations using a survey to collect as much data as possible about other organisations' experience. It was moreover decided to explore existing surveys to avoid repeating questions which had previously been asked. Agile adoption surveys that were conducted since 2006 (Ambler 2006) were available with their raw data so other researchers can reanalyse them. The surveys received good number of responses (4232 responses in 2006, 781 in 2007, 642 in 2008) and they included questions that can be useful for our research, we decided to further analyse these surveys data for our research purpose. This chapter will present three surveys; their initial results, how they support to the research hypotheses, and it will present our own analysis of the data and our results.

7.2 Analysis

Ambler presented descriptive statistics when analysing the data. In order to further analyse the data we used more complex statistical approaches namely Spearman correlation discussed in previous chapter and the factor analysis, which will be discussed later in this chapter. In order to apply statistical methods on the current data we had to recode them into numbers using SPSS. This was done using a simple syntax that was applied on all columns we need to recode. The result was a new set of columns with coded data. The frequencies of the emerging data were compared against the original ones to make sure that the recoding was done correctly.

7.3 Agile Adoption Survey 2006

7.3.1 *Description and Summary of the Results*

The agile adoption survey was performed in March 2006 and received 4235 responses (Ambler 2006). The survey was sent out to Dr. Dobb's Journal and software development mailing lists. It collected information about the respondent, agile adoption, and agile approaches impact on productivity, quality, and stakeholder satisfaction. In September 2006, the results were published in Dr. Dobb's Journal and they indicated that agile software development methods and techniques are gaining more interest. Ambler concluded that the adoption of agile *techniques* is further ahead than the adoption of agile *methods*, he related that to the idea that most organisations chose to perform software process improvement on an incremental basis, so it made sense that we would see some organisations just getting started.

The average of the respondents appear to be highly skilled, there was a fair representation of organisations; 54% believed that they have limited or very limited knowledge regarding agile techniques where 33% believed that they had average understanding. Regarding agile adoption, XP and Scrum were the popular options; also FDD had a strong showing. Quality oriented techniques such as coding guidelines, refactoring, continuous integration, and TDD had high acceptance rates. The results showed that agile approaches had good impact on productivity as only 6% indicated that their productivity was lowered. No change in productivity was reported by 34% of respondents and 60% reported increased productivity. Furthermore, agile approaches had good impact on quality, with 66% responding that the quality is higher. Considering the high rate of adoption of quality-oriented techniques, this should not come as a surprise. 58% of the organisations reported improved stakeholder's satisfaction, whereas only 3% reported reduced satisfaction. Finally, 15% reported increased cost, whereas 14% reported reduced cost when using agile approaches. We can see that the survey descriptive results support two of the generated hypotheses, H1: agile software development can achieve customer satisfaction, and H6: agile software development can produce good quality software, however this support is not statistically significant.

7.3.2 *Our Analysis*

In the agile adoption 2006 survey, we were particularly interested in four questions that were relevant to our research:

- Number of IT people in your organisation
- How have agile approaches affected your productivity?
- How have agile approaches affected the quality of systems deployed?
- How have agile approaches affected the cost of the systems?
- How have agile approaches affected the business stakeholder satisfaction?

Each impact question was followed with 6 options (much lower, somewhat lower, no change, somewhat higher, much higher, and do not know). The organisation size question had a list of ranges. We recoded each variable and considered the “do not know” option as missing so the results are more meaningful. The data was coded so the big number represents the higher change for the impact questions and the larger number of people for the size question. This coding was consistent throughout the thesis. We decided to analyse the data using correlation, which is a measure of the relationship between these variables. However, we first needed to explore the data in order to know what type of correlation is more appropriate. Screening the data showed that it was not normally distributed, and although we considered applying data transformation, however it did not change the distribution of the data. Therefore, we decided to apply Spearman’s correlation coefficient *rs* which is a non-parametric statistic and it can be used when the data is not normally distributed.

SPSS output is presented in table 7-1 where a matrix is displayed giving the correlation coefficient between the variables: productivity, quality, cost, satisfaction, as well as the organisation size. We omitted the bottom part of the matrix as it is symmetrical. From that table below we can see that:

- There is a significant relationship between productivity and quality, *rs* =.68, (*p* <0.01)
- Productivity is significantly correlated with satisfaction, *rs* =.60, (*p* <0.01).
- There is a significant relationship between quality and satisfaction, *rs* =.66,(*p* <0.01)
- There is a negative relation between cost and productivity *rs* = -.20, (*p* <0.01) , quality *rs* = -.06, (*p* <0.01) and satisfaction *rs* = -.06, (*p* <0.01).
- There is a positive significant relationship between organisation size and cost, *rs* =.065, (*p* <0.01).

	Productivity	Quality	Cost	Satisfaction	Organisation size
Productivity	1.000	.684**	-.203**	.603**	0.23
Quality		1.000	-.06**	.660**	-.004
Cost			1.000	-.066**	.065**
Satisfaction				1.000	0.14
Organisation size					1.000

** Correlation is significant at the 0.01 level

Table 7-1 Correlation between variables in agile adoption survey 2006

We can conclude that when agile approaches had good impact on quality they also had good impact on satisfaction. Therefore, we can say that quality is a factor in achieving stakeholders' satisfaction. In addition, we can say that as productivity improves, quality and satisfaction improve as well. The interesting result is that when productivity, quality, and satisfaction went higher the cost went lower as the correlation was negative. In addition, it seems like the impact of agile approaches on cost was higher in larger organisations.

7.4 Agile Adoption Survey 2008

7.4.1 *Description and Summary of the Results*

The agile adoption survey was performed in February 2008 and received 642 responses (Ambler 2008b). The survey was sent out to Dr. Dobb's Journal and software development mailing lists. Similarly to the 2006 survey, this one also collected information about the respondent, agile adoption, and how they were affected by agile in terms of productivity, quality, and stakeholder satisfaction. In addition, it collected information about agile projects, their success, iteration length, team size, co-location and off shoring. In May 2008 the results were published in Dr. Dobb's Journal and they indicated that agile software development appears to still be growing in popularity. Furthermore, agile strategies are being successfully adopted by the majority of organisations. The survey found that the majority of respondents indicated that their iterations were between one and four weeks in length. (2 weeks: 32.8%, 3 weeks: 16.7%, 4 weeks: 22.8%). The results showed that the average success rate for agile teams was 77%. Also agile approaches had good impact on productivity as only 5% indicated that their productivity was lowered an even better result than the 2006 survey. No change in productivity was reported by 13% of respondents and 81% reported increased productivity. Furthermore, agile approaches had good impact on quality, with 77% responding that the quality is higher compared to 66% in 2006 survey.

78% of organisations reported improved stakeholder satisfaction compared to 58%, whereas only 7% reported reduced satisfaction. Finally, 37% reported that agile approaches helped reducing cost where 23% reported that it increased cost. Again, this survey results support the same hypotheses (H1 and H6) as the previous one regarding agile software impact on software quality and stakeholders' satisfaction.

7.4.2 *Our Analysis*

In the agile adoption survey 2008 we were interested in 6 questions:

- What is the total number of people in your organisation?
- What is the overall success rate for all of your agile projects?
- How have agile approaches affected your productivity?
- How have agile approaches affected the quality of the systems produced?
- How have agile approaches affected the cost of development?
- How have agile approaches affected stakeholder satisfaction?

Each question was followed with a different set of options; as in the previous survey, we had to be careful with coding to keep the scales consistent. For example, the success rate question was coded as presented in table 7-2:

Option	Code
91-100%	10
81-90%	9
71-80%	8
61-70%	7
51-60%	6
41-50%	5
31-40%	4
21-30%	3
11-20%	2
10% or less	1
Not Applicable	Missing
Do not Know	Missing

Table 7-2 An example of recoding variables in SPSS

The agile approaches impact questions were coded the same way (the high number for higher impact).

Spearman correlation coefficient was applied on the coded data, and SPSS output is presented in table 7-3 where we can see that:

- There is significant positive relationship between productivity and quality, rs =.55, (p <0.01)
- There is a positive relationship between productivity and satisfaction, rs =.43, (p <0.01)
- There is positive significant relationship between quality and satisfaction, rs =.51, (p <0.01)
- Interestingly there is a negative relation between cost and each of productivity, rs = -.41, (p <0.01), quality rs = -.26, (p <0.01) satisfaction rs = -.28, (p <0.01), and success rate rs = -28., (p <0.01)
- Success rate had positive relationship with each of quality rs =.36, (p <0.01), productivity, rs =.41, (p <0.01) and satisfaction, rs =.27, (p <0.01), which is not surprising.
- Finally organisation size had a negative relationship with success rate rs = -.81, (p <0.01) and productivity, rs = -.11, (p <0.01)

	Success Rate	Productivity	Quality	Satisfaction	Cost	Organisation size
Success Rate	1.000	.411**	.366**	.274**	-.286**	-.181**
Productivity		1.000	.557**	.438**	-.413**	-.114*
Quality			1.000	.515**	-.268**	-.065
Satisfaction				1.000	-.281**	.023
Cost					1.000	.085
Organisation size						1.000

** Correlation is significant at the 0.01 level

* Correlation is significant at the 0.05 level

Table 7-3 Correlation between variables in agile adoption Survey 2008

The results of this survey are similar to the previous one. When agile approaches had good impact on one aspect, it had it on all the others, more importantly the results suggest that agile methods reduced the cost of software development in both surveys, and this was correlated with achieving higher quality, satisfaction, productivity, and success rate. In addition, it seems like large size organisation are having problems applying agile approaches, as there is a negative correlation between organisation size and each of success rate and productivity. The last thought is that the correlation between success rate and each of quality and satisfaction will be useful for analyzing surveys that did not measure the last two variables but measured success rate, and as the correlation existed, we can link variables that correlate with success rate to quality and satisfaction.

7.5 Agile Adoption Survey 2007

7.5.1 *Description and Summary of the Results:*

The reason why we presented the surveys from 2006 and 2008 first is that they were quite similar. The 2007 survey was performed in March 2007 and received 781 responses (Ambler 2007) and it was promoted in Jon Erickson's blog in www.ddj.com. The survey collected information not only about agile projects, success rate and iteration length, but it included a section about the effectiveness of different agile practices. In July 2007, the results were published in Dr. Dobb's Journal and they indicated that agile techniques have been successfully adopted within the majority of organisations and often at scale. The results showed high success rate as 77% of the respondents indicated that 75% or more of their agile projects were successful.

Similar to 2008, the majority of agile teams had short iterations between one and four weeks (1 week: 17%, 2 weeks: 32.6%, 3 weeks:12.5%, 4 weeks: 21%). Regarding the effectiveness of agile practices, the high scoring practices were iterative development, regular delivery of working software, and simple design. Pair programming did not score very well. Ambler argued that this might be because many organisations do not give it enough time or because he had to distinguish between promiscuous pairing where pairs are swapped regularly and nonpromiscuous pairing when he asked the question.

7.5.2 *Applying Factor Analysis on Agile Adoption Survey 2007*

Although Ambler presented the effectiveness of different practices, we needed to further explore how these practices are grouping together and how they are relating to success rate and therefore to quality and satisfaction. The survey asked about 58 practices categorized in five categories: development practices, modelling and documentation practices, testing and quality practices, management and organisational practices and work product. In order to understand the structure of these variables we needed to reduce the huge data set to more manageable size while retaining as much of the original information as possible. We believe that the best way to analyse this data set is by using factor analysis as described by (Field 2005). Factor analysis will reduce the data set (58 practices) into a smaller set of factors by explaining the maximum amount of common variance in a correlation matrix using the smallest number of explanatory concepts.

In order to apply this analysis we recoded the practices variables and the overall success rate variable. Each practice had a 5 points scale with 5 being very effective and 1 less effective and options of “do not know” and “not applicable” which were coded as missing. In the next section, we will explain how the factor analysis was applied and we will interpret its results.

- Initial Considerations

Sample Size: The reliability of the factor analysis is dependent on sample size. (Kass et al. 1979) recommended having between 5 and 10 participants per variable up to total 300. (Tabachnick et al. 2001) agreed that it is comforting to have at least 300 cases for factor analysis. So a sample of 300 or more will probably provide a stable factor solution. Another way is to measure the Kaiser-Meyer-Olkin measure of sampling adequacy (KMO), which represents the ratio of the squared correlation between variables to the squared partial correlation between variables. According to (Kaiser 1974) a KMO value that is greater than .5 is acceptable, values between .5 and .7 are mediocre, values between .7 and .8 are good, values between .8 and .9 are great, and values above .9 are superb. With our sample size and a KMO of .87 as measured by SPSS, we are confident that factor analysis is appropriate for the agile adoption survey data.

Data Screening: Before running the analysis, we have to screen the data to eliminate any variables that should be excluded before the analysis is run. We can do that using the correlate procedure to create a correlation matrix of all variables. We use this matrix to eliminate variables that do not correlate with any other variables or that correlate very highly with other variables (r<.9) (Field 2005). In our example, we could not find any variable that fits the previous description therefore; we included all the variables in the analysis.

- Running the Analysis and Interpreting the Results

We started with selecting the variables we need to include in the analysis. There are several options available. The descriptive option allows us to calculate a number of important measures, such as KMO which is .87 in our case.

Factors Extraction: There are several methods for unearthing factors in the data. The method choice depends on the analysis purpose. When factor analysis was originally developed it was assumed that it would be used to explore the data in order to generate future hypotheses. As such, it was assumed that this technique would be applied to the entire population of interest. Such techniques assume that the sample used is the population. Principal component analysis is an example of one of these techniques. Other techniques are available for other purposes, such as the maximum likelihood method and Kaiser's alpha factoring for results generalisation and the confirmatory factor analysis for testing a specific hypothesis (Field 2005).

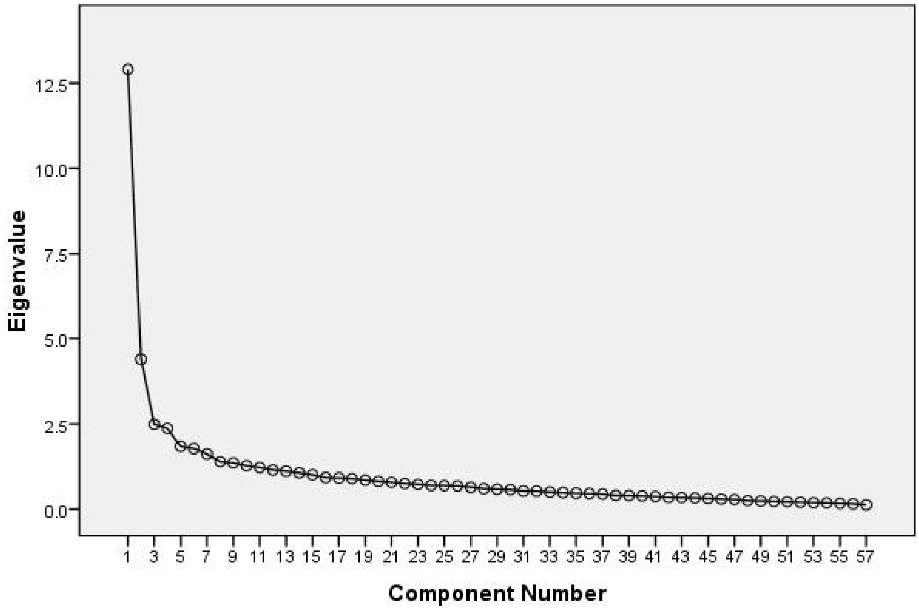


Figure 7-1 Screen plot for factor analysis

The factor extraction gave us the component matrix (table F-1 in Appendix F). Although the component matrix is not important for interpretation, it is important for understanding the importance of the factor rotation. We can see in this matrix that most variables load highly onto the first factor. At this stage, SPSS has extracted 15 factors. Statisticians recommend not to leave the final decision to SPSS regarding the number of extracted factors but to use its results as a guide. With a sample size over than 200 participants, the screen plot provides a fairly reliable criterion for factors selection (Stevens 1992). The screen plot shown in figure 7-1 is a graph of each eigenvalue against the factor which it is associated with, where the eigenvalues represent the amount of variation explained by a factor. (Kaiser 1974) recommended retaining all factors with eigenvalues greater than 1 which is a substantial amount of variation. These factors can be seen in the component matrix which contains the loading of each variable onto each factor which depends on the variable's correlation to the factor. We can see blank spaces for some variables because we requested SPSS to show suppress loadings that are less than .4 to make the interpretation simpler. Although for large samples small loadings can be considered statistically meaningful, (Stevens 1992) recommends interpreting factor loadings with an absolute value greater than .4.

Factors Rotation: The interpretability of factors can be improved through rotation. Rotation maximises the loading of each variable on one of the extracted factors which minimise the loading of the other variables. Therefore, this process makes it much clearer which variables are related to which factors. In order to decide which rotation method is more appropriate to our data, we tried to run both methods: the orthogonal rotation (varimax) and the oblique rotation. The late one produced a correlation matrix between the factors (table F-2 in Appendix F). If the components were independent then we would expect the oblique rotation to provide an identical solution to the orthogonal rotation and the component correlation matrix should be an identity one. The fact that these correlations exist tells us than we cannot assume independence and therefore the results of the orthogonal rotation should not be trusted and the obliquely rotated solution is more meaningful. The oblique rotation produced two matrices: the pattern matrix (table F-3 in Appendix F) and the structure matrix (table F-4 in Appendix F). The pattern matrix contains the factor loadings that are calculated after rotation. We can see that the rotation of the factors has clarified things considerably. The structure matrix takes into account the relationships between factors. At this stage we could look at the practices that load onto the same factor and try to identify common themes, then we double check with the structure matrix by doing the same thing (Field 2005).

As we have 15 factors which is a large number we will discuss a couple of examples in detail and the rest are presented in table 7-4. The practices that load highly on factor 15 are iterative development, incremental delivery, small release, and sustainable pace. These practices are the core of agile software development. We can call this factor iterative and incremental development. Also, the practices that load highly on factor 6 are all agile quality assurance practices: continuous code integration, test driven development, code refactoring and developers' tests. We can call this factor agile quality assurance practices. We can see that the factor analysis has re-categorized the 58 agile practices so we can study a smaller set of variables (15 compare to 58).

Factor1:architecture modelling • initial agile architectural modelling • initial agile requirements modelling • evolutionary design • proved architecture early	Factor2: traditional analysis • Gantt chart details • Gantt chart high-level • case tool modelling • architecture specification detailed • requirements specification details	Factor3: process/governance • burn down chart • velocity • planning game • daily stand up meeting • iteration task list • *regular status report* • *defect trend metrics*
Factor4: database practices • continuous database integration • database testing • database refactoring • data naming conventions	Factor5: communication (team) – whiteboard Practices • whiteboard sketches • whiteboard sketching modelling	Factor6: agile quality assurance • continuous code integration • test driven development • code refactoring • developer tests • *flexible architecture* • *evolutionary design* • *simple design* • *collective ownership*
Factor7: communication (team) • paper based modelling • paper models • pair programming	Factor8: code analysis and inspection • static code analysis • code inspection	Factor9: lightweight testing and review • independent confirmatory exploratory testing • customer acceptance tests • model document reviews
Factor10: architecture and configuration • architecture specification high-level • configuration management • architecture specification detailed	Factor11: traditional quality assurance • test plan • source code • defect reports • *regular status report*	Factor12: coding standards • coding standard • data naming conventions
Factor 13: lightweight requirements • requirements specification high-level • use cases light	Factor14: incremental and iterative development • incremental delivery • small releases • iterative development • sustainable pace • *active stakeholder participation* • *working demoable software*	Factor15: communication (customers) • co located team • active stakeholder participation

Table 7-4 The extracted factors and their related variables

After studying both pattern and structure matrices, we were able to recognize a 15 factors which can be used when studying any agile project. The practices in italic have been added after considering the structure matrix. We can see that many practices are related to more than one factor, which is not surprising. The extracted factors can be used as a checklist in case a company or organisation wants to focus on improving one aspect of the development process. If we consider the factor governance for example, the practices that formed this factor such as burn down chart, velocity, and planning game can be used as a guide for the company when focusing on governance. An interesting factor is the agile quality assurance factor which includes all agile practices that relate to quality assurance such as continuous integration, refactoring and test driven development, where traditional quality assurance practices formed a different factor.

Factor Scores: The factor scores are another important output of the factor analysis. A factor can be described in terms of the variables measured and the relative importance of them for that factor. Therefore, it should be possible to estimate a person's score on a factor based on their scores for the constituent variables. The most use of factor scores is to reduce a large set of data into a smaller subset of measurable variables where the factor scores tell us an individual score on this subset of measures. Furthermore, we can carry out future analysis on the factor scores rather than the original data.

There are several techniques for calculating factor scores, of which the regression method preferred as it is the most easily understood one. However, the problem with this method is that it produces factor scores that are biased as they can correlate with other factor scores. There are two methods to solve this problem; the Barlett Method which produces scores that are only correlated with their own factors, and the Anderson-Rubin method that produces uncorrelated scores. In our example correlation scores are not a problem therefore the Barlett method is used. The factor scores will be added to the original data where we will have 15 new columns for the 15 new factors and now we can apply different types of analysis on the new factors (Field 2005).

When applying correlation between the extracted factors and success rate we got the correlation matrix in table 7-5, below we present the significant correlations for the factors are related to this research:

- success rate has a positive relationship with each of
  - agile quality assurance practices, r_s =.16, (p <0.01)
  - iterative and incremental development, r_s =.25, (p <0.01)
- success rate has a negative relationship with each of
  - traditional analysis practices, r_s = -.12, (p <0.05)
  - communication within the team (whiteboard practices), r_s = -.16, (p <0.01)
  - coding standards practices r_s = -.16, (p <0.01)
- governance practices has a positive relationship with each of
  - architecture modelling, r_s =.12, (p <0.05)

  - agile quality assurance, *rs* =.20, (*p* <0.01)
  - iterative and incremental development, *rs* =.21, (*p* <0.01)
  - Communication with the team, *rs* =.17, (*p* <0.01)
- governance practices has a negative relationship with each of
  - Traditional quality assurance, *rs* = -.13, (*p* <0.05)
  - communication with the customers, *rs* = -.19, (*p* <0.01)
- agile quality assurance has a positive relationship with each of
  - architecture modelling, *rs* =.14, (*p* <0.05)
  - iterative and incremental development, *rs* =.32, (*p* <0.01)
  - Communication with the team, *rs* =.16, (*p* <0.01)
- agile quality assurance has a negative relationship with each of
  - communication with the customers, *rs* = -.11, (*p* <0.05)
  - communication within the team (whiteboard practices), *rs* = -.20, (*p* <0.01)
- iterative and incremental development has a positive relationship with architecture modelling, *rs* =.26, (*p* <0.01)
- iterative and incremental development has a negative relationship with communication with customers, *rs* = -.11, (*p* <0.01)
- communication with customers has a positive relationship with communication within the team (whiteboard practices), *rs* =.19, (*p* <0.01)

	SR	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
SR	1.000	.069	-.125*	.064	-.046	-.164**	.169**	-.020	-.013	.021	-.035	-.072	-.163**	.062	.257**	-.053
F1		1.000	.235**	.120*	-.273**	-.171**	.149*	.153**	.134*	.208**	-.175**	-.169**	-.123*	.096	.265**	-.105
F2			1.000	.059	-.138*	-.016	.002	.103	.214**	.176**	-.118*	-.232**	-.042	.138*	-.038	.046
F3				1.000	-.173**	-.231**	.205**	.179**	.150**	.172**	-.049	-.135*	-.043	-.023	.216**	-.192**
F4					1.000	.091	-.197**	-.186**	-.130*	-.166**	.105	.147*	.125*	-.023	-.208**	.053
F5						1.000	-.207**	-.085	-.091	-.102	.112	.126*	.124*	-.063	-.205**	.192**
F6							1.000	.164**	.141*	.062	.002	-.063	-.129*	.025	.320**	-.117*
F7								1.000	.158**	.098	-.118*	-.080	-.066	.013	.128*	-.080
F8									1.000	.151**	-.066	-.153**	-.060	.094	.067	-.042
F9										1.000	-.060	-.236**	-.042	.103	.130*	-.118*
F10											1.000	.078	-.018	-.030	-.030	.037
F11												1.000	.121*	-.128*	-.095	.021
F12													1.000	-.061	-.028	.031
F13														1.000	.104	-.042
F14															1.000	-.245**
F15																1.000

*. Correlation is significant at the 0.05 level (2-tailed).

**. Correlation is significant at the 0.01 level (2-tailed).

Table 7-5 Correlation coefficient between the extracted factors and SR (Success Rate)

According to the previous results, we can argue that people who applied iterative and incremental development and agile quality assurance practices had a high success rate. In addition, people who applied governance practices also applied agile quality assurance practices but there was not much emphasis on high communication with the customers. We have to be careful here as only two practices; co-location and active stakeholder participation contributed to the communication with the customer factor. Communication with the team factor had a positive relation with governance and agile quality assurance practices. A negative but not significant relation was found between traditional quality assurance and agile quality assurance. This maybe because agile projects have tended to abandon more traditional quality assurance practices as they move more towards agile quality assurance. Interestingly, success rate related negatively with traditional analysis methods such as Gantt chart and detailed requirements specification.

7.6 Related Work

This section will look at the related work conducted by other researchers. The 2006 survey was reanalysed by Parsons and Lal (Parsons et al. 2007). The analysis compared the impact on outcomes when using no agile methods with the outcomes when using at least one agile method. The analysis findings suggested that the adoption of at least one agile method improves the outcomes of quality, satisfaction, and productivity over the use of non-agile methods, without a statistically significant increase in cost. We analysed the data differently as we can argue that when a company is not using any named agile method, this does not mean that they are not using agile software development. The survey results support our claim as the number of responses who said that they are not using any agile method (59%) is larger than the number of respondents who did not use any agile techniques (34%).

It worth mention that factor analysis was used in a study conducted by So and Scholl (So et al. 2009). The paper presented a measurement instrument to study the social-psychological effect of eight agile practices. The practices were chosen by the researchers, then qualitative methods were used to produce a set of items for each practice which formed a questionnaire. The factor analysis, namely principal component analysis, was used to test the validity of the existed factors structure. In other words, the analysis was used to check whether the extracted factors will be the same factors (practices) introduced by the researcher. In our case, the analysis was used for a different purpose, as we did not have an initial list of factors, instead the analysis extracted 15 new factors that were identified and named by us. This restructured a large set of practices into a smaller set of factors, which made applying further analysis much easier.

Regarding the impact of agile methods on stakeholder satisfaction, an empirical study tested five hypotheses that related agile characteristics with stakeholder's satisfaction. The study was conducted using a questionnaire and collected 59 responses from a South African organisation. The data was analysed using correlations and t-test, however the selected tests were not justified, i.e. the authors did not provide any information about the normality of the data. The study concluded that agile practices, namely iterative development, continuous integration, collective ownership, test driven development and feedback, has good impact on stakeholder's satisfaction (Ferreira et al. 2008).

7.7 Validity Issues

In this chapter, we re-analysed data from existing surveys. Although the researcher did not collect the data, this survey was conducted by a well-known and respected researcher within the agile community. However, the data still has the same limitations as any survey, mainly, the collected data is self-reported, and poor memory or misunderstanding of the questions can all contribute to inaccuracies in the data (Nardi 2002). One important issue to discuss is that as the data is based on the respondent's opinions; one threat to the factor analysis results could be that people may have rated agile practices based on how effective they think they are rather than reporting their real experience.

7.8 Summary

In this chapter, three existed agile adoption surveys were reanalysed. The findings suggested that there is a statistically significant relationship between agile methods impact on quality and their impact on satisfaction. In addition, the results showed that as productivity improves, quality and satisfaction improve and vice versa. The interesting finding is that whenever agile helped improve one of the aspects: quality, satisfaction and productivity it reduced cost and this was statistically significant in both 2006 and 2008 surveys. In addition, the results of the 2008 survey showed that when success rate went up cost was reduced. The surveys results supported two of the research hypotheses H1: agile software development can achieve customer satisfaction, and H6: agile software development can produce good quality software.

Furthermore, factor analysis was applied on a set of data that studied the effectiveness of 58 different agile practices. The analysis extracted 15 factors; each was associated with a list of practices. These factors with the associated practices can be used as a guide for agile process improvement. Correlations between the extracted factors were calculated, and the findings suggested that people who applied iterative and incremental development and quality assurance practices had a high success rate. Communication with the customer was not very popular according to the results as it had negative correlations with governance and iterative and incremental development. People who applied governance practices also applied quality assurance practices. Interestingly success rate related negatively with traditionally analysis methods such as Gantt chart and detailed requirements specification.

Chapter 8

Agile Projects Governance Survey

8.1 Introduction

At the early stage of this research, the research questions were focused on the impact of agile methods on software quality and customer satisfaction. The repeated iteration monitor survey and the deep analysis of the agile adoption surveys helped supporting five of the research hypotheses. Furthermore, the quantitative analysis of the previous surveys produced new questions regarding agile projects governance. Does it exist? How people are governing agile projects? and how does their experience differ from IBM teams' experience?

In order to answer these questions, the agile projects governance survey was conducted. The main purpose of this survey was to investigate agile projects governance by collecting data about how people are monitoring the progress of projects developed using agile methods, practices and principles according to the agile manifesto (Highsmith et al. 2001)

The survey was particularly interested in projects using agile retrospectives, reflection meetings, and metrics. This chapter will start with a brief literature review about IT governance and existing work on agile software governance. Then it will present the agile projects governance survey, its design and analysis, and it will present the results that describe the agile projects governance state of the art.

8.1 Information Technology Governance

According to the Compact Advanced Learner's Dictionary (governance 2009), the word governance (noun) is the action or manner of governing. IT governance is emerging as an important area by academics and practitioners (Webb et al. 2006). IT Governance has evolved from corporate governance, strategic information systems, and strategic information systems planning. A systematic definition of IT governance was suggested by Webb and Pollard (Webb et al. 2006) as "*the strategic alignment of IT with the business such that maximum business value is achieved through the development and maintenance of effective IT control and accountability, performance management and risk management*". The previous definition was the result of analysing twelve definitions found in the literature, which revealed five elements that constitute IT governance all were included in the proposed definition. Given different strategies and organisational structures, governance arrangements can vary from centralised approaches to decentralised ones or a hybrid approach that balance the first two (Weill et al. 2004).

According to a report by the IT Governance Institute, IT governance, like other governance subjects, is "the *responsibility of the board and executives*". In addition, the report mentioned that IT governance usually occurs at different layers, including team leaders reporting to and receiving direction from their managers, managers reporting up to the executive, and the executive to the board of directors (Report 2003).

8.1.1 *Governance of Agile Software Development Projects*

After a brief look at the general aspect of IT governance, we will investigate IT governance for projects that use agile software development. The first observation is that when studying governance for agile projects we will face the same challenges when we studied quality assurance during the literature review phase. As this is not the focus of this research, we will limit our discussions to reviewing the existing research on agile projects governance. However, we will consider investigating how agile software development can benefit from the existing governance models in our future work.

In an article published in Dr. Dobb's Portal in 2007 (Ambler 2007) the author stated that " *it is a lot easier to govern agile projects than traditional ones*". He supported this with two reasons, both are related to stakeholders' involvement, the first is that producing working software on a regular basis will give the stakeholders a great visibility to the work done by the team. Second, as the stakeholders are controlling the budget and schedule, they will direct the team effectively.

A workshop on software development governance has been running since 2008 within the international conference on software engineering (ICSE). The workshop focuses on the implementation of governance through tools and techniques in order to provide teams and organisations with the ability to effectively steer the business of software development. (Dubinsky et al. 2009). Governance for agile teams was included as one of the workshop themes. Two papers from this workshop will be discussed here. The first one studied the software development challenges that faced a company in an agile transaction. The study concluded that two challenges were related directly to functionality, lack of feedback loops, and lack of business theme prioritization. The rest were related to the company's transition to agile methods (Lehto et al. 2009). The second paper analysed three governance events within a project that implemented agile practices including studying the metrics that triggered each event. The study concluded that governance iterations can be unified within agile development iterations which can be useful in identifying issues and resolving them in an effective and timely manner (Talby et al. 2009). We can argue that this idea is similar to the iteration monitor proposed in chapter 6, as a tool to govern the project iteration by iteration.

It can be easily observed that the research focusing on governance for agile projects is still in its early stages. Therefore, the agile projects governance survey will help understanding the topic by collecting data about what we think is related to agile projects governance.

8.2 The Method

The agile projects governance survey is a web-based survey. It was designed and run using the SurveyMonkey online survey tool. The questions were generated based on the undertading and information gathered about IT governance and agile governance and on the questions raised during our previous studies, mainly, the iteration monitor survey and the agile adoption survey (Ambler 2007).

The survey has been reviewed and approved by the University of Southampton Ethical committee. The collected data was kept confidential and were used for research purpose only. The data and the results are anonymous; therefore, it was not possible to identify people, organisations, or projects from the data or from the results.

The survey was available online during the month of September 2009. The survey was sent to all the major agile mailing lists available on Yahoo and Google groups. Also it was sent to Facebook agile groups. The message included an introduction to the survey, its purpose and goals with an indication that the survey is anonymous and a web link to the survey was included.

8.2.1 *The Design*

The survey consisted of three sections: gathering information about the respondent, their current or most recent project, and agile governance.

The "who you are" section asked about the respondent's position, experience, his/her organisation sector, size, and how long he/she has been involved in agile development. Each question had a list of options that sometimes included an “other” option. An open text box was not included as required from the ethical committee so the identity of the respondents is protected. The purpose of this section is to understand the respondents’ background and their experience so existing relationships between different variables could be investigated.

The “on your current or most recent project” section asked specific questions about a project including the team size, the iteration and project length, the quality of the code produced, whether the project measured customer satisfaction and whether the project was successful. This section was included to get accurate data on the impact of agile code quality and project success. Therefore, it was necessary to collect information about a specific project rather than asking the question generally. This data will allow studying the impact of agile methods on project success, and software quality. Furthermore, it will help exploring the relations between project success and each of project length, team size, and iteration length.

A list of options followed each question. Ranges were used for the team size, iteration and project lengths, while scales were used for the other questions. Regarding the project success, it is always difficult to define success as it is different from a project to another, sector to another, and it is subjective. Therefore, the survey left it to the respondent’s judgment, and gave the respondent a scale range from “definitely” to “clearly failed” including a “too early to say” option for projects that have just started.

The last section was about agile governance which is the main purpose of the survey. As no previous survey focused on this aspect, it is important to first explore the state of the art of agile governance, particularly, the use of retrospectives and metrics in agile projects. Most importantly, this data can be useful in identifying relationships and connections which may help providing advice so teams can improve the use of these techniques. The section started with a brief description of an agile retrospective as a meeting held at regular intervals where the team reflect on what went well and what did not and how to become more effective in future iterations/sprints. The questions in this section focused on retrospectives and reflection meetings, their frequency, length, comments recording, team participation and their impact on the team practices. Also, the section asked whether the respondent’s organisation collects any metrics. This question allowed multiple answers, as an organisation may have more than one way to collect metrics, automatic, manual, part of the process or may have tried it and found it useless. The final question presented a list of metrics and asked whether the respondent’s company is using them. Each metric had a five point scale ranging from always to never.

Both phrases, agile retrospective and reflection meetings were used so the questions are as clear as possible, which will lead to more accurate and suitable responses. The phrases are assumed to be synonymous.

The survey has received 129 responses. All respondents completed the first section, 117 completed the first and the second section, and 106 completed all three sections.

8.2.2 *The Analysis*

The survey tool provides the data files in Excel, CVS formats, as well as a coded (numerical) data set. The coded file was revised and changed as needed. This included recoding some questions so all the question follow the same coding pattern. This means the size is coded so the small code presents the smaller size, and for the scales, the smaller code presents bad quality, less frequently, the never option and so on. As usual, all the frequencies were calculated after coding and checked against the results to make sure the coding was done correctly. The data set was stored as a SPSS file so further analysis can be performed.

As most of the data is nominal, or ordinal with a “not applicable” option, the most suitable way of presenting the results is via frequencies. The mean was not used because there were no interval data. It was possible to use the mean for the ordinal data such as the Likert-scale which looks like interval. However, the standard deviations were big relatively to their means so frequencies were used. The data was presented using tables as well as graphs in most cases so it helps the reader understanding the results. In the results tables, the highest percentage is marked with a bold font and frame. When mentioning the percentage in the text, it was rounded to the nearest tenth for easier reading, and the exact values are presented in the tables and the graphs.

The data was not normally distributed so Spearman’s correlation was used to study the relationships between the different variables.

For questions that allowed multiple answers, cross tabulation was used. Cross tabulation displays the joint distribution of two or more variables, it is easy to use and can be used with any type of data. The SurveyMonkey online tool provides this feature.

8.3 The Results

The results section will present the survey findings using descriptive statistics for each section and then it will discuss any existing relationship between the different variables.

8.3.1 *Section 1: Respondent's Background*

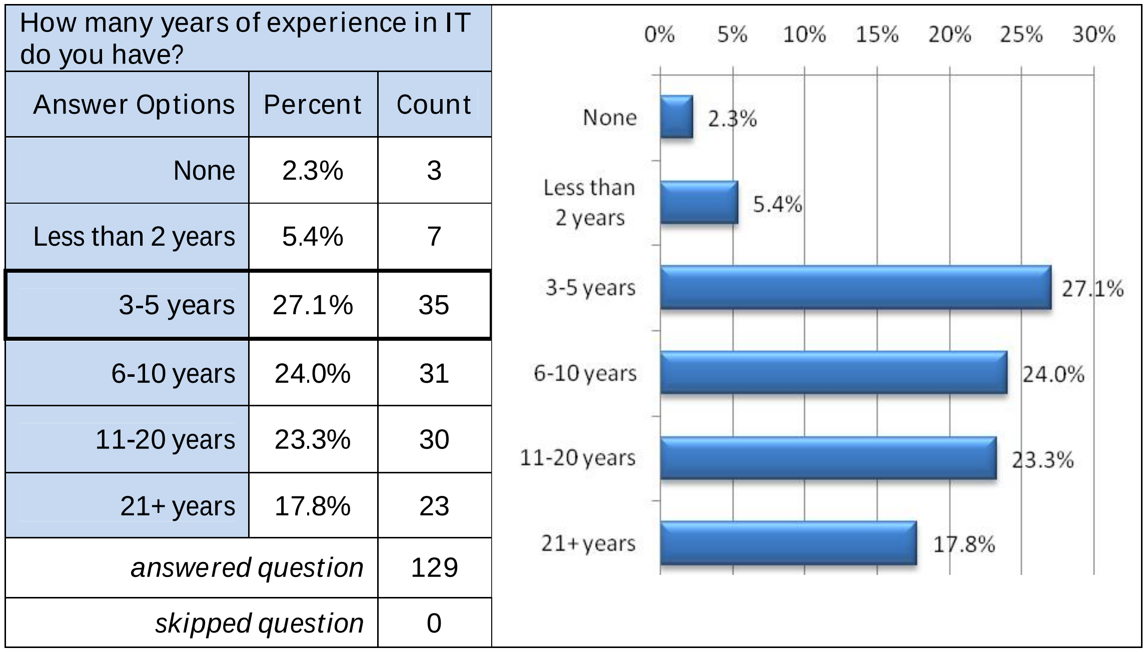
The purpose of this section is to collect information about the survey participants and their backgrounds. The majority (26%) were developers and the minority (4%) were business stakeholders.

Interestingly, 15% of the respondents chose "other", which indicates that the survey missed a number of available positions, for example a number of consultants replied to the survey email to report that their positions were not included. This feedback will be important in future surveys. Table 8-1 gives the data collected about the respondents' positions.

What describes best your current position in the organisation?		
Answer Options	Percent	Count
Business Stakeholder	3.9%	5
Developer	25.6%	33
Scrum Master	10.1%	13
Project Manager	14.0%	18
Tester	13.2%	17
Quality Assurance	12.4%	16
Architect	5.4%	7
Other	15.5%	20
answered question		129
skipped question		0

Table 8-1 Respondents' positions in their organisations

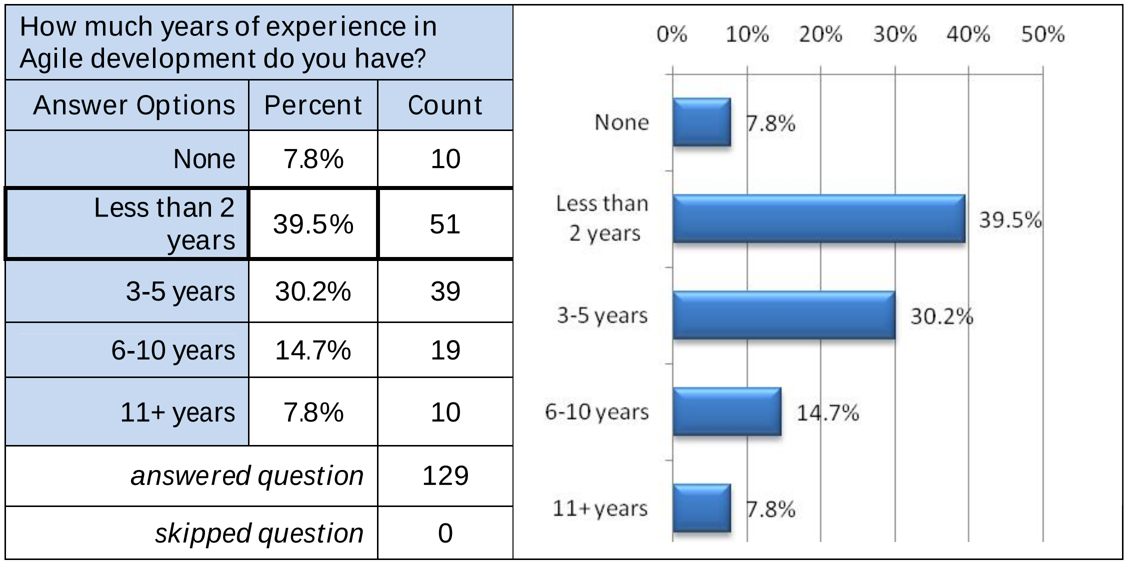
The respondents had more experience in IT than in agile. As can be seen in figure 8-1, 27% of the respondents had 3-5 years of experience in IT, and 41% had 11+ years of experience.



How many years of experience in IT do you have?		
Answer Options	Percent	Count
None	2.3%	3
Less than 2 years	5.4%	7
3-5 years	27.1%	35
6-10 years	24.0%	31
11-20 years	23.3%	30
21+ years	17.8%	23
answered question		129
skipped question		0

Figure 8-1 *Respondents' experience in IT*

On the other hand, when asking about agile development experience, 39% of the respondents had less than 2 years of agile experience, whereas 8% had +11 years experience. The respondents' agile experience results are presented in figure 8-2.



How much years of experience in Agile development do you have?		
Answer Options	Percent	Count
None	7.8%	10
Less than 2 years	39.5%	51
3-5 years	30.2%	39
6-10 years	14.7%	19
11+ years	7.8%	10
answered question		129
skipped question		0

Figure 8-2 *Respondents' experience in agile development*

The respondents came from organisations varying in size, sector, and experience in agile. Regarding size, 28% worked in organisation of 1000+ people. Detailed results can be seen in figure 8-3.

What is the total number of people in your organisation?		
Answer Options	Percent	Count
1-10	17.1%	22
11-100	23.3%	30
101-1000	31.8%	41
1001-10000	18.6%	24
10001-100000	5.4%	7
Over 100000	3.9%	5
answered question		129
skipped question		0

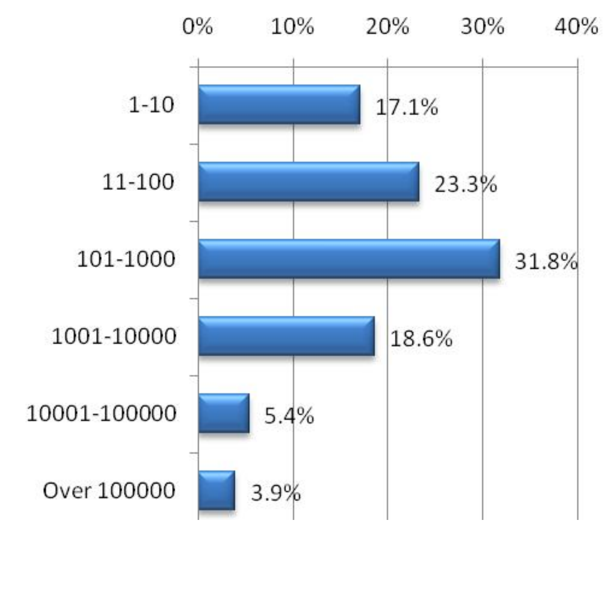


Figure 8-3 Respondents' organisations size

When asking about the organisation sector, 32% of the respondents worked with software and 25% with IT services. Also, 15% chose other, which means that more options should have been included in the list. The sector results are shown in figure 8-4.

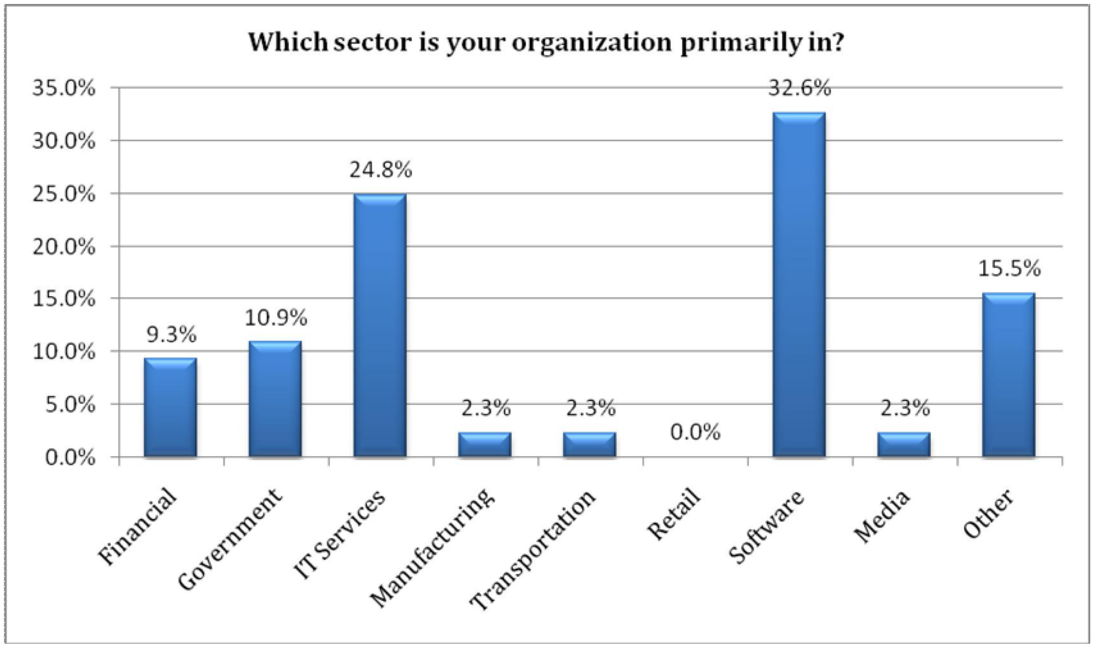


Figure 8-4 Respondents' organisations sectors

Most of the respondent's organisations are relatively new to agile development, as 53% have been using agile methods for 2 years of less. Figure 8-5 shows the organisation agile experience results.

How long your company has been doing agile development?		
Answer Options	Percent	Count
We have no agile experience	11.6%	15
Less than 1 year	11.6%	15
1-2 years	30.2%	39
3-5 years	29.5%	38
6-10 years	10.1%	13
11+ years	7.0%	9
answered question		129
skipped question		0

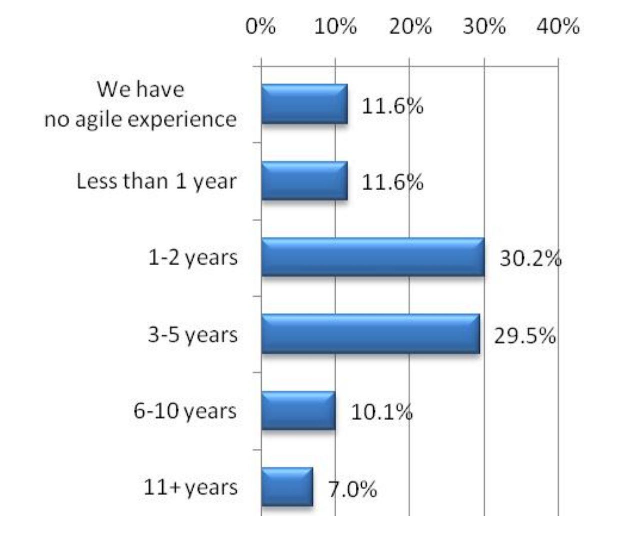


Figure 8-5 *Respondents organisations experience in agile*

8.3.2 *Section 2: Information about a Specific Project*

This section asked about a specific agile project; its length, team size, iteration length, whether it measured customer satisfaction, its code quality and success. The reader should keep in mind that 117 respondents completed this section, compared to 129 who completed the previous section. Most of the respondents had a small team size, as shown in figure 8-6, 44% had a team of (6-10) people, and 33% had a team of (1-5) people.

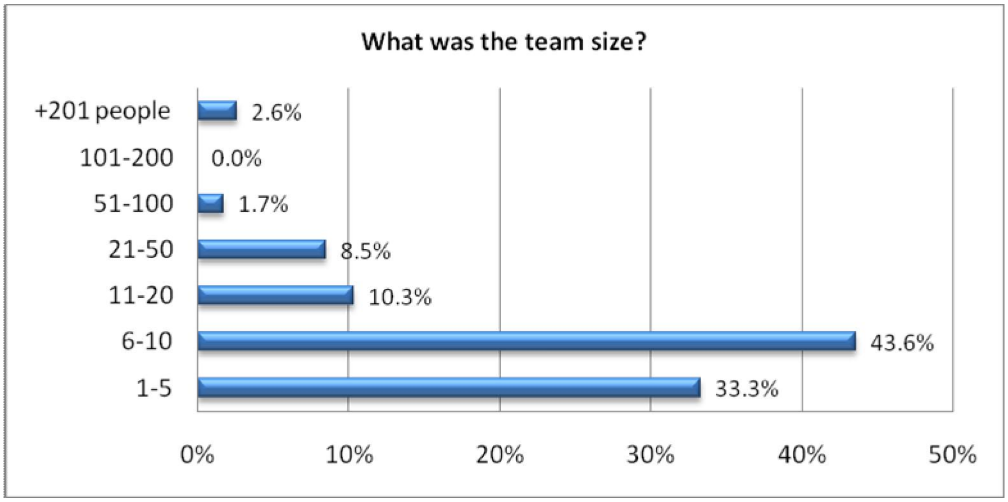


Figure 8-6 Respondents team size

Regarding iteration length, short iterations were more popular, 2 weeks length was the most popular option with 41%. In second place came the 4 weeks length with 19%. The project length varied from +24 months (17%) to 3-6 months (24%) and 19% were less than 3 months old. This indicates that most of the projects were either short or they were at an early stage. The iteration and project length results are shown in figures 8-7 and 8-8 respectively.

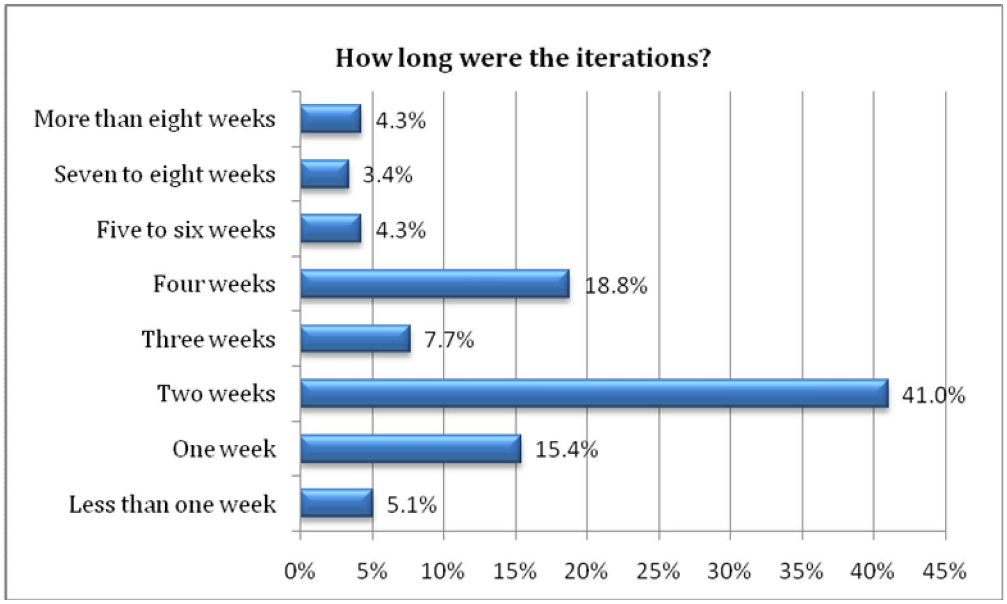


Figure 8-7 Iteration length

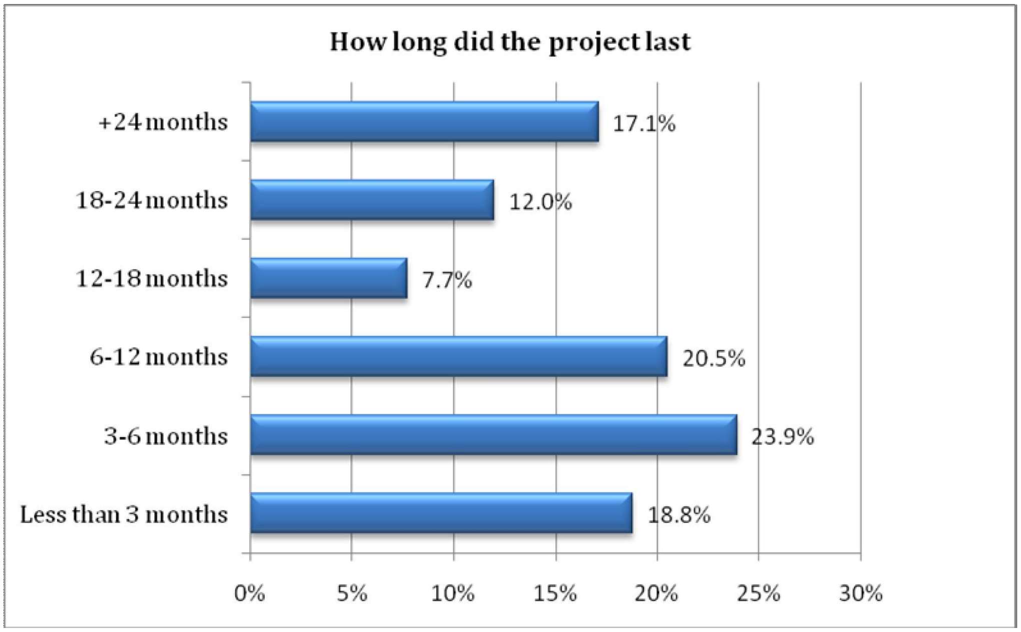


Figure 8-8 Project length

After the team size, iteration and project length questions, the survey asked how often the team measured customer satisfaction. The results are very encouraging as 72% of the respondents measured customer satisfaction often or always. The remaining percentages can be seen in figure 8-9.

How often do you measure customer satisfaction?		
Answer Options	Percent	Count
Always	27.4%	32
Often	35.0%	41
Sometimes	23.1%	27
Rarely	10.3%	12
Never	4.3%	5
answered question		117
skipped question		12

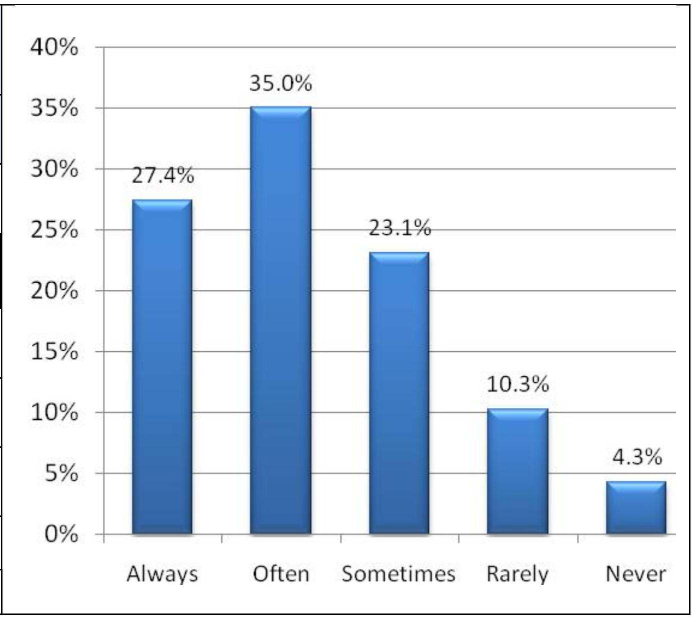


Figure 8-9 The frequency of measuring customer satisfaction

When asking about how the respondents rated code quality, 65% reported high or above code quality which is again very encouraging. Also these results support hypothesis H6: agile software development can produce good quality software. The code quality results are illustrated in figure 8-10.

How do you rate the code quality?		
Answer Options	Percent	Count
Very high	12.0%	14
high	53.0%	62
Average	28.2%	33
Low	6.0%	7
Very low	0.9%	1
answered question		117
skipped question		12

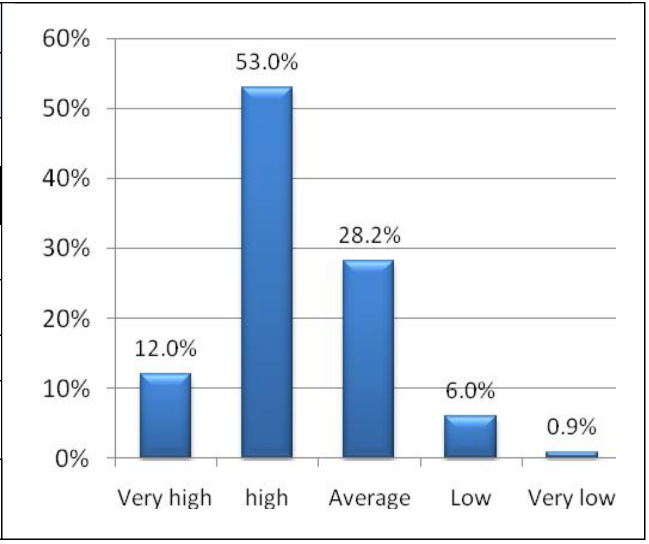


Figure 8-10 The code quality

The final question in this section is about project success. The results showed that 45% reported their projects were definitely successful, whereas 1.7% stated that their projects clearly failed. Detailed results are shown in figure 8-11.

Was the project successful?		
Answer Options	Percent	Count
Definitely	45.3%	53
Somewhat	30.8%	36
Partially	13.7%	16
Clearly failed	1.7%	2
Too early to say	8.5%	10
answered question		117
skipped question		12

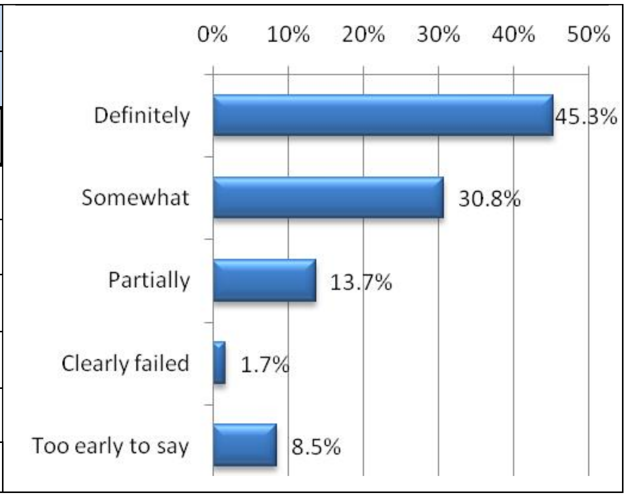


Figure 8-11 Project success

8.3.3 *Section 3: Agile Governance*

In this section, the survey asked about the use of agile retrospectives in the team and the use of metrics within the organisation. Generally, the results were encouraging and it indicates a high level of governance within the projects and teams that responded to the survey. 65% of the teams performed retrospectives after each iteration, and 19% performed one when they had problems. The full results are shown in figure 8-12.

When do you perform your retrospectives/reflections meeting?		
Answer Options	Percent	Count
After each iteration	65.1%	69
Every other iteration	9.4%	10
When we have problems	19.8%	21
After each release sometimes	14.2%	15
Rarely	6.6%	7
Never	3.8%	4
answered question		106
skipped question		23

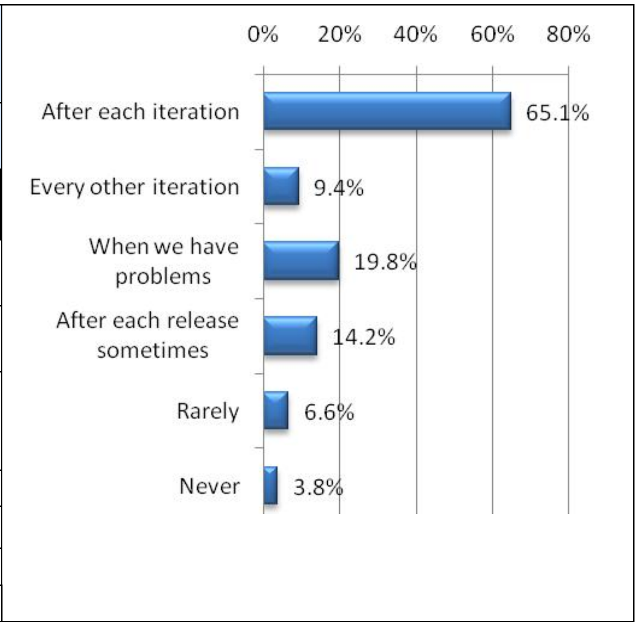


Figure 8-12 Frequency of retrospectives

The next question asked whether the team recorded the comments they made during the retrospective. The results showed that 50% of the respondents always recorded the comments and 19% often recorded them. This indicates that the culture of documentation does exist within the agile development environment, even though this was on one specific aspect. The full results of this question are presented in figure 8-13.

Do you have a record of the comments that we made during the retrospective/reflection meeting?		
Answer Options	Percent	Count
Always	50.0%	53
Often	18.9%	20
Sometimes	17.9%	19
Rarely	7.5%	8
Never	1.9%	2
N/A	3.8%	4
answered question		106
skipped question		23

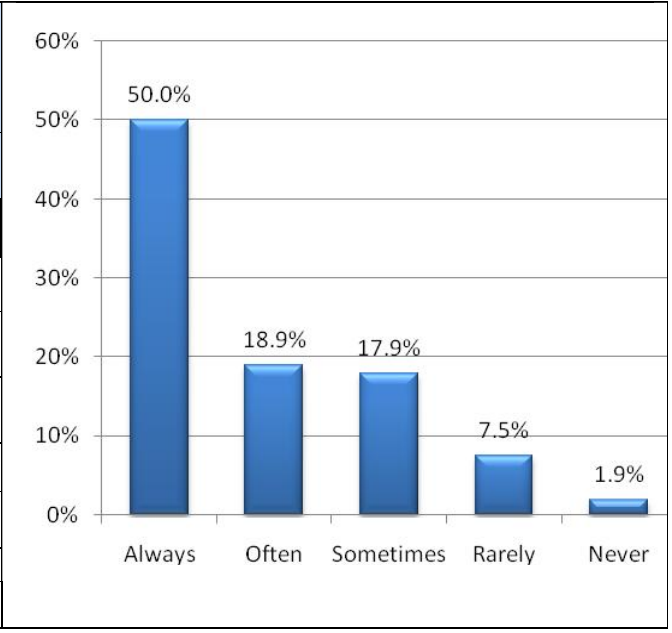


Figure 8-13 Recording retrospective comments

The following question asked about the length of the retrospective; 48% had a retrospective that lasted 1-2 hours, and 41% had less than an hour meeting.

The next two questions focused on team participation in the reflection meeting, first the survey asked whether the whole team participated in the reflection meeting, and 53% said always, and 21% answered with often. The results are presented in figure 8-14.

Does the whole team participate in the retrospective/reflection meeting?		
Answer Options	Percent	Count
Always	52.8%	56
Often	20.8%	22
Sometimes	13.2%	14
Rarely	5.7%	6
Never	3.8%	4
N/A	3.8%	4
answered question		106
skipped question		23

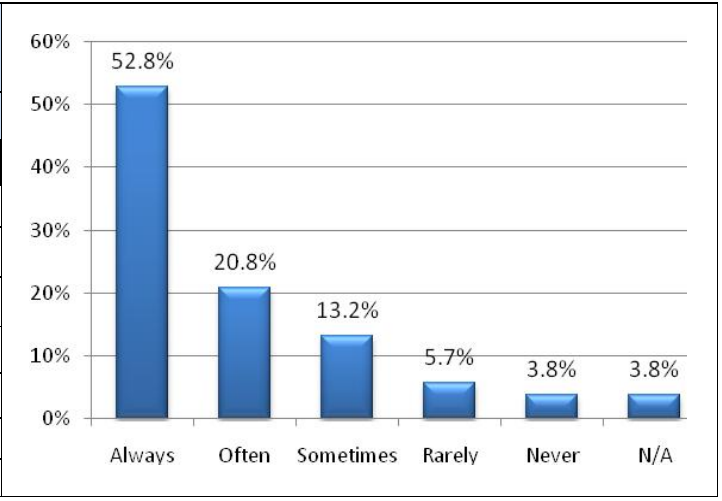


Figure 8-14 The team participation in the retrospective

Moreover, when asking whether everyone in the team had their say in the meeting, 72% of the respondent said always and 16% said often. So almost everybody had his or her say during the meeting which is very healthy for an agile environment. The question results are presents in figure 8-15.

Can everyone have their say in the meeting?		
Answer Options	Percent	Count
Always	71.7%	76
Often	16.0%	17
Sometimes	5.7%	6
Rarely	0.0%	0
Never	1.9%	2
N/A	4.7%	5
answered question		106
skipped question		23

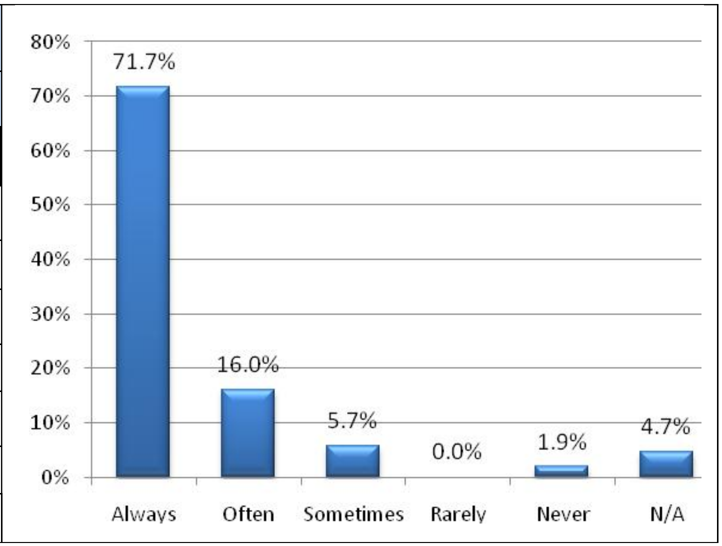


Figure 8-15 The team individual participation in retrospective

The last retrospective question asked whether these meetings changed the team practices. The results were spread as 39% said that the retrospective sometimes did affect their practices, 35% said that it often did, and 15% said it always did. Small percentages (4%, 5%) said that retrospective rarely or never changes their practices. These results suggest that using agile retrospectives helps agile teams to improve their process. Detailed results are presented in figure 8-16.

Does the retrospective change your practices?		
Answer Options	Percent	Count
Always	15.1%	16
Often	34.9%	37
Sometimes	38.7%	41
Rarely	2.8%	3
Never	3.8%	4
N/A	4.7%	5
answered question		106
skipped question		23

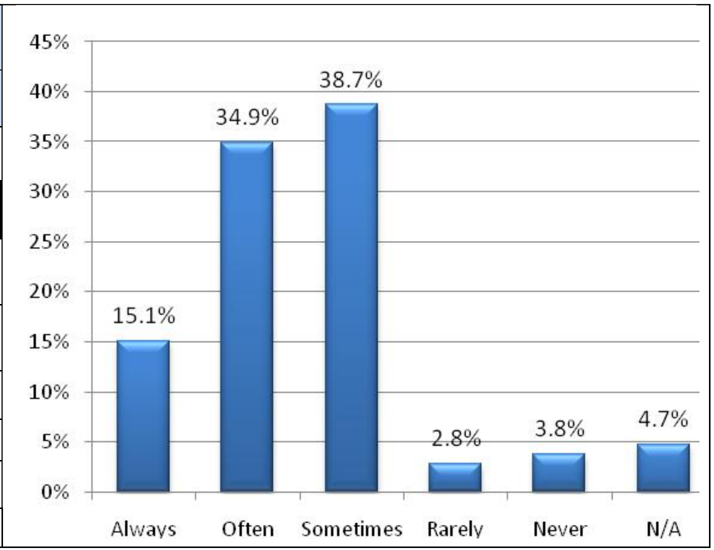


Figure 8-16 The impact of agile retrospective on team's practices

The final two questions in the survey, focused on metrics and measurements. First, the survey asked the respondents if their companies collect any metrics or measures. This question allowed multiple answers as the company might apply more than one provided technique. The results reported that 57% of the respondents generate metrics manually, whereas 38% generate metrics automatically. Detailed results are presented in figure 8-17.

In your company, do you collect any measures or metrics? (multiple answers are allowed)		
Answer Options	Percent	Count
We automatically generate metrics using tools	37.7%	40
We manually generate metrics	57.5%	61
We tried collecting metrics but we found them useless	12.3%	13
We have to, it is part of our process	15.1%	16
Do not know	13.2%	14
answered question		106
skipped question		23

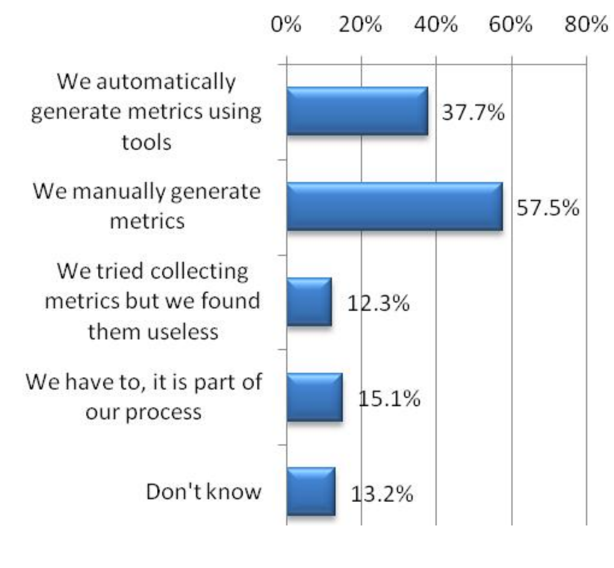


Figure 8-17 Metrics collection within the respondents company

The final question in the survey, gave the respondents a list of available metrics, mainly “agile metrics”. The results showed that the participants were aware of the provided metrics as the percentages for the “never” option were relatively low. In addition, the frequency of applying the measurement varied depending on the metric, but generally, 6 metrics were used by 50% of the respondents at least always or often. The responses count and percentage are presented in table 8-2 and figure 8-18 respectively.

In your company, do you measure/use any of the following?					
Answer Options	Always	Often	Sometimes	Rarely	Never
Burn charts	39	14	18	7	28
Story points	43	15	13	9	26
Functions points	12	10	14	10	60
Team velocity	39	19	19	9	20
Business value delivered	35	15	21	8	27
RTF: Running testing features	31	14	14	9	38
Defect count after testing	38	19	16	6	27
Number of Test cases	34	16	15	12	29
TTOR: Time to obstacle removal	10	12	14	12	58
OR/I: obstacles removed per iteration	8	11	14	12	61

Table 8-2 The frequency of the use of different metrics (count)

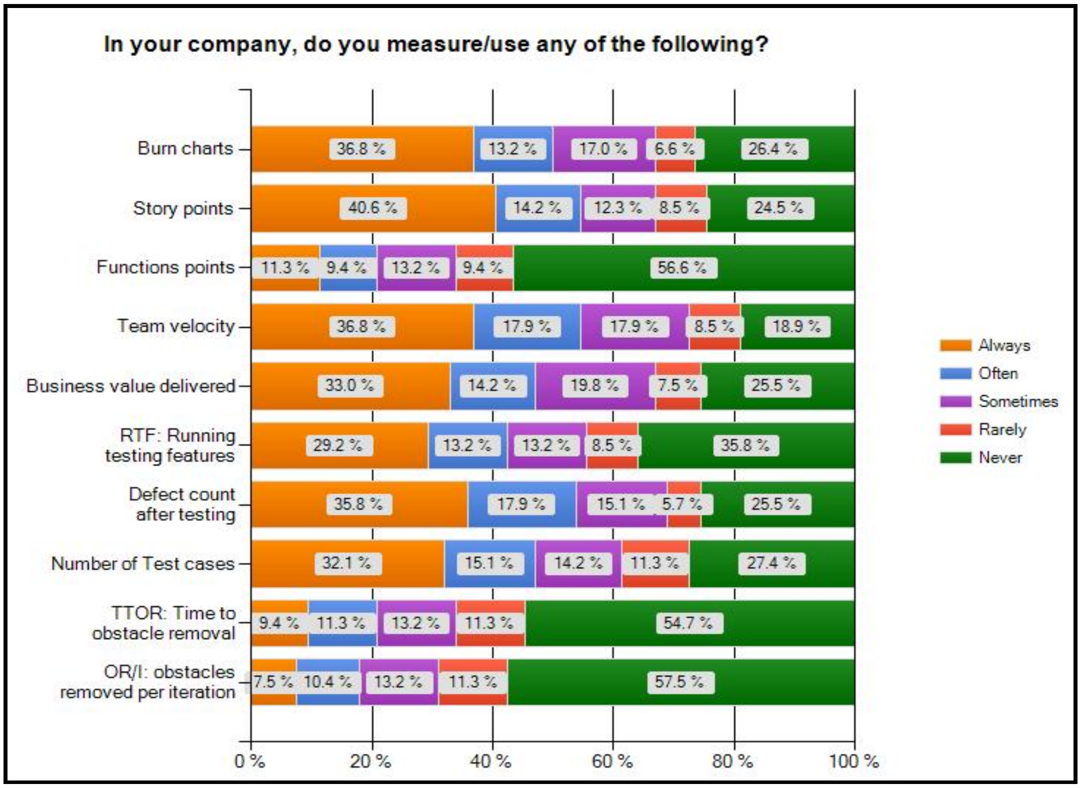


Figure 8-18 The frequency of the use of different metrics (percentage)

Interestingly, when asked whether the company collected any metrics in the previous question, the results showed higher percentage (57% for manual collection and 38% for the automatic) than the specific metrics question. This could be because the respondents used different metrics that were not included in the list, or because the frequency of their use varied. In other words, when asked about the use of metrics the options did not ask about the frequency of the measure. For example, if we add both "always" and "often" percentages together from the specific metrics question, they will cover at least 50% of the respondents as mentioned before, which agrees more with the previous question results.

8.4 Relationships between Survey Variables

The previous section presented the descriptive results that illustrated the state of the art of agile governance, in particular, agile retrospectives, and the use of metrics in agile projects. This section will explore any existing relationships between the different variables studied in the survey. As there are many variables and many interesting correlations, the results will be discussed in four categories, each covering a number of related variables:

- Organisation variables (size, experience in agile development)
- Project variables (length, iteration length, team size, success, code quality, frequency of customer satisfaction measure)

- Retrospective variables (frequency, length, records, impact, team participation, team contribution)
- Metrics variables (general collection, usage of each metric)

The following subsections will discuss each category, the relationships within the category variables, and their relationships with other categories' variables.

8.4.1 *Organisation Variables*

In this section, we investigate whether organisation's size and/or agile development experience have any impact on project success, code quality or the measure of customer satisfaction. This investigation will help testing one of the thesis hypotheses H26: agile development maturity increases over time. Spearman correlation coefficient was applied on the coded data, and SPSS output is presented in table 8-3. Regarding the organisation size the following significant correlations occur:

- There is a positive relationship between organisation size and each of
  - Team size, *rs* =.38, (*p* <0.01) and also
  - Iteration length, *rs* =.30, (*p* <0.01)
  - There is a negative relationship between organisation size and Project success, *rs* =.26, (*p* <0.01)

	Organisation Size	Organisation Agile Experience
Organisation Size	1.000	.038
Organisation Agile Experience	.038	1.000
Team Size	.387**	.058
Iteration Length	.308**	.007
Project Length	.122	-.111
Customer Satisfaction Measure	-.096	.298**
Code Quality	-.099	.204*
Project Success	-.267**	.402**
Retrospective Impact	-.111	.211*
Retrospective Contribution	-.091	.339**
Retrospective Participation	-.084	.289**
Retrospective Records	.134	.055
** Correlation is significant at the 0.01 level (2-tailed) * Correlation is significant at the 0.05 level (2-tailed)		

Table 8-3 Organisation variables correlations

Here we can see that large organisations tend to have bigger teams and longer iterations. Although this was for one project only, it could help understand the adoption of agile methods in large organisations. In addition, it seems that agile development is less successful with large organisation. This negative relation also was founded when the agile adoption survey 2008 was analysed as the correlation found (- .18 with p <0.01) between organisation size and project success. The same results were found by (Livermore 2007) who reported significant negative correlation between implementation success and the size of the organisation attempting to implement an agile methodology.

Regarding the organisation's experience with agile development the correlation table shows that:

- There is a positive relationship between organisation agile experience and the following variables:
  - The frequency of customer satisfaction measure, *rs* =.29, (*p* <0.01)
  - Code quality, *rs* =.20, (*p* <0.05)
  - Project Success, *rs* =.30, (*p* <0.01)
  - Retrospective impact, *rs* =.21, (*p* <0.05)
  - Retrospective contribution, *rs* =.33, (*p* <0.01)
  - Retrospective records, *rs* =.28, (*p* <0.01)

As expected, organisations with more agile experience are doing much better regarding project success and code quality. Also, more experienced organisations measure customer satisfaction more frequently. Applying agile retrospectives seems to be more mature in more experienced organisations. This can be seen as a sign of maturity, which supports hypothesis H26: the adoption of agile methods goes in stages and it improves over iterations, releases and projects.

8.4.2 *Project Variables:*

The purpose of the second section of the survey is to study one specific project as reported by each respondent. Applying Spearman correlation on project' variables produced table 8-4.

	Team Size	Iteration Length	Project Length	Customer Satisfaction Measure	Code Quality	Project Success
Team Size	1.000	.158	.128	.069	.121	-.060
Iteration Length		1.000	.380**	.007	-.123	-.133
Project Length			1.000	-.120	-.163	-.137
Customer Satisfaction Measures				1.000	.470**	.402**
Code Quality					1.000	.351**
Project Success						1.000

** Correlation is significant at the 0.01 level (2-tailed)

Table 8-4 Project variables correlations

- There is a positive relationship between iteration length and project length, *rs* =.38, (*p* <0.01).
- There is a positive relationship between the frequency of measuring customer satisfaction and
    - Code quality, *rs* =.47, (*p* <0.01)
    - Project success, rs =.40, (p <0.01)
- There is a positive relationship between code quality and project success, *rs* =.35, (*p* <0.01)

To summarise, the longer projects tend to have longer iterations. Also, the three factors, project success, code quality and measuring customer satisfaction are related positively; this relationship appeared multiple times throughout the thesis (see 7.3, 7.4) from different surveys, so it seems like these three factors are strongly related. The difference this time is that this survey asked about the frequency of measuring customer satisfaction and not the result of this measure. The point here is that measuring customer satisfaction once and getting good results is not good enough, instead the team is advised to measure customer satisfaction frequently. Interestingly, there is no significant relationship between team size and project success, yet it is a negative relationship (smaller teams tend to be more successful). Previous survey conducted by Livermore didn't find a significant relationship between these two factors (Livermore 2007). This comes in conflict with authors who argued that team size is an important factor and that agile methods do best with small teams (Cockburn et al. 2001b; Boehm et al. 2003; Beck et al. 2004; Cockburn 2005).

8.4.3 *Retrospective Variables*

The survey asked different questions about agile retrospectives. The first question asked when the team performed them. As this question allowed multiple answers, a more suitable way to analyse its relationship with other variables is cross tabulation. The Surveymonkey tool provides this feature. Applying cross tabulation on this question gave interesting results. Detailed results tables are included in Appendix G.

Out of the 69 respondents who performed retrospectives after each iteration:

- 45% had a team of 1-6 people and fewer than 4% respondents had a team over 50
- 42% often measured customer satisfaction
- 56% had high code quality
- 56% had definitely successful projects
- 42% said retrospectives changed the team practices

- 68% manually collected metrics

Out of the 21 respondents who performed retrospectives when the team had problems:

- 33% had a team of 1-6 people and fewer than 4% respondents had a team over 50
- 33% always measured customer satisfaction
- 57% had high code quality
- 57% had definitely successful projects
- 38% said retrospectives changed the team practices
- 48% manually collected metrics

These results suggest that retrospectives are more suitable for small teams. Also the results recommend performing retrospectives after each iteration for the best outcomes regarding code quality and project success. The results suggest that using retrospectives when the team has a problem can be useful. Furthermore, it seems like the teams who perform retrospectives are also applying metrics. In other words, when a team does it right, it does for all aspects.

Spearman correlation was used to study the relationships between the other retrospective variables. Table 8-5 presents the correlations between these different aspects and with the other variables in the survey.

- There is a positive relationship between retrospectives impact and the followings other variables
  - Customer satisfaction, *rs* =.31, (*p* <0.01).
  - Code quality, *rs* =.24, (*p* <0.05).
  - Project success, *rs* =.33, (*p* <0.01).
  - Retrospective contribution, *rs* =.24, (*p* <0.05).
  - Retrospective participation, *rs* =.44, (*p* <0.01).
  - Retrospective recording, *rs* =.27, (*p* <0.01).
- There is a positive relationship between retrospective contribution and the followings variables
  - Project success, *rs* =.27, (*p* <0.01)
  - Retrospective impact, *rs* =.24, (*p* <0.05)
  - Retrospective participation, rs =.54, (p <0.01)
- There is a negative relationship between retrospective contribution and iteration length, *rs* =.37, (*p* <0.01)

- There is a positive relationship between retrospective participation and the following variables
  - Project success, *rs* =.26, (*p* <0.05)
  - Retrospective impact, *rs* =.44, (*p* <0.01)
  - Retrospective contribution, *rs* =.54, (p <0.01)
  - Retrospective records, *rs* =.36, (p <0.01)

	Retrospective Impact	Retrospective Contribution	Retrospective Participation	Retrospective Records
Team Size	-.002	-.140	.001	.002
Iteration Length	-.119	-.375**	-.155	.057
Project Length	-.063	-.162	-.079	.022
Customer Satisfaction	.316**	.153	.139	.065
Code Quality	.240*	.185	.125	.102
Project Success	.334**	.271**	.262*	.039
Retrospective Impact	1.000	.248*	.440**	.275**
Retrospective Contribution	.248*	1.000	.543**	.172
Retrospective Participation	.440**	.543**	1.000	.376**
Retrospective Records	.275**	.172	.376**	1.000

** Correlation is significant at the 0.01 level (2-tailed)

* Correlation is significant at the 0.05 level (2-tailed)

Table 8-5 Retrospective variables correlations

These results suggest that agile retrospectives are effective when applied properly. As reported earlier, the retrospectives had more impact when the whole team participated, everybody had their say, and the retrospective comments were recorded. The findings suggested negative relations between team size and both retrospective impact and contribution, that supports our observation with IBM team where the iteration monitor was introduced to get more input from the team about the iteration as the team was getting bigger.

8.4.4 *Metrics Variables*

The survey asked two questions about metrics. The first one asked generally about the organisation policy regarding collecting metrics. This question allowed multiple answers, so similar to the retrospective frequency question, cross tabulation was applied. The complete crosstab results are included in Appendix G, and here are the results highlights:

Out of the 40 respondents, whose companies automatically collected metrics

- 32% always measured customer satisfaction
- 67% had high code quality
- 57% had definitely successful projects

Out of the 61 respondents, whose companies manually collected metrics:

- 43% always measured customer satisfaction
- 56% had high code quality
- 57% had definitely successful projects

The results show a relationship between collecting metrics and different variables; however, it is not as strong as expected.

Table 8-6 presents the results of applying Spearman correlation on the list of different metrics. The significant correlations suggest that:

- There is a positive relationship between burn charts and the following variables
    - Story points, *rs* =.64, (*p* <0.01)
    - Team velocity, *rs* =.57, (*p* <0.01)
    - Defect count after testing, *rs* =.21, (*p* <0.05)
- There is a positive relationship between story points and team velocity, *rs* =.61, (*p* <0.01)

	Burn Charts	Story Points	Function Points	Team Velocity	BVD	RTF	Defect Count After Testing	Number of Test Cases	TTOR	OR/I
Burn Charts	1.000	.640**	.091	.579**	.109	.159	.217*	.094	.094	.062
Story Points		1.000	.112	.615**	.052	.052	.052	.019	-.016	-.010
Function Points			1.000	-.026	.219*	.258**	.188	.318**	.464**	.551**
Team Velocity				1.000	.270**	.207*	.132	.064	.045	.063
BVD					1.000	.370**	.333**	.220*	.406**	.450**
RTF						1.000	.400**	.474**	.450**	.403**
Defect Count AT							1.000	.578**	.327**	.322**
Test Cases								1.000	.298**	.363**
TTOR									1.000	.820**
OR/I										1.000

** Correlation is significant at the 0.01 level (2-tailed)

* Correlation is significant at the 0.05 level (2-tailed)

Table 8-6 Metrics correlations

- There is a positive relationship between function points and the following variables
  - Business value, *rs* =.21, (*p* <0.05)
  - Running testing features, *rs* =.25, (*p* <0.01)
  - Number of test cases, *rs* =.31, (*p* <0.01)
  - Time to obstacle removal, *rs* =.46, (*p* <0.01)
  - Obstacles removed per iteration, *rs* =.55, (*p* <0.01)
- There is a positive relationship between team velocity and the following variables
  - Business value, *rs* =.27, (*p* <0.01)
  - Running testing features, *rs* =.20, (*p* <0.05)
- There is a positive relationship between business value delivered and the following variables
  - Running testing features, *rs* =.37, (*p* <0.01)
  - Defect count after testing, *rs* =.33, (*p* <0.01)
  - Number of test cases, *rs* =.22, (*p* <0.05)
  - Time to obstacle removal, *rs* =.40, (*p* <0.01)
  - Obstacles removed per iteration, *rs* =.45, (*p* <0.01)
- There is a positive relationship between Running testing features and the following variables
  - Defect count after testing, *rs* =.40, (*p* <0.01)
  - Number of test cases, *rs* =.47, (*p* <0.01)
  - Time to obstacle removal, *rs* =.45, (*p* <0.01)
  - Obstacles removed per iteration, *rs* =.40, (*p* <0.01)
- There is a positive relationship between defects count after testing and the following variables
  - Number of test cases, *rs* =.57, (*p* <0.01)
  - Time to obstacle removal, *rs* =.32, (*p* <0.01)
  - Obstacles removed per iteration, *rs* =.32, (*p* <0.01)
- There is a positive relationship between number of test cases and the following variables
  - Time to obstacle removal, *rs* =.29, (*p* <0.01)
  - Obstacles removed per iteration, *rs* =.36, (*p* <0.01)
  - Defect count after testing, *rs* =.32, (*p* <0.01)
- There is a positive significant relationship between number of test cases and obstacles removed per iteration, *rs* =.82, (*p* <0.01)

The results suggest that when a team is collecting metrics, it does not collect only one. For example the team who measured the “business value delivered” metric they also measured every other metric provided apart from the first two. In addition, although the function points metric is a traditional one, teams who collected it also collected agile metrics.

Finally, correlation was used to test the relationship between project success and the different metrics, the results showed that there is a positive significant relationship between project success and the following metrics:

- Team velocity, , *rs* =.36, (*p* <0.01)
- Business value delivered, , *rs* =.40, (*p* <0.01)
- Running testing Features, , *rs*=.23, (*p* <0.05)
- Defect count after testing, *rs*=.31, (*p* <0.01)
- Number of test cases, *rs*=.22, (*p* <0.05)

There was no other significant relationship with the other metrics.

8.4.5 *Project Success Factors*

The survey asked about project success, therefore it might be useful to cross tab this question and see which factors made a project succeed in the respondents opinion. Out of the 53 projects that were <u>definitely</u> successful, the factors with 60% or over will be considered here:

- 72% had teams of 10 people or less (small team size)
- 77% always or often measured customer satisfaction (frequent customer satisfaction measure) (supported by correlation results)
- 79% had high or very high code quality (high code quality) (supported by correlation results)
- 78% performed retrospective after each iteration (performing retrospectives after each iteration)
- 66% had the whole team participate in the retrospective (having the whole team participating in the retrospective) (supported by correlation results)
- 88% reported that everybody can have their say in the retrospective (team contribution retrospective) (supported by correlation results)
- 64% manually collected metrics (collecting metrics)
- 60% never measured OR/I (choosing the needed metrics for the project)

This list can be used by agile team as a guide to achieve success as it is based on a survey that collected 129 responses from people who had different positions, experiences, and who came from different teams and organisations.

A research paper studied the critical success factors in agile software development using a survey. The study collected data about 109 projects and used regression to analyse the data. The study listed six critical success factors for agile software development; each had a number of attributes. The suggested success factors are delivery strategy, agile software engineering techniques, team capacity, project management process, team environment, and customer involvement (Chow et al. 2008).

8.5 Reflection on Research Hypotheses

As mentioned during the data analysis, the agile projects governance survey results supported two of the research hypotheses:

H6: agile methods can produce good quality software: this hypothesis was supported as 65% of the respondents reported high or above when they were asked to rate code quality (not statistically significant).

H26: Agile adoption goes in stages and it improves over iterations, releases and projects (statistically significant). When analysing correlations between organisation experience and other variables, the results suggested that organisations with more agile experience are doing much better regarding project success and code quality. Also, more experienced organisations measure customer satisfaction more frequently.

8.6 Limitations

The great strength of survey research is that for a relatively little cost we can collect data about a number of variables from a large number of persons. Also, surveys are useful in collecting data on aspects of behaviour that are difficult to observe directly. However, surveys also have a number of limitations. The most serious weakness concerns the validity and reliability of responses obtained as the collected data is self-reported, and poor memory, or misunderstanding of the questions can all contribute to inaccuracies in the data (Nardi 2002).

Specifically to the agile projects governance survey, it was difficult to calculate the response rate as the survey was distributed to online email groups whose numbers are constantly changing and it is impossible to guarantee that all group members are active members in the group and thus had read the distributed email. Regarding the survey design, some questions were not very useful to the study such as the position and organisation sector. Especially since the position question got high number of responses choosing the "other" option which indicated that there are more positions that were not included in the options list. The analysis linked the questions about a specific project to general organisation questions that maybe or maybe not applied to that specific project. However, it was difficult to ask all questions about one project, as it was needed to understand the organisation strategy regarding agile governance. In addition, in order to get more accurate results, the survey should have focused on either current or recent project so the data can give a better idea about the project length and its relation to other variables. Finally, it was difficult to apply different statistical tests, as the data collected is nominal and ordinal. Furthermore, the ordinal questions had a "not applicable" option which was necessary to make the survey user-friendly. However, this made it difficult to code the data and treat it as interval/ratio. So when coding the data the N/A responses were treated as missing so that the correlations are meaningful.

8.7 Comparison with Previous Surveys

Although this is the first survey that focused on agile projects governance, the survey asked questions about agile development and agile projects success. Several previously conducted surveys asked similar questions so it worth comparing this survey's results with the existing ones.

In chapter 3, summaries of the existed agile surveys were presented. Also in chapter, seven Ambler's surveys were summarised as we applied further analysis on his data. Table 8-7 presents the common questions between the most recent existing surveys and the agile project governance survey as well as each survey results. The results presented in the table are the answers with the largest percentages.

From the comparison table, we can see that although the number of responses for the different surveys varied, the backgrounds of the respondents are very similar. Also it looks like agile is mostly applied in small teams of 6-10 teams with 2 weeks iterations. This does not reject the hypothesis that agile methods can be used in large projects and be successful, but most likely it is more comfortable in the small teams' zone.

Survey	Agile projects governance 2009	Agile adoption rate 2007	Agile adoption rate 2008	Agile practices 2009
#responses (completed)	129 (103)	781 (skipping allowed)	642 (492)	123 (106)
Position	26% developers	51% developers	55% developers	31% developers
Agile experience	39% less than 2 years	N/A	N/A	34% 5-10 years
Team size	44%	N/A	35%	34%

	6-10 team size		6-10 team size	6-10 team size
Iteration length	41% 2 weeks	32% 2 weeks	33% 2 weeks	N/A
Organisation size	32% 101-1000	26% 101-1000	27% 101-1000	N/A

Table 8-7 Surveys comparison

The agile adoption survey conducted in 2008 reported that agile approaches had good impact on quality with 77% responding that the quality is higher compared to 66% in 2006 survey. This agrees with the agile projects governance survey where 65% of the respondents reported that their projects had high or very high quality. Also the agile adoption survey conducted in 2007 reported that 77% of the respondents stated that 75% or more of their agile projects were successful. The agile projects governance survey reported that 45% of the projects were definitely successful. It is difficult to compare the project success results, since one survey was asking about the overall percentage of successful agile projects, whereas the other asked about the success of a specific project.

8.8 Conclusions and Summary

This chapter presented the agile project governance survey and the analysis of its results. The survey collected 129 responses of which 103 completed the survey's three sections. The respondents had more experience in IT than in agile, came from organisations that varied in terms of size, sector and experience in agile, and are relatively new to agile development.

Most of the respondents had a small team size, whose projects were either short or they were at early stages. The results are very encouraging as 72% of the respondents measured customer satisfaction often or always, also 65% reported high or above code quality, and 45% reported that their project was definitely successful.

Generally, the results were encouraging and they indicate high level of governance within the projects and teams that responded to the survey. Retrospectives were performed after each iteration by 65% of the respondents, and when the team had problems by 19%. Also, 69% of the respondents always or often recorded the retrospectives comments. This indicated that the culture of documentation does exist within the agile development environment. In addition, 74% of the respondents' teams always or often participated in the retrospective. Moreover, 88% of the respondents reported that they always or often have their say in the retrospectives. Regarding the retrospectives impact, the results were spread as 39% said that the retrospectives sometimes did affect their practices, 35% said that it often did, and 15% said it always did. Small percentages (4%, 5%) said that retrospective rarely or never changed their practices. These results suggest that using agile retrospective is helping agile team improving their process. Regarding collecting measures, the results reported that 57% of the respondents generate metrics manually, whereas 38% generate metrics automatically. Burn charts and story points and team velocity were the most popular metrics.

Interestingly, even though most of the respondents are new to agile development, and they mostly have small projects, yet the level of governance and monitoring is high which suggests that agile governance existed even for small teams.

Large organisations tend to have bigger teams and longer iterations. Although this was for one project only, it could help understand the adoption of agile methods in large organisations.

As expected, organisations with more agile experience are doing much better regarding project success and code quality. Also, more experienced organisations measure customer satisfaction more frequently. Applying agile retrospectives seems to be more mature in more experienced organisations.

The longer projects tend to have longer iterations. Also, the three factors, project success, code quality and measuring customer satisfaction are related positively; this relationship appeared multiple times throughout the thesis. The difference this time is that this survey asked about the frequency of measuring customer satisfaction and not the result of this measure. The survey results suggest that frequent measure of customer satisfaction is worthwhile. Interestingly, there is no significant relationship between team size and project success, yet it is a negative relationship.

These survey results suggest that retrospectives are more suitable for small teams. Also the results recommend performing retrospectives after each iteration for the best outcomes regarding code quality and project success. The results also suggest that using retrospectives when the team has a problem can be useful. Furthermore, it seems like the teams who perform retrospectives are also applying metrics. In other words, when a team does it right, it does for all aspects. Moreover, agile retrospectives are effective when applied properly. The retrospectives had more impact when the whole team participated, everybody had their say, and the retrospective comments were recorded properly. The findings suggested negative relations between team size and both retrospective impact and contribution, this supports our observation with the IBM team where the iteration monitor was introduced.

The results suggest that when a team is collecting metrics, it does not collect only one. For example the team who measured the “business value delivered” metric they also measured every other metric provided apart from the first two.

The reported successful projects had small team sizes (10 or less), frequently measured customer satisfaction, had high code quality, performed retrospective after each iteration, had the whole team participating in the retrospective, had everybody participating in the retrospective, and collected metrics either manually or automatically.

Finally, the survey results supported two of the research hypotheses, H6: agile software development can produce good quality software and H26: Agile adoption goes in stages and it matures over iterations, releases, and projects.

Chapter 9 Conclusions and Future Work

9.1 Introduction

The thesis presented the work conducted throughout the period of the research. This final chapter will present the research conclusions, and the future work directions.

9.2 Conclusions

In this section, the conclusions of different sections will be presented.

9.2.1 *The Literature Review Conclusions*

- The Waterfall Model Dilemma

The literature review conducted suggested that the waterfall model did not invariably work well. Evidence from both practice and literature was included to support this claim. The most important evidence from practice is the USA Department of Defense DoD standards, DoD-STD-2167, DOD-STD-2167A, and MIL-STD-498. These standards moved gradually from waterfall-based to recommending developing software in incremental builds, added more flexibility to the development process, and decreased the emphasis on documentation. In addition to the standards, two air traffic control projects were reported where the development process started as a massive waterfall then the projects were cancelled and restarted with an iterative or/and incremental approaches. The evidence from literature presented a number of publications as early as the 70s, which criticised the waterfall model and discussed its limitations.

- The Origins of Agile Methods

Although agile methods are new as a whole, their principles and ideas existed long time ago, and people who criticized the traditional methods suggested alternative approaches which were nothing but agile ideas. Unfortunately, these alternative approaches had not been treated seriously enough. The literature review concluded with the argument that although agile methods are new as a whole, they have strong roots in the history of software engineering. In addition to the iterative and incremental approaches that have been in use since 1957 people who criticised the traditional methods suggested alternative approaches which were actually agile ideas such as the response to change, customer involvement, and valuing working software over documentation. We therefore presented and discussed the reasons behind the development and introduction of agile methods, as a reaction to traditional approaches, as a result of people's experience, and in particular focusing on reusing ideas from the history.

- An Agile Method Definition

We proposed a general definition of an agile method as Adaptive: welcomes change, in technology and requirements, even to the point of changing the method itself. In addition, it responds to feedback about previous work. Iterative and incremental: the software is developed in several iterations, each from planning to delivery. In each iteration part of the system is developed, tested, and improved while a new part is being developed. In each iteration, the functionality will be improved. In addition, the system is growing incrementally as new functionality is added with each release. After each iteration(s), a release will be delivered to the customer in order to get feedback. People-oriented: in an agile method, people are the primary drivers of project success. Therefore, the role of the process in an agile method is to support the development team determine the best way to handle work, furthermore, an agile method emphasises on face-to-face communication within the team and with the customer who is closely involved with the development process rather than written documents.

9.2.2 *The Empirical Study Conclusions*

- Project A Summary

The first case study gave us a good understanding of the adoption of agile software development and how it is related to the different aspect of quality within a traditional organisation such as IBM. The project did not follow any specific agile method at the beginning but as the time passes, the used process was influenced more and more with

Scrum. The project started with 16 members and the team size was stable throughout the period of the interviews. The team followed 2 weeks iterations, and used iteration planning, TDD, refactoring and continuous integration. The team was happy and motivated which played a big role in the success of the project. Off-site members did not work very well so they moved to join the rest of the team on-site. The company culture affected the development in a number of ways including delays in early deliveries because of the legal issues, the team was unable of keeping whiteboards overnight, and the quality plan was not flexible enough to fit the agile way of working. However, we argue that the company culture had a more positive impact on the project than negative. This is because the project followed the existing good practices in IBM such as the emphasis on measurements. In addition, although quality plans were inflexible, they worked well and they are on the way of producing an agile quality plan. The project delivered on time, defects count was as predicted, and was similar to previous projects, and customer satisfaction was improved.

- Project B Summary

Studying Project B and keeping up with the project changes over three years gave us not only a good understanding of the adoption of agile software development and how it is related to the different aspect of quality, but it helped us understand how agile adoption is evolving over time. The team followed 4 weeks iterations, and used iteration planning, pair programming at some stages and also refactoring. There were no problems reported regarding off-site members as there were in project A. The company culture affected the development in a number of ways including delays because of the legal issues, and sometimes some features were needed as part of the company policy, which increased the load on the team. However, as in project A, we argue that the company culture had a more positive impact on the project than negative. This is because the project followed the existing good practices in IBM such as measuring customer satisfaction, and collecting defects and quality measures. With six releases in the market all on time, the project is a success as it is delivering high quality code with low defect rate and the customers are satisfied. In the early stages, it was not clear if the low defect rate was because of the new process or because the project is not mature or heavily used yet. After release 4, however, and with the increase in customers number it is more certain that the used approach has a measurable and positive impact on quality.

The project did not follow any particular agile method and this did not change over the project, however new practices were introduced such as using user stories. More importantly, the iteration plan changed after release 4 to reduce the pressure on developers. The other change is the project team size, which grew to 77 members. One of the important changes we observed and was confirmed by the project manager is the project agility. This was demonstrated with the requirements, although they were flexible at the beginning of the project, this however changed as the project grew and matured and this meant that customer demands has to wait until the following iteration or even release.

- Generated Hypothesis

Analyzing the interviews resulted in a list of 31 grounded hypotheses; this initial list was reviewed and reduced into the following list of 13 hypotheses:

H1: Agile software development can achieve customer satisfaction

H2: Customer involvement, demands, and requests increase throughout the project

H5: Agile software development assures quick and effective response to customer's requests

H21: Team members are happy and motivated when using agile software development

H6: Agile software development can produce good quality software

H8: Automated tests can assure high quality code

H9: In agile software development, testing is more effective when testers and developers are working in parallel

H10: The code developed using agile software development has the same (if not lower) defect rate than traditional methods

H11: The quality of the code increases as the number of iterations increases

H15: Refactoring can help improving product quality in agile software development

H22: Testing should be the responsibility of all team members in agile software development

H25: Communication level between different stakeholders is higher in agile software development than in traditional approaches

H26: Agile adoption goes in stages and it matures over iterations, releases, and projects

9.2.3 *The Iteration Monitor Conclusions*

The iteration monitor was introduced as a diagnosis tool in order to identify issues and trends and hence improve the process in the following iterations. The iteration monitor is recommended when retrospectives or reflection meetings are not very effective, especially when the team is getting larger so it becomes difficult to capture everybody's opinion. Also, the iteration monitor can be useful when the project is running for a long time so keeping a record for each iteration will be very useful in identifying trends and changes over time. The iteration monitor was used to address the needs of IBM's team; and data was collected over three iterations. Other teams may adapt the monitor as needed by changing it according to their settings. Analyzing the data collected by the iteration monitor over the three iterations produced a large set of results; also, comparing the three iterations presented the changes that occurred over the three months.

9.2.4 *Applying Correlations and Factor Analysis on Existing Surveys Conclusions*

The results of analyzing the data from the agile adoption survey conducted in 2006 (Ambler 2006) suggested that agile approaches had a good impact on quality when they had positive impact on stakeholders' satisfaction. Also the findings suggested that as productivity improves, quality and satisfaction improve as well. The interesting result is that when productivity, quality, and satisfaction went higher, the cost went lower as the correlation coefficient was negative. Also it seems likely that the impact of agile approaches on cost was higher in larger organisations.

The results of analysing the data from the agile adoption survey conducted in 2008 (Ambler 2008b) suggested that when agile approaches had a good impact on one aspect, this was true for others, and more importantly the results suggest that when the impact of agile approaches was positive on quality, satisfaction and production, the cost of software development went down. Also, it seems like large size organisations are having problems applying agile approaches as there is a negative correlation between organisation size and both success rate and productivity.

Applying the factor analysis on agile practices effectiveness data from a survey conducted in 2007 (Ambler 2007) resulted in reducing 58 practices to 15 factors presented below. Each factor is associated with a list of agile practices that can be used as a checklist when improving the related factor.

Factor1: architecture modelling
Factor2: traditional analysis
Factor3: process/governance
Factor4: database practices
Factor5: communication (team) – whiteboard practices
Factor6: agile quality assurance
Factor7: communication (team)
Factor8: code analysis and inspection
Factor9: lightweight testing and review
Factor10: architecture and configuration

Factor11: traditional quality assurance
Factor12: coding standards
Factor 13: lightweight requirements
Factor14: incremental and iterative development
Factor15: communication (customers)

The relationships between the extracted factors were studied using correlations. The results suggested that people who applied iterative and incremental development and agile quality assurance practices had a high success rate. Also, people who applied governance practices also applied agile quality assurance practices but there was not much emphasis on high communication with the customers. We have to be careful here as only two practices; co-location and active stakeholder participation contributed to the communication with the customer factor. Communication with the team factor had a positive relation with governance and agile quality assurance practices. A negative but not significant relation was found between traditional quality assurance and agile quality assurance. This maybe because agile projects have tended to abandon more traditional quality assurance practices as they move more towards agile quality assurance. Interestingly, success rate related negatively with traditional analysis methods such as Gantt chart and detailed requirements specification.

9.2.5 *Agile Projects Governance Survey Conclusions*

The survey collected 129 responses of which 103 completed the survey's three sections. The respondents had more experience in IT than in agile, came from organisations that varied in terms of size, sector and experience in agile, and are relatively new to agile development.

Most of the respondents had a small team size, whose projects were either short or at early stages. The results are very encouraging as 72% of the respondents measured customer satisfaction often or always, also 65% reported high or above code quality, and 45% reported that their project was definitely successful. These results support the hypothesis that agile methods have good impact on code quality.

Generally, the results were encouraging and indicate high level of governance within the projects and teams that responded to the survey. Retrospectives were performed after each iteration by 65% of the respondents, and when the team had problems by 19%. Also, 69% of the respondents always or often recorded the retrospectives comments. This indicated that the culture of documentation does exist within the agile development environment. In addition, 74% of the respondents' teams always or often participated in the retrospective. Moreover, 88% of the respondents reported that they always or often have their say in the retrospectives. Regarding the retrospectives impact, the results were spread as 39% said that the retrospectives sometimes did affect their practices, 35% said that it often did, and 15% said it always did. Small percentages (4%, 5%) said that retrospective rarely or never changed their practices. These results suggest that using agile retrospective is helping agile team to improve their process. Regarding collecting measurements, the results reported that 57% of the respondents generate metrics manually, whereas 38% generate metrics automatically. Burn charts and story points and team velocity were the most popular metrics.

Large organisations tend to have bigger teams and longer iterations. Although this was for one project only, it could help understand the adoption of agile methods in large organisations.

As expected, organisations with more agile experience are doing much better regarding project success and code quality. Also, more experienced organisations measure customer satisfaction more frequently. Applying agile retrospectives seems to be more mature in more experienced organisations. This can be seen as a sign of maturity, which supports the hypothesis: the adoption of agile methods goes in stages and it improves over iterations, releases, and projects.

The longer projects tend to have longer iterations. Also, the three factors, project success, code quality and measuring customer satisfaction are related positively; this relationship appeared multiple times throughout the thesis. The difference this time is that this survey asked about the frequency of measuring customer satisfaction and not the result of this measure. The survey results suggest that frequent measure of customer satisfaction is worthwhile. Interestingly, there is no significant relationship between team size and project success, yet it is a negative relationship.

These survey results suggest that retrospectives are more suitable for small teams. Also the results recommend performing retrospectives after each iteration for the best outcomes regarding code quality and project success. The results also suggest that using retrospectives when the team has a problem can be useful. Furthermore, it seems like the teams who perform retrospectives are also applying metrics. In other words, when a team does it right, it does for all aspects. Moreover, agile retrospectives are effective when applied properly. The retrospectives had more impact when the whole team participated, everybody had their say, and the retrospective comments were recorded properly. The findings suggested negative relations between team size and both retrospective impact and contribution, this supports our observation with the IBM team where the iteration monitor was introduced.

The results suggest that when a team is collecting metrics, it does not collect only one. For example the team who measured the “business value delivered” metric they also measured every other metric as well.

Finally, the reported successful projects had small team sizes (10 or less), frequently measured customer satisfaction, had high code quality, performed retrospective after each iteration, had the whole team participating in the retrospective, had everybody participating in the retrospective, and collected metrics either manually or automatically.

9.2.6 *Significant Relationships between Different Variables*

Different types of analysis were conducted on different sets of data throughout the research. Some of the findings were statistically significant, and some were not. In this section, statistically significant relationships between different variables will be summarised. We will focus on the variables that have impact on quality, satisfaction and project success as this is the focus of this thesis. The factors that are related to quality, stakeholder satisfaction, and project success are presented in tables 9-1, 9-2, and 9-3 respectively along with their correlation coefficient values. The numbers 0.01 and 0.05 presents the probability at which the correlation was significant.

Variables		Adoption Survey 2006		Adoption Survey 2008		Agile Project Governance Survey	
		0.01	0.05	0.01	0.05	0.01	0.05
Quality	Productivity	.68		.55			
	Satisfaction	.66		.51			
	Cost	-.06		-.26			
	Success			.36		.31	
	Organisation experience						.20
	Retrospective Impact						.24
	Frequency of customer satisfaction measure					.47	

Table 9-1 Variables which have a significant correlation with quality, and their correlation coefficients

Variables		Adoption Survey 2006		Adoption Survey 2008	
		0.01	0.05	0.01	0.05
Satisfaction	Productivity	.60		.43	
	Quality	.66		.51	
	Cost	-.06		-.28	
	Success			.27	

Table 9-2 Variables which have a significant correlation with satisfaction, and their correlation coefficients

Variables		Adoption Survey 2008		Agile Project Governance Survey	
		0.01	0.05	0.01	0.05
Project Success	Productivity	.41			
	Cost	-.28			
	Quality	.36			
	Frequency of customer satisfaction measure			.40	
	Organisation size	-.18		-.26	
	Organisation Experience			.40	
	Retrospective impact			.33	
	Retrospective contribution			.27	
	Retrospective participation				.26

Table 9-3 Variables which have a significant correlation with project success, and their correlation coefficients

Although correlation gives no indication about the direction of causality, we can still conclude for example that the more experience the organisation has the more likely the project will succeed. Although there is no statistical reason why project success cannot cause an increase in organisation experience, however it does not make human sense.

9.3 Research Contributions

I. An empirical study of two agile projects was carried out, focussing on quality in agile projects. It was notable that both projects were on-time, through multiple releases, achieving high level of customer satisfaction and low defect rates. A list of grounded hypotheses was generated and refined. The 13 remaining hypotheses were organized in three main groups: the impact of agile software methods on software quality, stakeholder's satisfaction, and process quality. Six of these hypotheses in particular were supported throughout the research:

H5: Agile software development assures quick and effective response to stakeholders' requests (Chp.6 – P.113)

H6: Agile software development can produce good quality software (Chp.6 – P.113) – (Chp.7 – P.115-118) – (Chp.8 – P.140)

H1: Agile software development can achieve customer satisfaction (Chp.7 – P.115-118)

H21: Team members are happy and motivated when using agile software development (Chp.6 – P.113)

H11: The quality of the code increases as the number of iterations increases (Chp.6 – P.113)

H26: The adoption of agile methods goes in stages and it improves over time, releases, and projects (Chp.8 – P.147)

II. The iteration monitor was designed to identify issues and trends within a team in order to improve the process and understand changes between iterations. One of the IBM teams used the iteration monitor over three iterations.

III. Three existing agile adoption surveys were re-coded and re-analysed. New and statistically significant results were obtained which suggest that:

a. When agile methods had good impact on one aspect, they also had good impact on others. Good impact on quality, customer satisfaction, and productivity were positively correlated, so that as productivity improves, quality and satisfaction improve, and cost is reduced.

b. 58 techniques used by agile teams were clustered into 15 factors which can be used as a guide for agile projects process improvement.

c. Agile quality assurance practices and iterative and incremental development have a positive, statistically significant, relationship with project success.

IV. A new survey to study agile projects governance was conducted. The results presented the state of art of agile project governance including the use of retrospectives and metrics in an agile software development environment. The statistical analysis of this survey suggested that:

a. Organisation size has a negative, statistically significant relationship with project success (also supported in contribution III)

b. Retrospectives are more effective when applied properly as they had more impact on the project when the whole team participated, everybody had their say, and comments were recorded.

   c. Project success has a positive and statistically significant relationship with a number of agile metrics such as team velocity, business value delivered, running tested features, as well as more traditional metrics such as number of test cases, and defect count after testing.
   d. Good practices are related together: (good quality, high productivity, high customer satisfaction, and low cost) and (performing retrospective, team participation, comments recording, collecting metrics). In other words when a team or an organisation does one aspect well, they do all aspects well.

V. Our review of the literature on traditional and agile methods generated new insights and understanding into the nature of agile methods and their roots.
   a. The reasons behind the development and introduction of agile methods are identified, as a reaction to traditional methods, as a result of people's experience, and in particular focusing on reusing ideas and techniques from the history of software development.
   b. A new definition of agile methods is given whereby they are defined as adaptive, iterative and incremental, with a people oriented process.

As with all empirical research, there are a number of threats to the validity of the previous conclusions. They are based on interviews and questionnaires, so the data collected is subjective and based on the subjects' perception of quality rather than direct measures. As adaptivity and people-orientation are key components of agile methods, it is not possible to come up with definitive recommendations: instead, each project and team needs to select and refine those techniques which work well for them.

9.4 Mapping the Different Research Aspects to TQM

In order to capture the different aspects covered by the research, we will use TQM key elements presented by (Kan 2002)

- Customer focus: the impact of agile software development on customer satisfaction was investigated with focus on the importance on the frequency of measuring this satisfaction
- Process improvements: the 15 factors extracted using the factor analysis can be used as a process improvement tool to enhance one particular area by focusing on its different associated practices; in addition, the iteration monitor can contribute to process improvement.
- Human side of quality: the impact of agile software development on the team satisfaction and morale was studied; also the different aspects of retrospective were highlighted.

- Measurement and analysis: the iteration monitor is a measurement tool, and the discussion in chapter 8 covered the collection and analysis of other metrics.

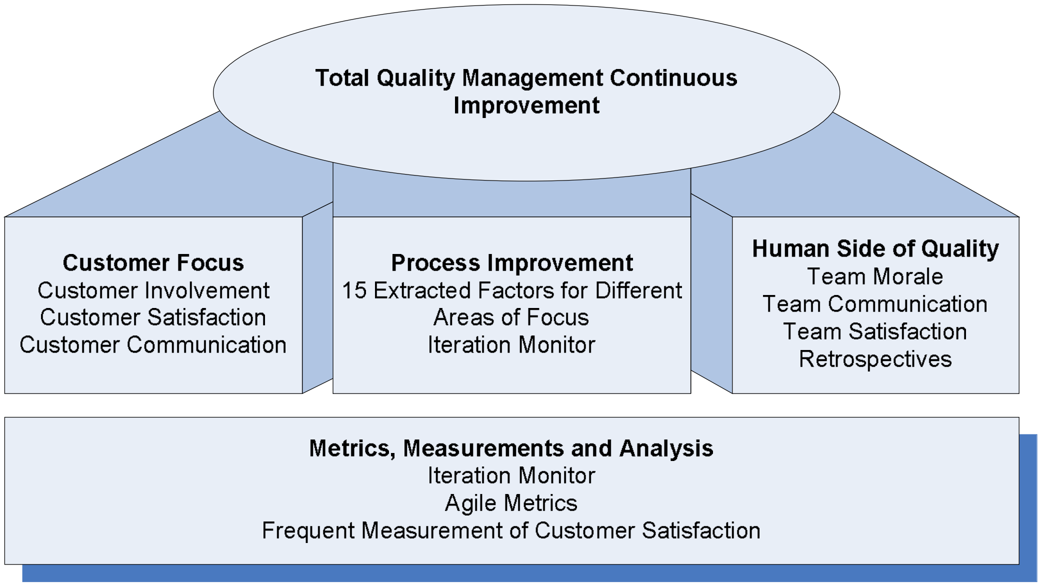


Figure 9-1 Mapping the research aspects to TQM

9.5 Future Work

9.5.1 *Short Term Plans*

- After using the iteration monitor over three iterations, the project manager in IBM was interested in continuing the use of the monitor with the team. Collecting more data using the iteration monitor will give us the opportunity to monitor the project for longer time and that will help identifying project changes and it will help improving the iteration monitor.
- Automating the iteration monitor will be an interesting future work. This can make this tool available for other teams. This can include the following features:
    - Add/remove questions
    - Add/remove sections
    - Generating reports that present the descriptive results
    - A function to compare the current iteration with previous one/ones.
- We would like to analyse the quality metrics collected for project B in IBM and compare the results with an old project also from IBM where non-agile software development approaches were used. Analyzing this data will be useful

to compare the differences between the two approaches. This study will also help support some of the hypotheses generated during this research.

9.5.2 *Long Term Plans*

- The agile adoption survey was conducted in 2009 and most likely will run again in 2010. It will be interesting to analyse the data collected and compare it with the results we obtained from our research.
- The agile projects governance survey can be repeated in 2011 to study the change in agile governance practices.
- During the literature review, we observed that the amount of empirical research conducted on agile methods and agile software development has increased over the last three years. It will be interesting conduct a new systematic review of the empirical studies that were published after 2005, as the last available systematic review included studies of agile software development up to and including 2005.
- We have used different empirical approaches in this research, qualitative and quantitative. In addition, different statistical tests were studied and used. It will be useful to write an article about these different empirical methods so other researchers can benefit from our experience.
- The interest in governance for agile software development emerged in a later stage of this research. We found this area very interesting and engaging, therefore we would like to consider this area as a future research direction. Especially studying how agile software development can benefit from the available IT governance models and framework.

9.6 Final Words

Our understanding of agile methods developed over the period of conducting this research. It moved from thinking of agile methods as “the best” way of doing software development, to a good approach to software development that can achieve good quality software, satisfied stakeholders and project success when it is applied properly by the people who are willing to work in an “agile” way. We found that although empirical software development research is challenging, yet it is very interesting, enjoyable and most importantly, when applied rightly, can give valuable results. Finally, we found that applying statistical analyses can be highly useful in empirical software engineering research; however, finding the suitable statistical analysis for the collected data requires good understanding of the data itself, the goals of the study as well as the statistical tests available.

References

Abrahamsson, P., O. Solo, J. Ronkainen and J. Warsta (2002). Agile Software Debvelopment Methods, VTT technical Research Centre of Finland.

agile. (2009). "Cambridge Advanced learner's Dictionary Online." Retrieved 09/12/2009.

Alegria, J. A. H. and M. C. Bastarrica (2007). Implementing CMMi using a Combination of Agile Methods

Ambler, S. (2005). "Quality in an Agile World." Software Quality Professional 7(4): 34-40.

Ambler, S. (2006). "Results from Scott Ambler's 2006 Agile Adoption Rate Survey " Retrieved 28/07/2008, from www.ambysoft.com.

Ambler, S. (2007). "Results from Scott Ambler's 2007 Agile Adoption Rate Survey " Retrieved 28/07/2008, from www.ambysoft.com.

Ambler, S. (2008a). "Acceleration: An Agile Productivity Measure " Retrieved 08/12/2009.

Ambler, S. (2008b). "Results from Scott Ambler's February 2008 Agile Adoption Survey." Retrieved 08/12/2009, from www.agilemodeling.com/surveys/

Ambrosio, J. (1988). Software in 90 days Software Magasine, Wiesner Publications, Inc.

Anderson, D. J. (2005). Stretching Agile to fit CMMI level 3 Microsoft Corporation

Barnett, L. (2006a). "Agile Survey Results: Solid Experience And Real Results." Agile Journal Retrieved 28/07/2008, from http://www.agilejournal.com/content/view/93/.

Barnett, L. (2006b). "And The Agile Survey Says…." Agile Journal Retrieved 28/07/2008, from http://www.agilejournal.com/content/view/29/44/.

Basili, V. R. (1996). The Role of Experimentation in Software Engineering: Past, Current, and Future. Proceedings of the 18th International Conference on Software Engineering. Berlin, Germany, IEEE Computer Society.

Basili, V. R., G. Caldiera and H. D. Rombach (1994). The Goal Question Metric Approach. Encyclopedia of Software Engineering J. J. Marciniak, Wiley: 528-532.

Basili, V. R., R. W. Selby and D. H. Hutchens (1986). "Experimentation in software engineering." 12(7): 733-743.

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

Basili, V. R. and M. V. Zelkowitz (2007). "Empirical studies to build a science of computer science." 50(11): 33-37.

Beck, K. and C. Andres (2004). Extreme Programming Explained: Embrace Change (2nd Edition), Addison-Wesley Professional.

Benington, H. D. (1956). "Production of Large Computer Programs." Symposium on Advanced Programming Methods for Digital Computers: 15-27.

Benington, H. D. (1983). "Production of Large Computer Programs." Annual of the History of Computing 5(4): 299-310.

Benington, H. D. (1987). Production of large computer programs. Proceedings of the 9th international conference on Software Engineering. Monterey, California, United States, IEEE Computer Society Press.

Boaden, R. J. (1997). "What is total quality management...and does it matter?" Total Quality Management; Vol. 8, No. 4: 153-171.

Boehm, B. (1976). "Software Engineering." IEEE Transactions on Computers C-25(12).

Boehm, B. (1979). Guidelines for Verifying and Validating Software Requirements and Design Specifications, Euro IFIP 79, P. A. Samet (editor), North-Holland Publishing Company, IFIP.

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

Boehm, B. (1988). "A Spiral Model of Software Development and Enhancement." IEEE Computer 21(5): 61-72.

Boehm, B. (2002). "Get Ready for Agile Methods with Care." Computer 35(1): 64-69.

Boehm, B. (2007). Personal Conversation. N. Abbas. Madrid.

Boehm, B., A. Egyed, J. Kwan, D. Port, A. Shah and R. Madachy (1998). "Using the WinWin Spiral Model: A Case Study " IEEE Computer 31(7): 33-44.

Boehm, B. and R. Turner (2003). Balancing Agility and Discipline: A Guide for the Perplexed, Addison-Wesley Longman Publishing Co., Inc.

Boehm, B. W., J. R. Brown and M. Lipow (1976). Quantitative evaluation of software quality. Proceedings of the 2nd international conference on Software engineering. San Francisco, California, United States, IEEE Computer Society Press.

Booch, G. (1995). Object Solutions: Managing the Object-Oriented Project, Addison Wesley Longman Publishing Co., Inc.

Booch, G., J. Rumbaugh and I. Jacobson (1996). Unified Modeling Language for Object-Oriented Development, Rational Software Corporation.

Brooks, F. P. (1979). The Mythical Man-Month, Addison-Wesley.

Brooks, F. P. (1995). The Mythical Man-Month, Addison-Wesley.

Charette, R. N. (2002). Foundation of Lean Development: The Lean Development Manager's Guide. The Foundations Series on Risk Management, (CD). Spotsylvania,Va.:ITABHI Corporation. 2.

Chow, T. and D.-B. Cao (2008). A Survey Study of Critical Success Factors in Agile Software Projects, Elsevier Science Inc. 81: 961-971.

Coad, P., J. deLuca and E. Lefebvre (1999). Java Modeling Color with Uml: Enterprise Components and Process with Cdrom, Prentice Hall PTR.

Cockburn, A. (1997). Using VW Staging to Clarify Spiral Development, OOPSLA'97 Practitioner's Report, Humans and Technology technical report. .

Cockburn, A. (1999). Characterizing People as Non-linear First-Order Components in Software Development, Humans and Technology Technical Report.

Cockburn, A. (2002a). Agile Software Development, Addison-Wesley Longman Publishing Co., Inc.

Cockburn, A. (2002b). "Agile Software Development Joins the “Would-Be” Crowd." Cutter IT Journal 15(1): 6-12.

Cockburn, A. (2005). Crystal Clear A Human -Powered Methodology for Small Teams, Addison-Wesley.

Cockburn, A. (2007). Personal Communication. N. Abbas. online.

Cockburn, A. and J. Highsmith (2001a). "Agile Software Development: The Business of Innovation." Computer 34(9): 120-127.

Cockburn, A. and J. Highsmith (2001b). "Agile Software Development: The People Factor " Computer 34(11): 131-133.

Cockburn, A., P. McBreen and S. Hutchinson. (2005). "Iterative Vs Incremental."   Retrieved 28/07/2008, from http://c2.com/cgi/wiki?IterativeVsIncremental.

Cohen, D., M. Lindvall and P. Costa (2004). "An Introduction to Agile Methods." Advances in Computers: 1-66.

Corporate Report (2003). Agile Methodologies Survey Results, Shine Technologies Pty Ltd., Victoria, Australia.

Crosby, P. B. (1980). Quality is Free, Mentor.

Deming, W. E. (1982). Out of the Crisis MIT Press

Deursen, A. v. (2001). "Customer Involvement in Extreme Programming: XP2001 Workshop Report." 26(6): 70-73.

Deutsch, M. S. and R. R. Willis (1988). Software Quality Engineering, A Total Technical Management Approach, Prentice Hall.

Dingsøyr, T., T. Dybå and P. Abrahamsson (2008). A Preliminary Roadmap for Empirical Research on Agile Software Development. Proceedings of the Agile 2008 - Volume 00, IEEE Computer Society.

Dubinsky, Y. and P. Kruchten (2009). 2nd Workshop on Software Development Governance (SDG). Proceedings of the 2009 31st International Conference on Software Engineering: Companion Volume, IEEE Computer Society.

Dyba, T. and T. Dingsoyr (2008). "Empirical Studies of Agile Software Development: A Systematic Review." Information and Software Technology(50): 833-859.

Evans, M. W. and J. J. Marciniak (1987). Software Quality Assurance and Management, John Wiley and Sons.

Ferreira, C. and J. Cohen (2008). Agile Systems Development and Stakeholder Satisfaction: A South African Empirical Study. 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. Wilderness, South Africa, ACM.

Field, A. (2005). Discovering Statistics Using SPSS, Sage.

Fielding, N. (2001). Ethnography in N.Gilbert (ed.) Researching Social Life, Sage: 145-163.

Firebaugh, G. (1997). Analyzing Repeated Surveys Sage.

Firesmith, D. (1987). The Mangement Implications of the Recursive Nature of Object-Oriented Development, AdaEXPO/ SigAda Conference proceeding, Boston,USA.

Fowler, M. (2005). "The New Methodology." Retrieved 28/07/2008, from http://www.martinfowler.com/articles/newMethodology.html.

Fowler, M., K. Beck, J. Brant, W. Opdyke and D. Roberts (1999). Refactoring: Improving the Design of Existing Code, Addison-Wesley Professional

Galin, D. (2003). Software Quality Assurance: From Theory to Implementation Addison Wesley

Garratt, P. (2007). Personal Communication. N. Abbas. Southampton.

Gilb, T. (1976). Software Metrics, Winthrop Publishers

Gilb, T. (1981). "Evolutionary Development." 6(2): 17-17.

Gilb, T. (1985). "Evolutionary Delivery versus the "waterfall model" " ACM SIGSOFT Software Engineering Notes 10(3): 49-61.

Gillies, A. C. (1992). Software Quality: Theory and Management, Chapman \&amp; Hall, Ltd.

Gladden, G. R. (1982). "Stop the Life-cycle, I Want to Get off." 7(2): 35-39.

Glaser, B. G. and A. Strauss (1967). The Discovery of Grounded Theory: Strategies for Qualitative Research, Aldine Transaction

governance. (2009). "Compact Oxford English Dictionary Online." Retrieved 09/12/2009.

Government Accounting Office (1998). Air Traffic Control:Evaluation and Status of FAA's Automation program, USA GAO.

Guaspari, J. (1991). I Know It When I See It: A Modern Fable About Quality American Management Association.

Hanssen, G. and T. Erlend (2006). Agile Customer Engagement: A Longitudinal Qualitative Case Study. Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering. Rio de Janeiro, Brazil, ACM.

Hartmann, D. and R. Dymond (2006). Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value. Proceedings of the conference on AGILE 2006, IEEE Computer Society.

Hashmi, S. I. and J. Baik (2007). Software Quality Assurance in XP and Spiral - A Comparative Study. Proceedings of the The 2007 International Conference Computational Science and its Applications, IEEE Computer Society.

Hashmi, S. I. and J. Baik (2008). Quantitative Process Improvement in XP Using Six Sigma Tools. Proceedings of the Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008), IEEE Computer Society.

Highsmith, J. (2000). Adaptive Software Development: a Collaborative Approach to Managing Complex Systems, Dorset House Publishing Co., Inc.

Highsmith, J. (2002). Agile Software Development Ecosystems, Addison-Wesley Longman Publishing Co., Inc.

Highsmith, J., k. Beck, A. Cockburn and R. Jeffries. (2001). "Agile Manifesto." Retrieved 28/07/2008, from www.agilemanifesto.org.

Hove, S. E. and B. Anda (2005). Experiences from Conducting Semi-structured Interviews in Empirical Software Engineering Research. Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS'05) - Volume 00, IEEE Computer Society.

Huo, M., H. Verner, l. Zhu and M. A. Babar (2004). Software Quality and Agile Methods, Proceeding of the 28th Annual International Software and Application Conference (COMPSA'04).

IEEE (1990). IEEE Standard 610.12-1990 Glossary of Software Engineerng Terminology. New York, The Institute of Electrical and Electronics Engineers.

IEEE (1998). IEEE Standard 1028-1997 For Softwae Reviews. New York, The Institute of Electrical and Electronics Engineers.

Increment. (2009). "Compact Oxford English Dictionary Online." Retrieved 09/12/2009.

Iterate. (2009). "Compact Oxford English Dictionary Online." Retrieved 09/12/2009.

Jacobson, I. (2006). "The Essential Unified Process : An introduction " Retrieved 28/07/2008, from http://www.ivarjacobson.com/products/essup.cfm.

Jeffries, R. (2004). "A Metric Leading to Agility " Retrieved 24/07/2008, from http://www.xprogramming.com/xpmag/jatRtsMetric.htm.

Johansson, C. and C. Bucanac (1999). The V-Model, IDE, University of Larlskrona / Ronneby.

Jones, C. (1991). Applied Software Measurement: Assuring Productivity and Quality, McGraw-Hill, Inc.

Juran, J. M. and F. M. Gryna (1988). Juran's Quality Control Handbook Mcgraw-Hill

Kaiser, H. F. (1974). "An Index of Factorial Simplicity." Psychometrika 39: 31-36.

Kan, S. H. (2002). Metrics and Models in Software Quality Engineering, Addison-Wesley Longman Publishing Co., Inc.

Kanji, G. K. (2006). 100 Statistical Tests, SAGE.

Karlstr, D. and P. Runeson (2006). "Integrating Agile Software Development into Stage-gate Managed Product Development." Empirical Software Engineering 11(2): 203-225.

Kass, R. A. and H. E. A. Tinsley (1979). "Factor Analysis." Journal of Leisure Research 11(120-138).

Kitchenham, B. and S. L. Pfleeger (1996). Software Quality: The Elusive Target, IEEE Computer Society Press. 13: 12-21.

Kitchenham, B. A., S. L. Pfleeger, D. C. Hoaglin, K. E. Emam and J. Rosenberg (2002). "Preliminary Guidelines for Empirical Research in Software Engineering." IEEE Transactions on Software Engineering 28(8): 721-734.

Kruchten, P. (2001). What is The Rational Unified process Rational Software Corporation 2001

Kruskal, W. H. and W. A. Wallis (1952). "Use of Ranks in One-criterion Variance Analysis." Journal of the American Statistical Association 47: 583-621.

Larman, C. (2004). Agile and Iterative Development:A Manager's Guide. C. Alistair and H. Jim, Pearson Education, Inc.

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

Layman, L., L. Williams and L. Cunningham (2004). Exploring Extreme Programming in Context: An Industrial Case Study. Proceedings of the Agile Development Conference, IEEE Computer Society.

Lehto, I. and K. Rautiainen (2009). Software Development Governance Challenges of a Middle-sized Company in Agile Transition. Proceedings of the 2009 ICSE Workshop on Software Development Governance, IEEE Computer Society.

Let, S. H. (1998). "An Earned Value Tracking System for Self-Directed Software Teams." Retrieved 28/07/2008, from http://www.scribd.com/doc/2060573/1998-BenchmarkQA-Whitepaper-An-Earned-Value-Tracking-System-for-SelfDirected-Software-Teams.

Levine, L. (2005). Reflections on Software Agility and Agile Methods, Software Engineering Institute Carnegie Mellon University Pittsburgh, PA U.S.A.

Likert, R. (1932). "A Technique for the Measurement of Attitudes." Archives of Psychology: 1-55.

Livermore, J. A. (2007). Factors that Impact Implementing an Agile Software Development Methodology. SoutheastCon, 2007. Proceedings. IEEE.

Liversidge, E. (2005). The Death of the V-Model, Harmonic Software Systems Ltd.

Madden, W. A. and K. Y. Rone (1984). "Design, Development, Integration: Space Shuttle Primary Flight Software System " Communications of the ACM 27(9): 914-925.

Marick, B. (1999). New Models for Test Development, Reliable Software Technologies.

Martin, J. (1991). Rapid Application Development, Macmillan Publishing Co., Inc.

McBreen, P. (2003). Quality Assurance and Testing in Agile Projects McBreen.Consulting.

McCall, J. A., P. K. Richards and G. F. Walters (1977). "Factors in Software Quality." Nat'l Tech. Information Service 1,2,3.

McConnell, S. (1996). Rapid Development: Taming Wild Software Schedules, Microsoft Press.

McCracken, D. D. and M. A. Jackson (1982). Lifecycle Concept Considered Harmful, ACM Press. 7: 29-32.

McGregor, D. (2006). The Human Side of Enterprise: Annotated Edition, McGraw-Hill Professional.

Milicic, D. (2006). Software Quality Models and Philosophies, http://www.bth.se/eng.

Mills, H. D. (1980). "The Management of Software Engineering. Part I: Principles of Software Engineering." IBM Systems journal 19(4): 414-480.

Mnkandla, E. and B. Dwolatzky (2006). Defining Agile Software Quality Assurance. Proceedings of the International Conference on Software Engineering Advances (ICSEA'06).

Nardi, P. M. (2002). Doing Survey Research: A Guide to Quantitative Research Methods Allyn & Bacon.

Nawrocki, J. R., M. Jasinski, B. Walter and A. Wojciechowski (2002a). Combining Extreme Programming with ISO 9000 EURAsia-ICT 2002, LNCS 2510 786-794.

Nawrocki, J. R., M. Jasiñski, B. Walter and A. Wojciechowski (2002b). Extreme programming Modified: Embrace Requirements Engineering Practices proceedings of the IEEE joint International Conference on Requirement Engineering (RE' 02)

Newberry, M. G. A. (1995). Changes from DOD-STD-2167A to MIL-STD-498 CrossTalk, STSC at Hill AFB, UT.

O'Neill, D. (1983). "Integration Engineering Perspective." Journal of Systems and Software 3(1): 77-83.

Opdyke, B. (1992). Refactoring Object-Oriented Frameworks, University of Illiniois at Urbana-Champaign.

Palmer, S. R. and M. Felsing (2001). A Practical Guide to Feature-Driven Development, Pearson Education.

Parsons, D., H. Ryu and R. Lal (2007). The Impact of Methods and Techniques on Outcomes from Agile Software Development Projects. IFIP International Federation for Information Processing, Springer Boston: 235-249.

Paulk, M. C. (2002). "Agile Methodologies and Process Discipline." CrossTalk- The Journal of defence Software Engineering: 15-18.

Perry, D. E., A. A. Porter and L. G. Votta (2000). Empirical Studies of Software Engineering: A Roadmap. Proceedings of the Conference on The Future of Software Engineering. Limerick, Ireland, ACM Press.

Perry, W. (1987). Effective methods for EDI quality assurance, Prentice-Hall.

Pikkarainen, M., J. Haikara, O. Salo, P. Abrahamsson and J. Still (2008). The Impact of Agile Practices on Communication in Software Development, Kluwer Academic Publishers. 13: 303-337.

Poppendieck, M. and T. Poppendieck (2003). Lean Software Development: An Agile Toolkit, Addison-Wesley Longman Publishing Co., Inc.

Pressman, R. S. (1987). Software Engineering : A Practitioner's Approach, 2nd Edition, McGraw-Hill.

Pressman, R. S. (2001). Software Engineering: A Practitioner's Approach, 5th Edition , McGraw-Hill

Quality. (2009). "Cambridge Advanced Learner's Dictionary Online." Retrieved 09/12/2009.

Radatz, J., M. Olson and S. Campbell (1995). MIL-STD-498, CrossTalk, STSC at Hill AFB, UT.

Rational (1998). Rational Unified Process: Best Practices for Software Development Teams, Rational Software White Paper TP026B, Rev 11/01.

Report (2003). Board Breifing on IT Governance, IT Governance Institute.

Robinson, H. and H. Sharp (2004). The Characteristics of XP Teams. Extreme Programming and Agile Processes in Software Engineering, Springer.

Robinson, H. and H. Sharp (2005a). Organisational Cultur and XP: Three Case Studies. Agile Developemnt Conference, IEEE Computer Soceity.

Robinson, H. and H. Sharp (2005b). The Social Side of Technical Practices. Extreme Programming and Agile Processes in Software Engineering, Springer

Royce, W. (1998). Software Project Management, Addison-Wesley.

Royce, W. (2002). CMM vs. CMMI: From Conventional to Modern Software Management Rational Software

Royce, W. W. (1970). Managing the Development of Large Software Systems Proceedings, IEEE WESCON 1-9.

Sanchez, J. C., L. Williams and E. M. Maximilien (2007). On the Sustained Use of a Test-Driven Development Practice at IBM. Proceedings of the AGILE 2007 (AGILE 2007) - Volume 00, IEEE Computer Society.

Schwaber, K. (1996). Controlled Chaos : Living on the Edge, Advanced Development Methods, Inc.

Schwaber, K. and M. Beedle (2001). Agile Software Development with Scrum, Prentice Hall PTR.

Seaman, C. B. (1999). "Qualitative Methods in Empirical Studies of Software Engineering." IEEE Transactions on Software Engineering 25(4): 557-572.

Seaman, C. B. and V. R. Basili (1998). "Communication and Organisation: An Empirical Study of Discussion in Inspection Meetings." IEEE Transactions on Software Engineering 24(7): 559-572.

Sfetsos, P., L. Angelis and I. Stamelos (2006). "Investigating the Extreme Programming System: An Empirical Study." Empirical Software Engineering 11(2): 269-301.

Sharp, H. and H. Robinson (2004a). An Ethnographic Study of XP Practice, Kluwer Academic Publishers. 9: 353-375.

Sharp, H., M. Woodman and F. Hovenden (2004b). Tensions around the Adoption and Evolution of Software Quality Management Systems: A Discourse Analytic Approach, Academic Press, Inc. 61: 219-236.

Siniaalto, M. and P. Abrahamsson (2007). A Comparative Case Study on the Impact of Test-Driven Development on Program Design and Test Coverage. International Symposium on Empirical Software Engineering and Measurement.

So, C. and W. Scholl (2009). Perceptive Agile Measurement: New Instruments for Quantitative Studies in Pursuit of the Social-Psychological Effect of Agile Practices. Agile Processes in Software Engineering and Extreme Programming: 83-93.

Sommerville, I. (2004). Software Engineering 7th Edition Pearson Addison Wesley

Spence, I. and K. Bittner (2005). What is Iterative Development?- Part 1: The Developer Perspective. IBM, Rational Technical Library.

Stamelos, I. G. and P. Sfetsos (2007). Agile Software Development Quality Assurance, IGI Global.

Stapleton, J. (1997). Dsdm: The Method in Practice, Addison-Wesley Longman Publishing Co., Inc.

Stevens, J. P. (1992). "Applied Multivariable Analysis of Variance Tests." Psychological Bulletin 88: 728-737.

Strong, K. C., R. C. Ringer and S. A. Taylor (2001). "THE* Rules of Stakeholder Satisfaction (* Timeliness, Honesty, Empathy) " Journal of Business Ethics Volume 32, Number 3: 219-230.

Tabachnick, B. G. and L. S. Fidell (2001). Using Multivariate Statistics, Bosten:Allyn and Bacon.

Talby, D. and Y. Dubinsky (2009). Governance of an Agile Software Project. Proceedings of the 2009 ICSE Workshop on Software Development Governance, IEEE Computer Society.

Thomas, D. (2006). "The Essential Unified Process (EssUP) - New Life for the Unified Process " Retrieved 28/07/2008, from http://www.ddj.com/architect/196702101.

Toth, K., P. Kruchten and T. Paine (1993). Modernizing ATC through Modern Software Methods, Proceedings of the 38th Annual Air Traffic Control Association Conference . Nashville, Tennessee.

Turner, R. and A. Jain (2002). Agile Meets CMMI: Culture Clash or Common Cause? , XP/Agile Universe 2002, LNCS 2418 153-165.

Versionone.com (2007). 2rd Annual Survey: 2007 “The State of Agile Development”.

versionone.com (2008). 3rd Annual Survey: 2008 “The State of Agile Development”.

Vriens, C. (2003). Certifying for CMM Level 2 and ISO9001 with XP@Scrum Proceeding of the Agile Development Conference (ADC'03)

Webb, P., C. Pollard and G. Ridley (2006). Attempting to Define IT Governance: Wisdom or Folly? Proceedings of the 39th Annual Hawaii International Conference on System Sciences - Volume 08, IEEE Computer Society.

WebBlog. (2005). "Two Motivational Metrics for Agile Teams." Retrieved 09/12/2009, from http://www.agileadvice.com/.

Weill, P. and J. W. Ross (2004). IT Governance on One page. CRSR Working Paper N.349.

Williams, L. (2004). A Survey of Agile Development Methodologies.

Williams, R. D. (1975). Managing the Evelopment of Reliable Software, ACM Press: 3-8.

Wohlin, C., P. Runeson, M. Host, M. C. Ohlsson, B. Regnell and A. Wessl (2000). Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers.

Wong, C. (1984). "A Successful Software Development." IEEE Transactions on Software Engineering 10(6).

"www.cmcrossroads.com." Retrieved 28/07/2008.

"www.versionone.com." Retrieved 28/07/2008.

Yin, R. K. (2003). Case Study Research Design and Methods, Sage Publications.

Zannier, C., G. Melnik and F. Maurer (2006). On the Success of Empirical Studies in the International Conference on Software Engineering. Proceeding of the 28th International Conference on Software Engineering. Shanghai, China, ACM Press.

Appendixes

Appendix A: Old Versions of the Waterfall Model

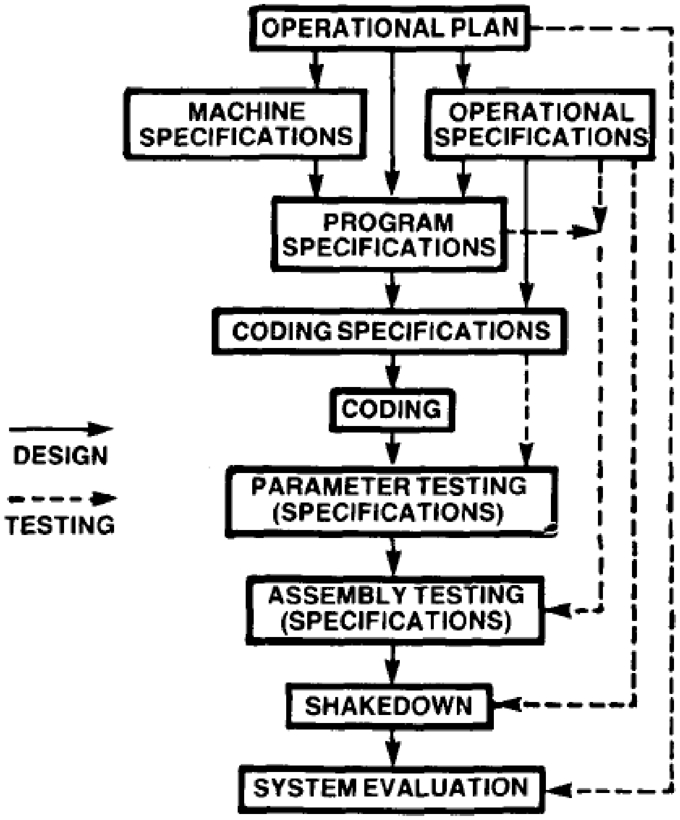


Figure A- 1 *Benington's 1956 Program Production as presented By him in 1983 (Benington 1983)*

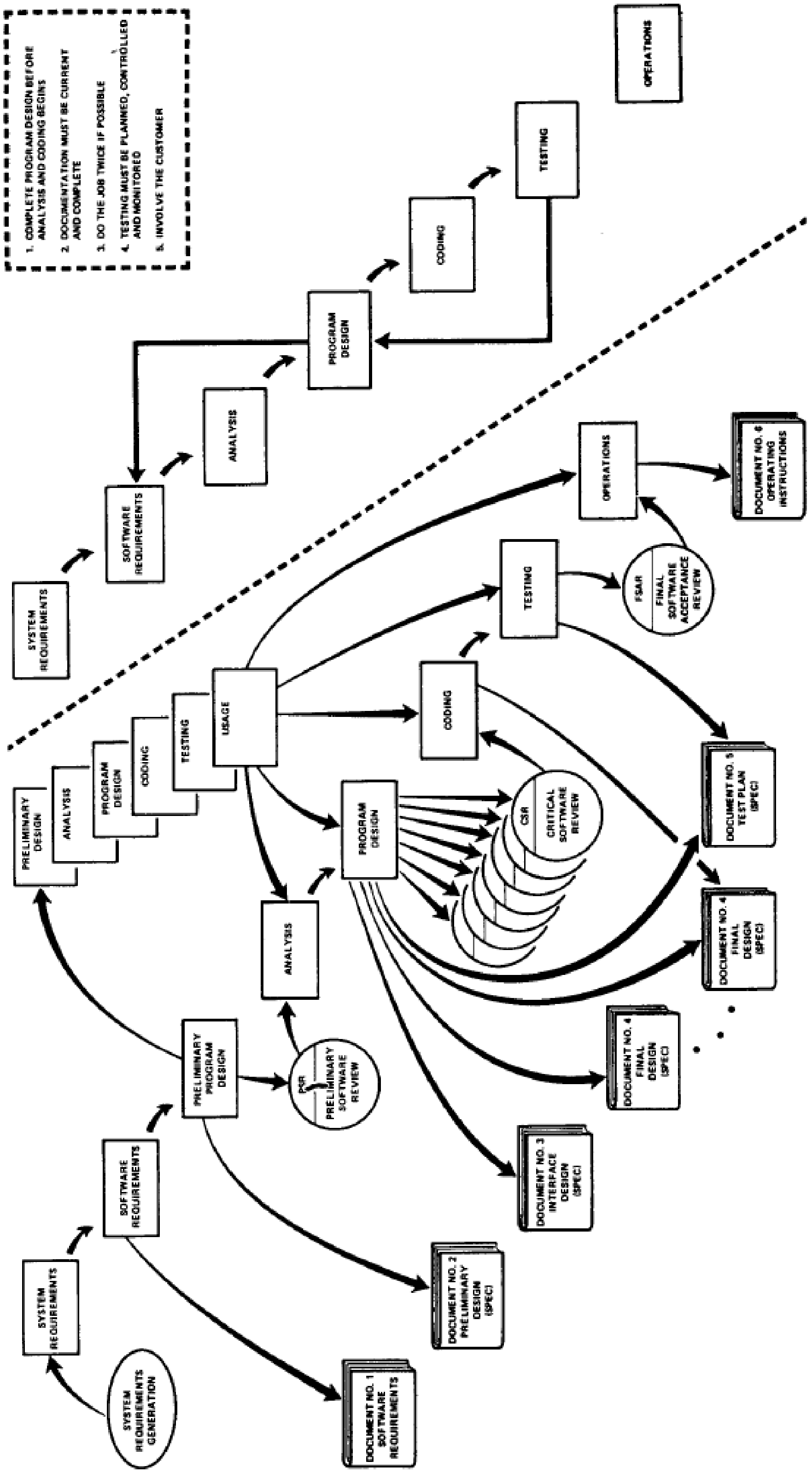

1. COMPLETE PROGRAM DESIGN BEFORE ANALYSIS AND CODING BEGINS
2. DOCUMENTATION MUST BE CURRENT AND COMPLETE
3. DO THE JOB TWICE IF POSSIBLE
4. TESTING MUST BE PLANNED, CONTROLLED AND MONITORED
5. INVOLVE THE CUSTOMER
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
OPERATIONS
SYSTEM REQUIREMENTS GENERATION
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
PRELIMINARY PROGRAM DESIGN
PSR
PRELIMINARY SOFTWARE REVIEW
PRELIMINARY DESIGN
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
USAGE
ANALYSIS
PROGRAM DESIGN
CODING
TESTING
OPERATIONS
CSR
CRITICAL SOFTWARE REVIEW
FSAR
FINAL SOFTWARE ACCEPTANCE REVIEW
DOCUMENT NO. 1 SOFTWARE REQUIREMENTS
DOCUMENT NO. 2 PRELIMINARY DESIGN (SPEC)
DOCUMENT NO. 3 INTERFACE DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 5 TEST PLAN (SPEC)
DOCUMENT NO. 6 OPERATING INSTRUCTIONS

Figure A- 2 *Royce's approach (Royce 1970)*

Appendix B: Principles behind the Agile Manifesto

We follow these principles:

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

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

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

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

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

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

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

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

The best architectures, requirements, and designs emerge from self-organising teams.

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

Appendix C: Agile Methods Lifecycles

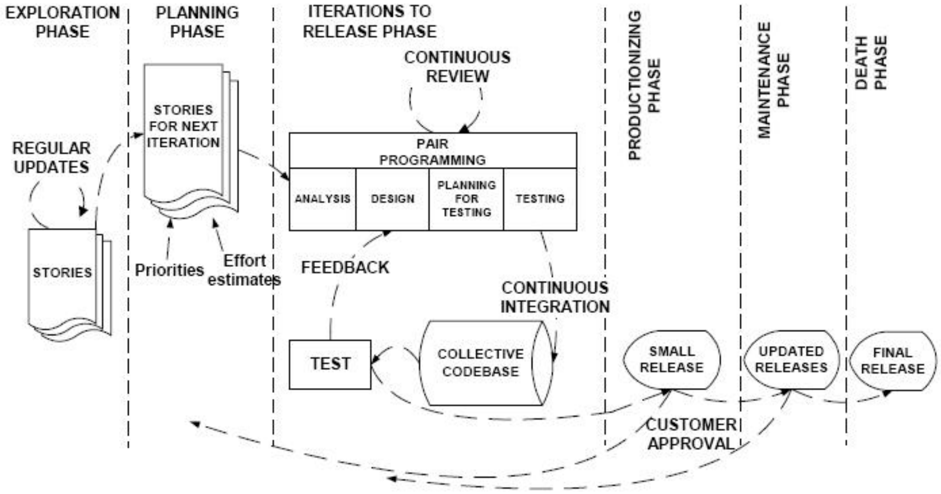


Figure *C-1* Lifecycle *of the XP Process (Abrahamsson et al. 2002)*

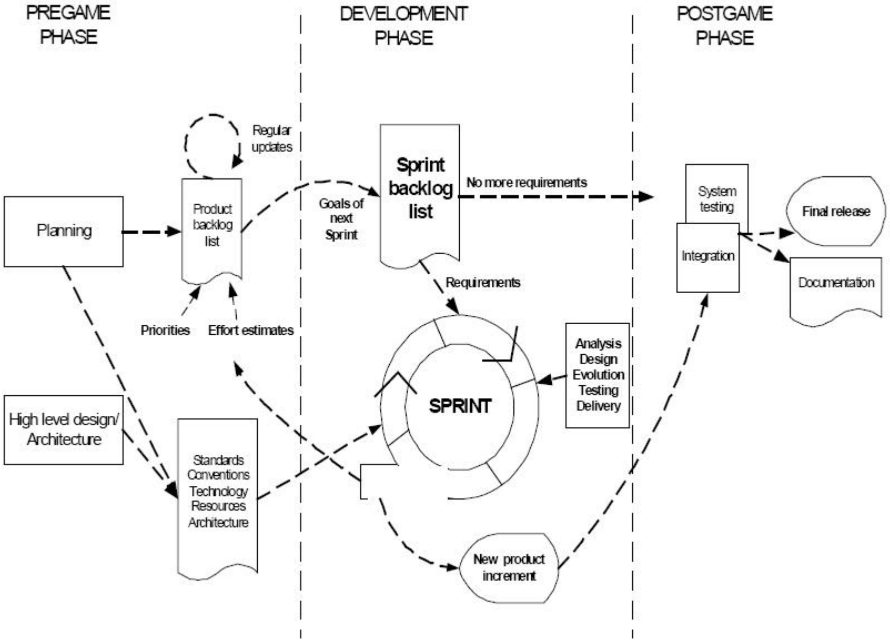


Figure C-2 Scrum Process (Abrahamsson et al. 2002)

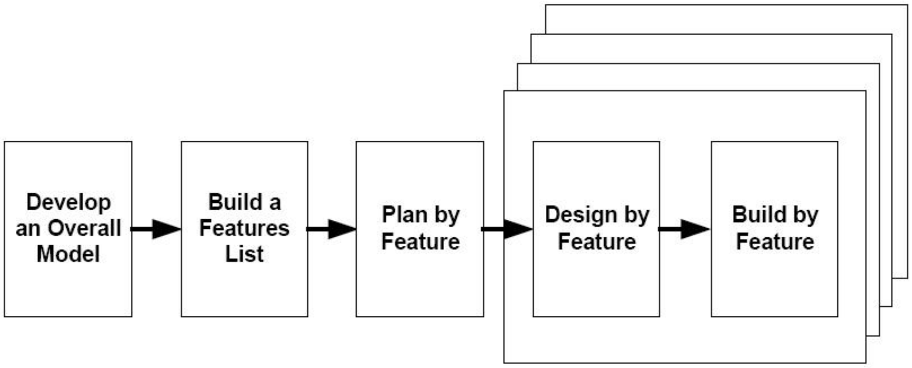


Figure C-3Process of FDD (Palmer et al. 2001)

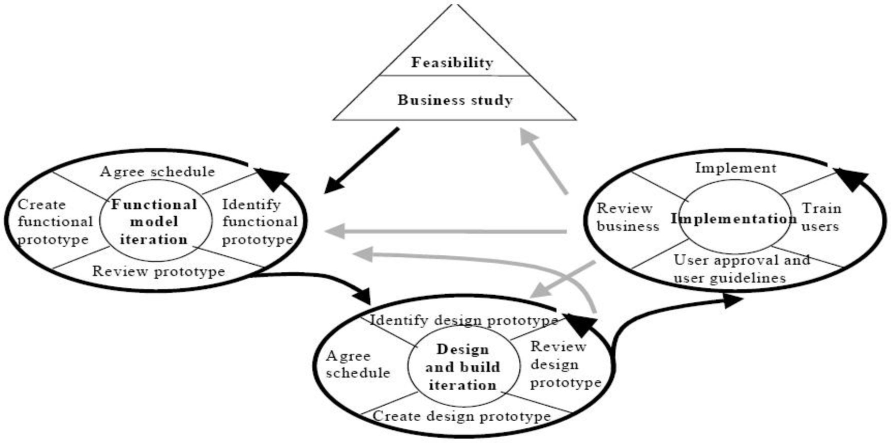

Feasibility
Business study
Agree schedule
Create functional prototype
Functional model iteration
Identify functional prototype
Review prototype
Implement
Review business
Implementation
Train users
User approval and user guidelines
Identify design prototype
Agree schedule
Design and build iteration
Review design prototype
Create design prototype

Figure C-4 DSDM Process Diagram (Stapleton 1997)

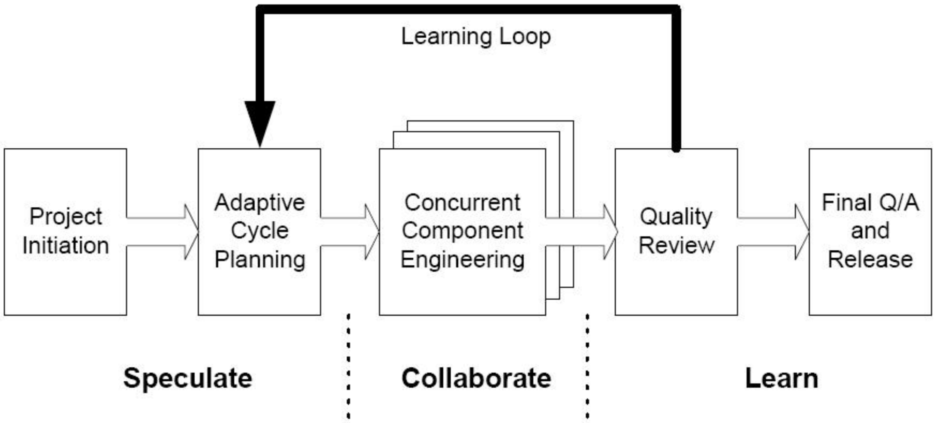
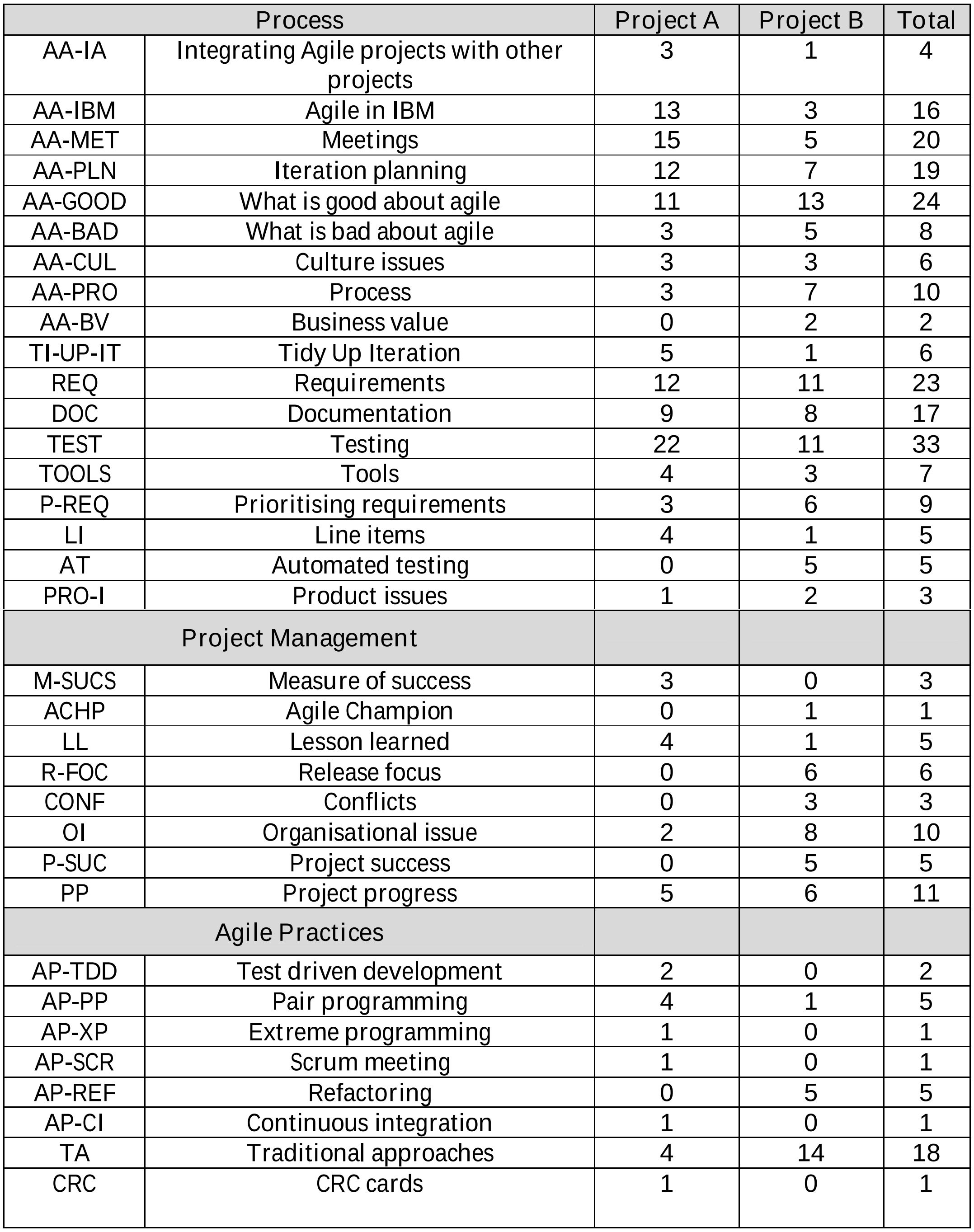


Figure C-5 ASD lifecycle phases (Highsmith 2000)

Appendix D: Empirical Study

General software development experience:
• Tell us about your previous experience in software development, how many projects, and what software development methodologies did you use for these projects • Did you use agile software development before, any particular agile method? • How do you rate the quality of an agile project? Do you think that agile software development is able to achieve high quality software?
Specific questions about the current project:
• What is the purpose of the current project? • Could you explain the process you are using for this project? (iterate vs increment, planning, delivery, relation with the customer, communication with the team) • What is your role in this project? • Do you like to work in an agile way? Why?
As research evolved more detailed questions with more focus on quality were asked:
• In your opinion, which agile practice(s) contribute to product quality? Why? • Is there anyone on your team (eg your architect) who acts as a customer proxy? • How do other stakeholders (support, marketing) provide input and interact with your team? • Do you distinguish between requests for new features, changes to existing features, preventative maintenance (eg refactoring) and documentation? • How do you prioritise these different types of request when planning the next iteration/release? • What techniques do you use to run an iteration/release retrospective? • How do you decide which proposed changes to adopt? • How do you ensure that the team "owns" the development process? • How do you ensure that the team "owns" the project/product? • Would it be true to say that you have achieved an increase in productivity, quality and velocity relative to company norms? • How and why did senior management decide to roll out agile across the company? • How do you think this roll-out should be managed to maximise the chance of success?

Table D-1 A list of questions used in the interviews

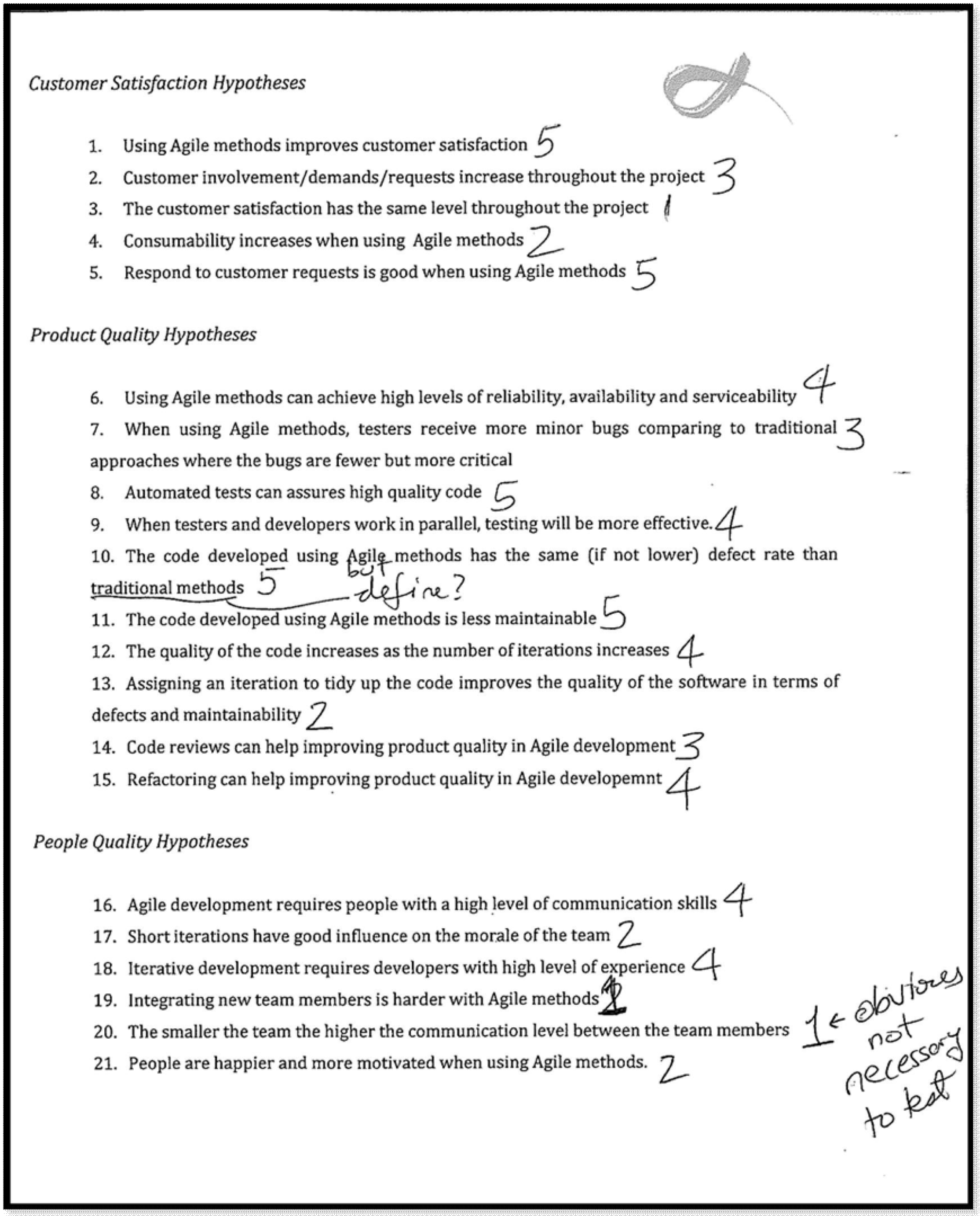


Process		Project A	Project B	Total
AA-IA	Integrating Agile projects with other projects	3	1	4
AA-IBM	Agile in IBM	13	3	16
AA-MET	Meetings	15	5	20
AA-PLN	Iteration planning	12	7	19
AA-GOOD	What is good about agile	11	13	24
AA-BAD	What is bad about agile	3	5	8
AA-CUL	Culture issues	3	3	6
AA-PRO	Process	3	7	10
AA-BV	Business value	0	2	2
TI-UP-IT	Tidy Up Iteration	5	1	6
REQ	Requirements	12	11	23
DOC	Documentation	9	8	17
TEST	Testing	22	11	33
TOOLS	Tools	4	3	7
P-REQ	Prioritising requirements	3	6	9
LI	Line items	4	1	5
AT	Automated testing	0	5	5
PRO-I	Product issues	1	2	3
Project Management				
M-SUCS	Measure of success	3	0	3
ACHP	Agile Champion	0	1	1
LL	Lesson learned	4	1	5
R-FOC	Release focus	0	6	6
CONF	Conflicts	0	3	3
OI	Organisational issue	2	8	10
P-SUC	Project success	0	5	5
PP	Project progress	5	6	11
Agile Practices				
AP-TDD	Test driven development	2	0	2
AP-PP	Pair programming	4	1	5
AP-XP	Extreme programming	1	0	1
AP-SCR	Scrum meeting	1	0	1
AP-REF	Refactoring	0	5	5
AP-CI	Continuous integration	1	0	1
TA	Traditional approaches	4	14	18
CRC	CRC cards	1	0	1

People Issues		Project A	Project B	Total
Team				
AA-CT	Communication within the team	8	18	26
DT-SKILLS	Development team skills	1	4	5
DT-ORG	Development team organisation	6	3	9
SP	Seating plan	5	3	8
TT-ORG	Test team organisation	1	0	1
MT	Moral of the team	2	12	14
AA-OST	Off-site teams	2	12	14
AA-DTS	Developing team skills	3	3	6
S-O-T	Size of the team	4	2	6
R-T-D	Relation between test team and development team	2	5	7
AA-SU	Share understanding	7	2	9
AA-OWN	Ownership	6	1	7
ROLES	Roles	15	8	23
PX	Previous experience	7	5	12
Customer				
AA-CC	Communication with the customer	4	7	11
AA-DC	Delivery to the customer	6	7	13

Quality		Project A	Project B	Total
Q-CODE	Quality of the code	3	6	9
Q-PPL	Quality of the people	2	2	4
Q-DEF	Defects	10	15	25
Q-CS	Customer satisfaction	7	8	15
Q-MEG	Quality measures	5	1	6
G-EN	Good Enough	2	5	7
CR	Code review	5	1	6
QA-STD	Quality standards	2	0	2
QA-PPL	Quality of the people	2	1	3
QA-PRO	Quality of the process	1	0	1
QA	Quality Assurance	2	0	2

Table D-2 List of codes and sub-codes for the empirical Study



Customer Satisfaction Hypotheses

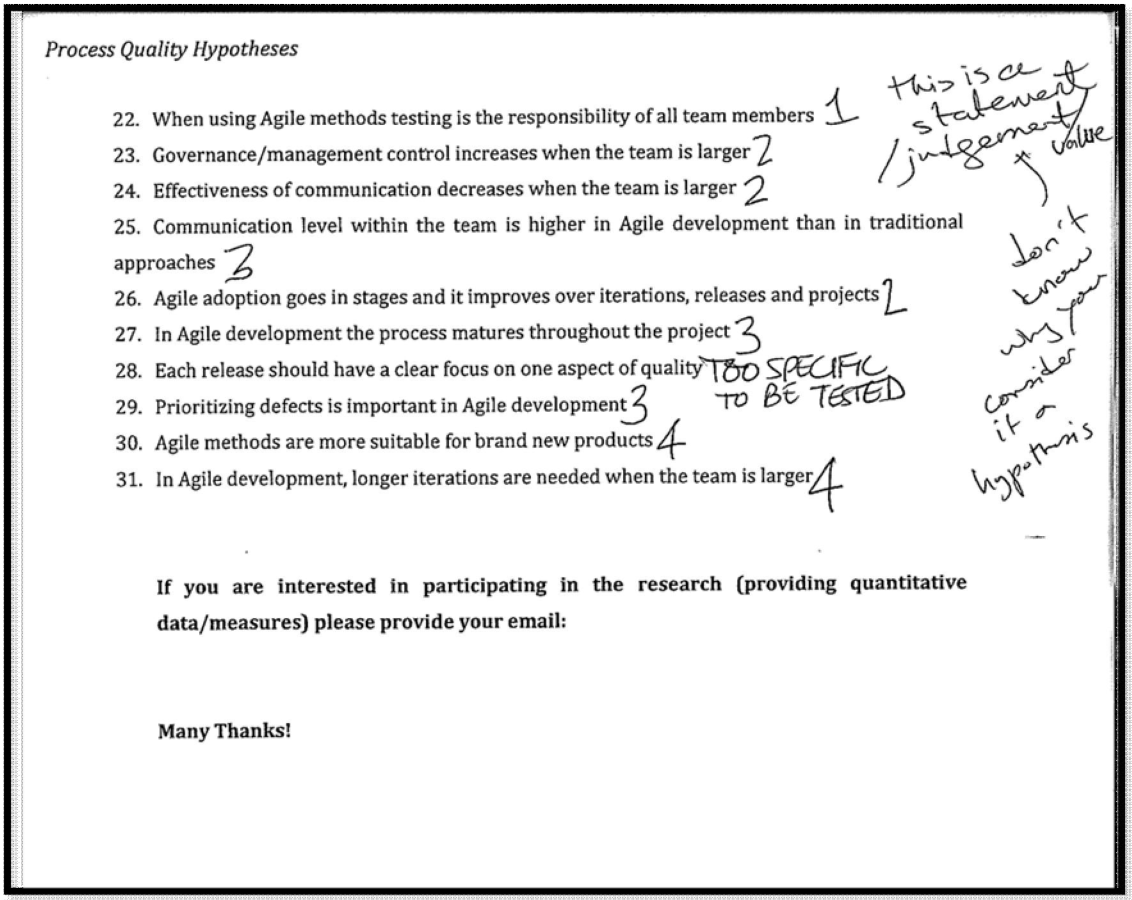
1. Using Agile methods improves customer satisfaction 5
2. Customer involvement/demands/requests increase throughout the project 3
3. The customer satisfaction has the same level throughout the project 1
4. Consumability increases when using Agile methods 2
5. Respond to customer requests is good when using Agile methods 5

Product Quality Hypotheses

6. Using Agile methods can achieve high levels of reliability, availability and serviceability 4
7. When using Agile methods, testers receive more minor bugs comparing to traditional approaches where the bugs are fewer but more critical 3
8. Automated tests can assures high quality code 5
9. When testers and developers work in parallel, testing will be more effective. 4
10. The code developed using Agile methods has the same (if not lower) defect rate than traditional methods 5 but define?
11. The code developed using Agile methods is less maintainable 5
12. The quality of the code increases as the number of iterations increases 4
13. Assigning an iteration to tidy up the code improves the quality of the software in terms of defects and maintainability 2
14. Code reviews can help improving product quality in Agile development 3
15. Refactoring can help improving product quality in Agile developemnt 4

People Quality Hypotheses

16. Agile development requires people with a high level of communication skills 4
17. Short iterations have good influence on the morale of the team 2
18. Iterative development requires developers with high level of experience 4
19. Integrating new team members is harder with Agile methods ~~4~~ 1
20. The smaller the team the higher the communication level between the team members 1 ← obvious not necessary to test
21. People are happier and more motivated when using Agile methods. 2



Process Quality Hypotheses

22. When using Agile methods testing is the responsibility of all team members 1
23. Governance/management control increases when the team is larger 2
24. Effectiveness of communication decreases when the team is larger 2
25. Communication level within the team is higher in Agile development than in traditional approaches 3
26. Agile adoption goes in stages and it improves over iterations, releases and projects 2
27. In Agile development the process matures throughout the project 3
28. Each release should have a clear focus on one aspect of quality TOO SPECIFIC TO BE TESTED
29. Prioritizing defects is important in Agile development 3
30. Agile methods are more suitable for brand new products 4
31. In Agile development, longer iterations are needed when the team is larger 4

this is a statement / judgement / & value

don't know why you consider it a hypothesis

If you are interested in participating in the research (providing quantitative data/measures) please provide your email:

Many Thanks!

Figure D-1 An example of the hypotheses evaluation forms during the agile 2008 conference

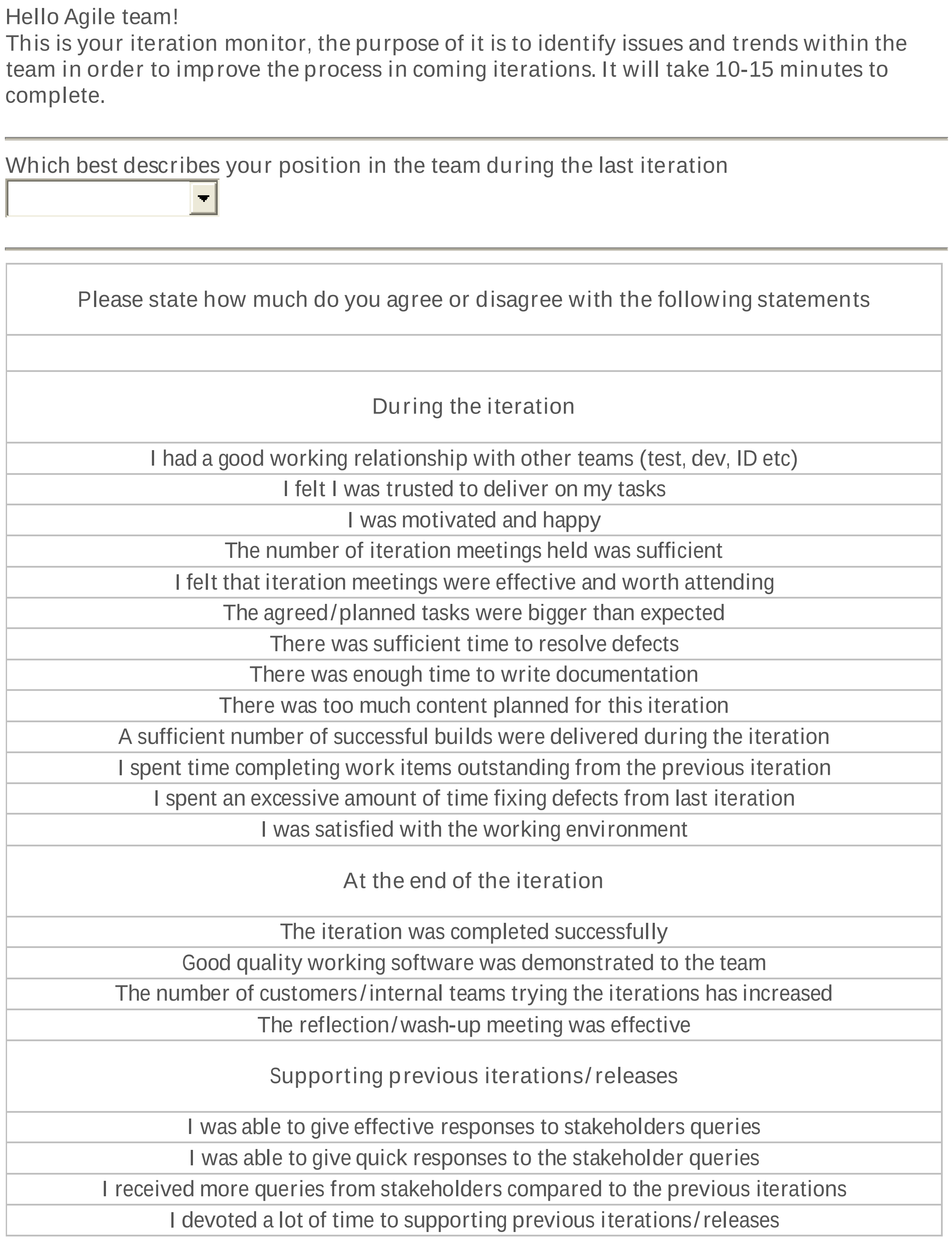
	# Of Valid Responses	# Of Missing Responses	Not interesting		Slightly interesting		Maybe interesting		Interesting		Very interesting	
			F	P	F	P	F	P	F	P	F	P
H1	33	8	1	3.0	4	12.1	4	12.1	7	21.2	16	48.5
H2	31	10	2	6.5	2	6.5	11	35.5	10	32.3	6	19.4
H3	29	12	10	34.5	8	27.6	7	24.1	2	6.9	2	6.9
H4	27	14	0	0.0	6	22.2	9	33.3	6	22.2	6	22.2
H5	33	8	1	3.0	3	9.1	6	18.2	10	30.3	13	39.4
H6	34	7	4	11.8	4	11.8	4	11.8	11	32.4	11	32.4
H7	31	10	6	19.4	5	16.1	8	25.8	8	25.8	3	9.7
H8	34	7	4	11.8	3	8.8	5	14.7	7	20.6	14	41.2
H9	32	9	1	3.1	3	9.4	3	9.4	8	25.0	17	53.1
H10	30	11	1	3.3	3	10.0	8	26.7	4	13.3	13	43.3
H11	35	6	5	14.3	5	14.3	5	14.3	7	20.0	13	37.1
H12	27	14	2	7.4	4	14.8	9	33.3	6	22.2	6	22.2
H13	29	12	1	3.4	5	17.2	8	27.6	7	24.1	7	24.1
H14	31	10	3	9.7	6	19.4	6	19.4	9	29.0	6	19.4
H15	31	10	3	9.7	2	6.5	3	9.7	11	35.5	11	35.5
H16	32	9	3	9.4	6	18.8	6	18.8	10	31.3	7	21.9
H17	31	10	2	6.5	5	16.1	5	16.1	15	48.4	4	12.9
H18	32	9	2	6.3	12	37.5	8	25.0	2	6.3	7	21.9
H19	30	11	7	23.3	6	20.0	10	33.3	3	10.0	4	13.3
H20	31	10	7	22.6	4	12.9	5	16.1	7	22.6	7	22.6
H21	31	10	4	12.9	2	6.5	8	25.8	7	22.6	9	29.0
H22	30	11	5	16.7	4	13.3	3	10.0	5	16.7	13	43.3
H23	28	13	1	3.6	4	14.3	13	46.4	4	14.3	5	17.9
H24	28	13	3	10.7	5	17.9	7	25.0	6	21.4	6	21.4
H25	27	14	2	7.4	3	11.1	8	29.6	6	22.2	8	29.6
H26	27	14	3	11.1	7	25.9	2	7.4	6	22.2	9	33.3
H27	27	14	2	7.4	5	18.5	6	22.2	7	25.9	4	14.8
H28	26	15	5	19.2	7	26.9	3	11.5	5	19.2	5	19.2
H29	28	13	3	10.7	3	10.7	9	32.1	5	17.9	8	28.6
H30	30	11	7	23.3	4	13.3	3	10.0	9	30.0	7	23.3
H31	31	10	9	29.0	5	16.1	4	12.9	7	22.6	6	19.4

Table D-3 Agile 2008 Research In Progress survey results

(The participants' opinions and feedback regarding the generated hypotheses

(F: Frequencies, P: Percentage)

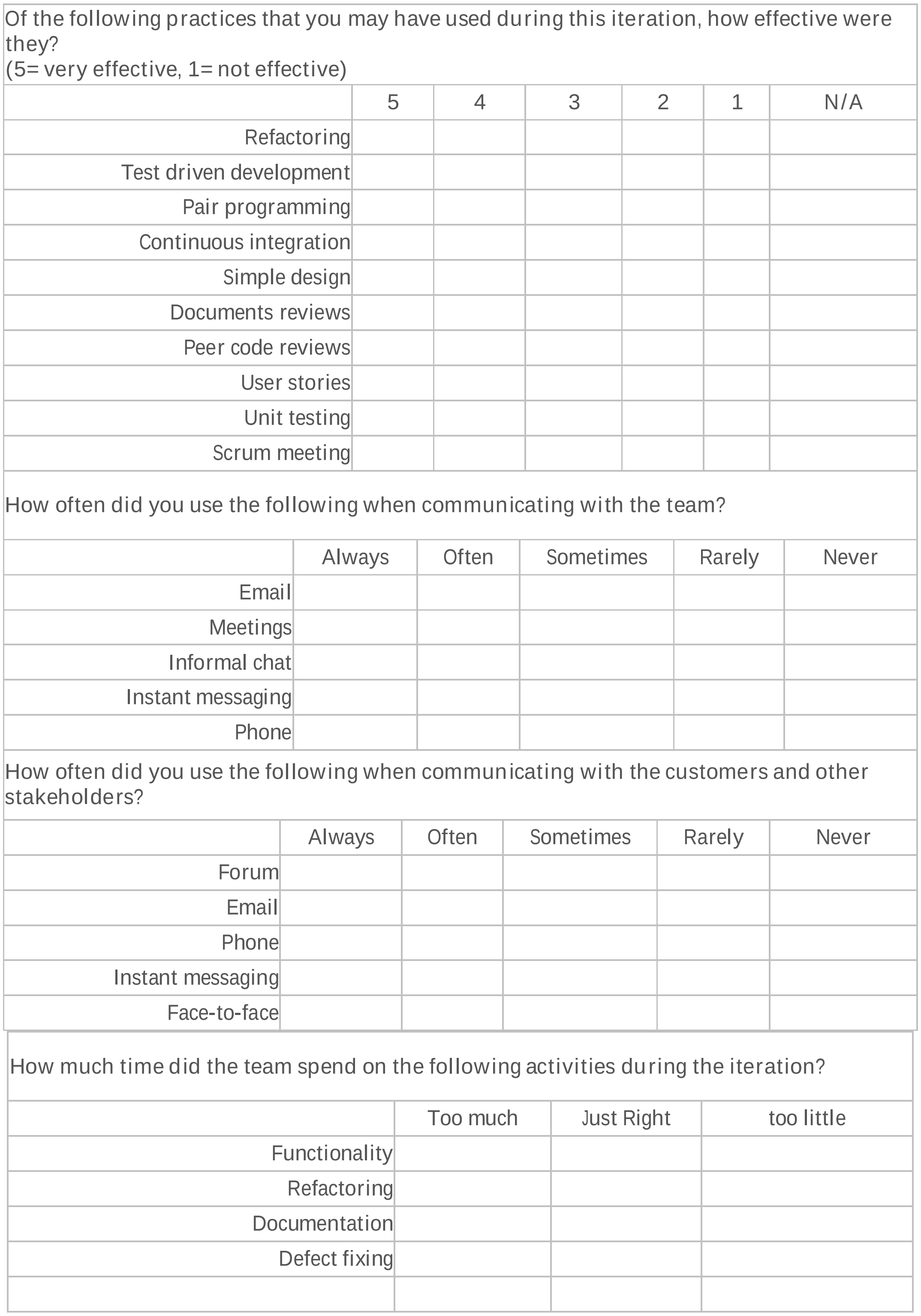
Appendix E: Iteration Monitor Design and Additional Results



Hello Agile team!
This is your iteration monitor, the purpose of it is to identify issues and trends within the team in order to improve the process in coming iterations. It will take 10-15 minutes to complete.

Which best describes your position in the team during the last iteration

Please state how much do you agree or disagree with the following statements
During the iteration
I had a good working relationship with other teams (test, dev, ID etc)
I felt I was trusted to deliver on my tasks
I was motivated and happy
The number of iteration meetings held was sufficient
I felt that iteration meetings were effective and worth attending
The agreed/planned tasks were bigger than expected
There was sufficient time to resolve defects
There was enough time to write documentation
There was too much content planned for this iteration
A sufficient number of successful builds were delivered during the iteration
I spent time completing work items outstanding from the previous iteration
I spent an excessive amount of time fixing defects from last iteration
I was satisfied with the working environment
At the end of the iteration
The iteration was completed successfully
Good quality working software was demonstrated to the team
The number of customers/internal teams trying the iterations has increased
The reflection/wash-up meeting was effective
Supporting previous iterations/releases
I was able to give effective responses to stakeholders queries
I was able to give quick responses to the stakeholder queries
I received more queries from stakeholders compared to the previous iterations
I devoted a lot of time to supporting previous iterations/releases



Of the following practices that you may have used during this iteration, how effective were they?
(5= very effective, 1= not effective)

	5	4	3	2	1	N/A
Refactoring						
Test driven development						
Pair programming						
Continuous integration						
Simple design						
Documents reviews						
Peer code reviews						
User stories						
Unit testing						
Scrum meeting						

How often did you use the following when communicating with the team?

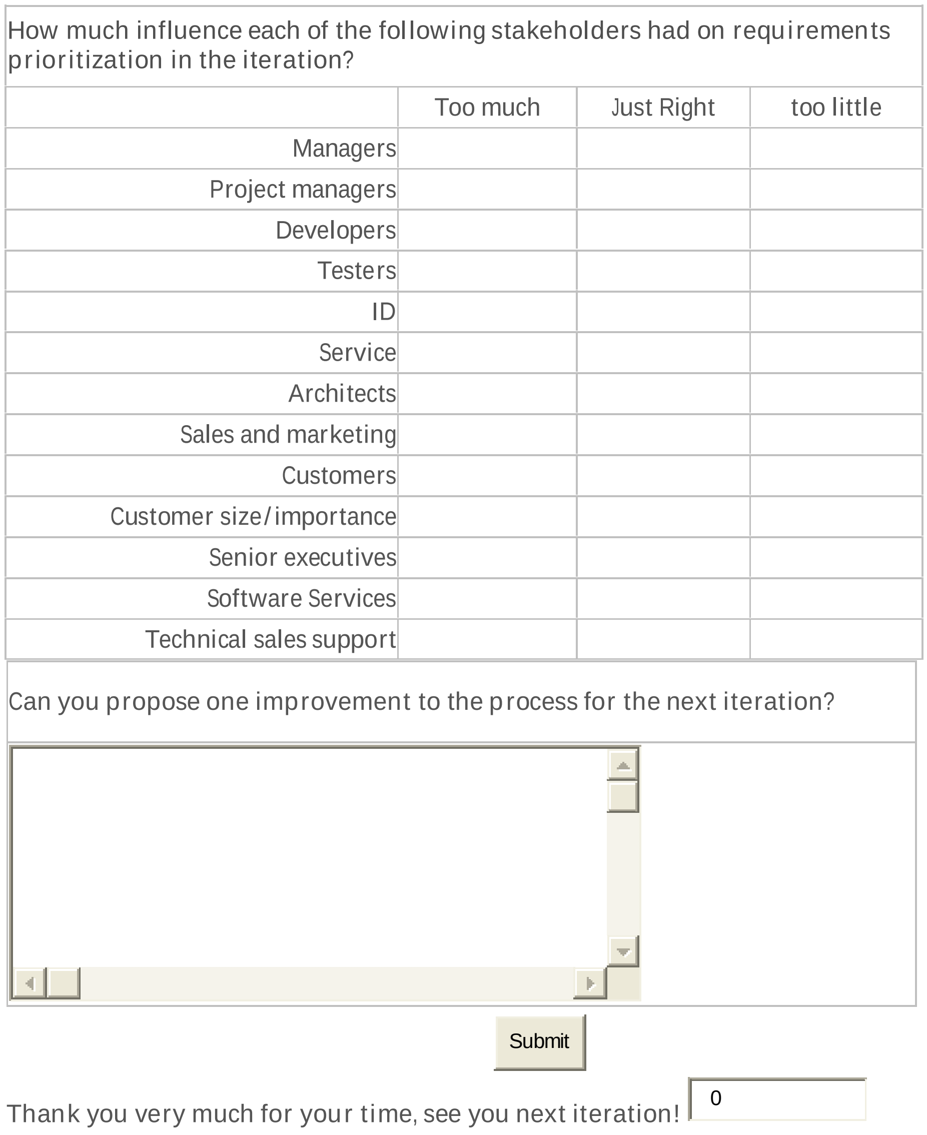
	Always	Often	Sometimes	Rarely	Never
Email					
Meetings					
Informal chat					
Instant messaging					
Phone					

How often did you use the following when communicating with the customers and other stakeholders?

	Always	Often	Sometimes	Rarely	Never
Forum					
Email					
Phone					
Instant messaging					
Face-to-face					

How much time did the team spend on the following activities during the iteration?

	Too much	Just Right	too little
Functionality			
Refactoring			
Documentation			
Defect fixing			



How much influence each of the following stakeholders had on requirements prioritization in the iteration?

	Too much	Just Right	too little
Managers			
Project managers			
Developers			
Testers			
ID			
Service			
Architects			
Sales and marketing			
Customers			
Customer size/importance			
Senior executives			
Software Services			
Technical sales support			

Can you propose one improvement to the process for the next iteration?

Thank you very much for your time, see you next iteration! 0

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1.000	.244	-.069	.282	.182	.344	-.136	-.113	.280	-.336	.028	-.207	.025	.212	.079	-.394	-.090	.428*	.158	-.238	.136
2		1.000	.406*	.313	.170	.478*	-.007	-.151	.153	.139	.075	-.027	.292	-.259	.000	-.088	-.101	.212	-.013	-.367	-.338
3			1.000	.559**	.038	.204	.380	.166	-.077	.369	-.106	.334	.640**	.069	.287	.206	.336	-.193	-.145	.165	-.136
4				1.000	.196	.512*	.165	-.048	.012	.069	-.054	.070	.458*	.041	.595**	-.057	.215	-.076	-.095	-.041	-.040
5					1.000	.318	-.204	-.070	-.047	.182	-.201	.000	.325	.270	.687**	.039	.073	-.049	-.187	-.102	-.203
6						1.000	-.380	-.458*	.426*	-.058	.054	.026	.279	-.381	.310	-.250	-.199	.050	-.064	-.425*	-.133
7							1.000	.612**	-.439*	.433*	-.315	.109	.148	.476*	-.009	.518**	.397	-.183	-.165	.321	-.261
8								1.000	-.463*	.468*	-448*	-.119	.031	.401	-.040	.372	.241	.051	.277	.442*	-.239
9									1.000	-.014	.462*	.116	.000	-.067	.043	-.387	-.193	.188	.058	-.282	-.088
10										1.000	-.330	.193	.208	.287	.112	.368	.010	-.018	.131	.130	-.554**
11											1.000	.117	.132	-.196	-.210	-.207	.046	.284	.171	-.042	.503*
12												1.000	.225	-.093	.117	-.291	.049	-.502*	-.297	.491*	.024
13													1.000	.095	.443*	.220	.259	-.054	-.007	.262	.231
14														1.000	.357	.422*	.340	.078	-.085	.170	-.210
15															1.000	.122	.330	-.255	-.340	.171	-.214
16																1.000	.500*	.024	-.105	.062	-.239
17																	1.000	-.098	-.193	.351	-.039
18																		1.000	.777**	-.363	.091
19																			1.000	.000	.249
20																				1.000	.252
21																					1.000

Table E-1 Correlations between iteration variables

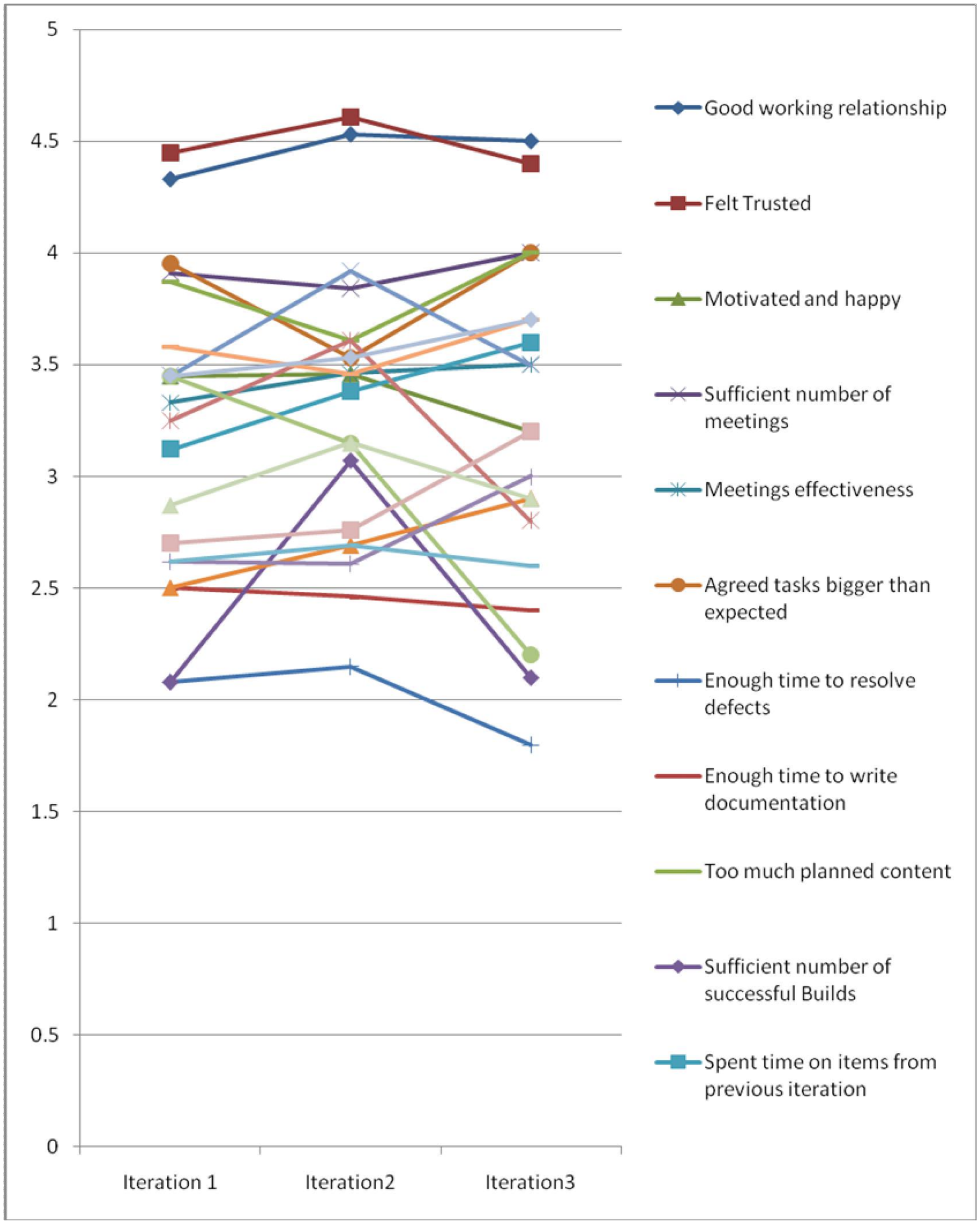


Figure E-1 Comparing iteration questions means

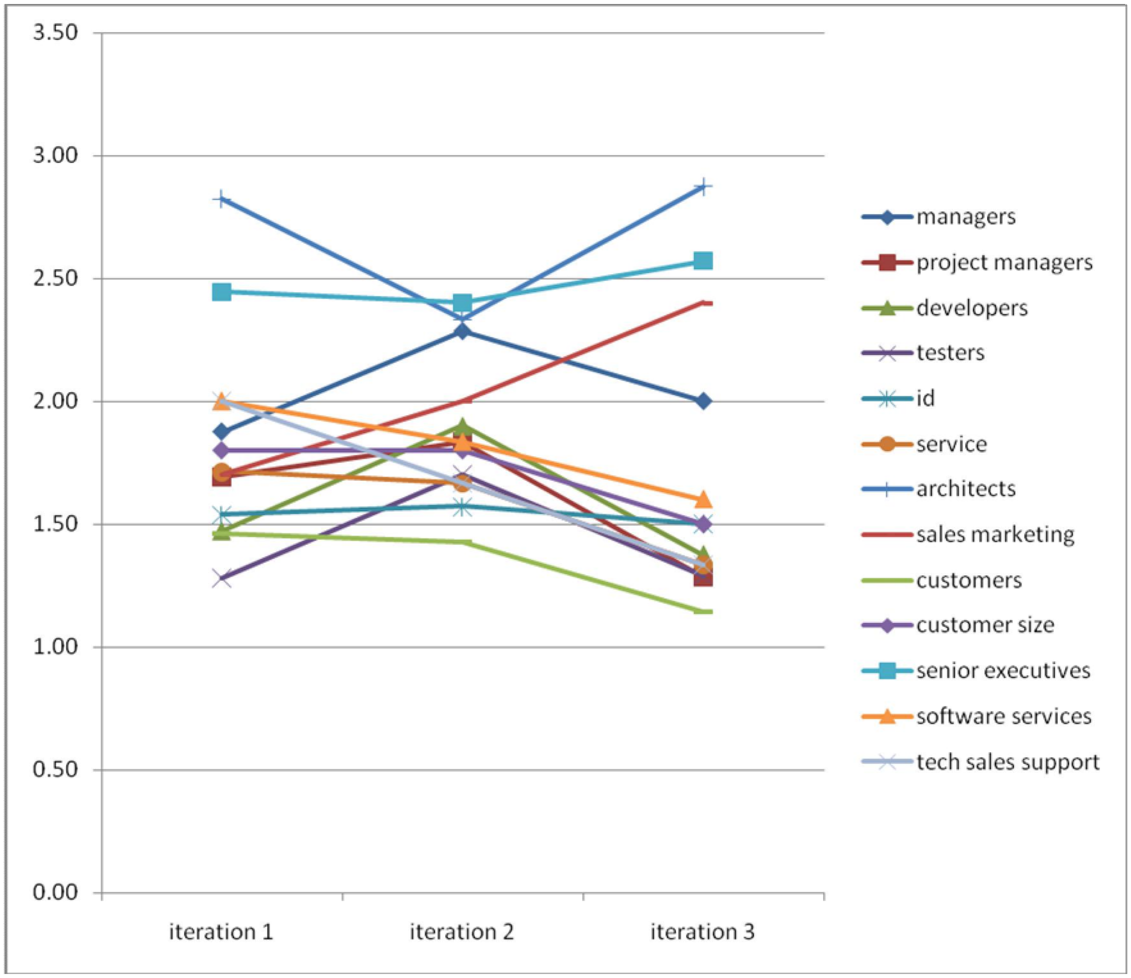


Figure E-2 Comparing the stakeholders influence on requirements prioritizing means

Test Statistics[b,c]

	Chi-Square	df	Asymp. Sig.
Good working relationship	1.678	2	.432
Felt Trusted	.813	2	.666
Motivated and happy	.303	2	.859
Sufficient number of meetings	.279	2	.870
Meetings effectiveness	.203	2	.903
Agreed tasks bigger than expected	2.756	2	.252
Enough time to resolve defects	2.145	2	.342
Enough time to write documentation	.105	2	.949
Too much planned content	8.810	2	.012
Sufficient number of successful Builds	2.351	2	.309
Spent time on items from previous iteration	1.415	2	.493
Spent time on fixing previous defects	2.932	2	.231
Satisfaction with work environment	3.213	2	.201
Iteration completed successfully	8.942	2	.011
Good quality work was demonstrated to the team	3.195	2	.202
Number of customers tried iteration increased	.063	2	.969
Reflection meeting effectiveness	.457	2	.796
Responses to stakeholder queries effectiveness	.803	2	.669
Quick responses to stakeholder queries	2.879	2	.237
Number of queries increased	.834	2	.659
functionality	.282	2	.868
refactoring	.409	2	.815
documentation	2.081	2	.353
Defect fix	10.196	2	.006
managers	2.464	2	.292
Project managers	3.364	2	.186
developers	4.686	2	.096
testers	5.101	2	.078
id	.064	2	.969
service	1.931	2	.381
architects	8.009	2	.018
Sales marketing	2.618	2	.270
customers	1.549	2	.461
Customer size	1.074	2	.584

Senior executives	.286	2	.867
Software services	1.212	2	.545
Tech sales support	3.146	2	.207
Too much	1.684	2	.431
a. Based on 10000 sampled tables with starting seed 1314643744.			
b. Kruskal Wallis Test			
c. Grouping Variable: iteration			

Table E-2 SPSS output for applying Kruskal-Wallis test to test the difference between the three iterations

Appendix F: Factor Analysis Extra Results

customer acceptance tests	.444				.442										
defect trend metrics	.443														
source code	.423														
Independent confirmatory exploratory testing	.419														
self organising teams	.416														
architectural spikes	.411														
continuous code integration	.410														
coding standard	.403														
static code analysis															
gantt chart details		.589													
architecture specification detailed		.567													
requirements specification details		.556													
gantt chart highlevel		.545								.443					
case tool modelling		.492													
use cases details	.419	.447													
burn down chart			.544												
continuous database integration	.422			-.424											
data naming conventions				-.401											
code inspection						.418									
test driven development	.430						.455								
configuration management															
co located team															-.465
planning game	.496														
active stakeholder participation	.494														
architecture specification highlevel	.487														
paper based modelling	.485														
sustainable pace	.481	-.434													
Velocity	.480		.473												
model document reviews	.479														
requirements specification highlevel	.471														
test plan	.471														
working demoable software	.469	-.435													
pair programming	.464														
defect reports	.447														
daily stand up meeting	.445		.408												

Table F-1 The component matrix

Component	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1.000	.256	.104	-.260	-.194	.150	.156	.087	.157	-.131	-.139	-.106	.063	.261	-.117
2	.256	1.000	.077	-.179	-.034	.031	.112	.188	.184	-.103	-.239	-.061	.136	-.005	.037
3	.104	.077	1.000	-.168	-.223	.191	.161	.107	.168	-.034	-.144	-.038	-.024	.216	-.184
4	-.260	-.179	-.168	1.000	.101	-.207	-.179	-.106	-.150	.096	.129	.143	-.036	-.180	.089
5	-.194	-.034	-.223	.101	1.000	-.171	-.078	-.089	-.083	.101	.129	.089	-.059	-.204	.190
6	.150	.031	.191	-.207	-.171	1.000	.164	.101	.084	-.063	-.097	-.074	.025	.301	-.162
7	.156	.112	.161	-.179	-.078	.164	1.000	.127	.059	-.100	-.068	-.045	.029	.107	-.051
8	.087	.188	.107	-.106	-.089	.101	.127	1.000	.099	-.071	-.149	-.089	.058	.041	-.061
9	.157	.184	.168	-.150	-.083	.084	.059	.099	1.000	-.044	-.195	-.051	.094	.139	-.096
10	-.131	-.103	-.034	.096	.101	-.063	-.100	-.071	-.044	1.000	.094	.058	-.039	-.075	.041
11	-.139	-.239	-.144	.129	.129	-.097	-.068	-.149	-.195	.094	1.000	.122	-.104	-.112	.049
12	-.106	-.061	-.038	.143	.089	-.074	-.045	-.089	-.051	.058	.122	1.000	-.022	.025	.015
13	.063	.136	-.024	-.036	-.059	.025	.029	.058	.094	-.039	-.104	-.022	1.000	.077	-.024
14	.261	-.005	.216	-.180	-.204	.301	.107	.041	.139	-.075	-.112	.025	.077	1.000	-.260
15	-.117	.037	-.184	.089	.190	-.162	-.051	-.061	-.096	.041	.049	.015	-.024	-.260	1.000

Table F-2 Component correlation matrix

	Component														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
initial agile architectural modelling	.758														
initial agile requirements modelling	.756														
evolutionary design	.501														
proved architecture early	.467														
flexible architecture															
gantt chart details		.883													
gantt chart highlevel		.845													
case tool modelling		.571													
architecture specification detailed		.502								-.449					
requirements specification details		.485													
burn down chart			.734												
Velocity			.718												
planning game			.629												
daily stand up meeting			.528												
iteration task list			.514												
defect trend metrics															
regular status report															
continuous database integration				-.826											
database testing				-.777											
database refactoring				-.771											
data naming conventions				-.480								-.445			
whiteboard sketches					-.752										
whiteboard sketching modelling					-.741										
working demoable software															
continuous code integration						.654									
test driven development						.587									
code refactoring						.581									
developer tests						.465									
simple design															
collective ownership															
paper based modelling							.701								
paper models							.624								
pair programming							.427								
static code analysis								.710							
code inspection								.664							
independent confirmatory exploratory testing									.585						
customer acceptance tests									.555						
model document reviews									.454						
architecture specification highlevel										-.569					
configuration management										.566					
test plan											-.678				
source code											-.541				
defect reports											-.506				

use cases details															
coding standard												-.702			
UI refactoring															
requirements specification highlevel													.535		
use cases light													.460		
architectural spikes															
UI testing															
incremental delivery														.796	
small releases														.786	
iterative development														.718	
sustainable pace														.554	
self organising teams															
co located team															-.794
active stakeholder participation															-.476

Rotation converged in 74 iterations.

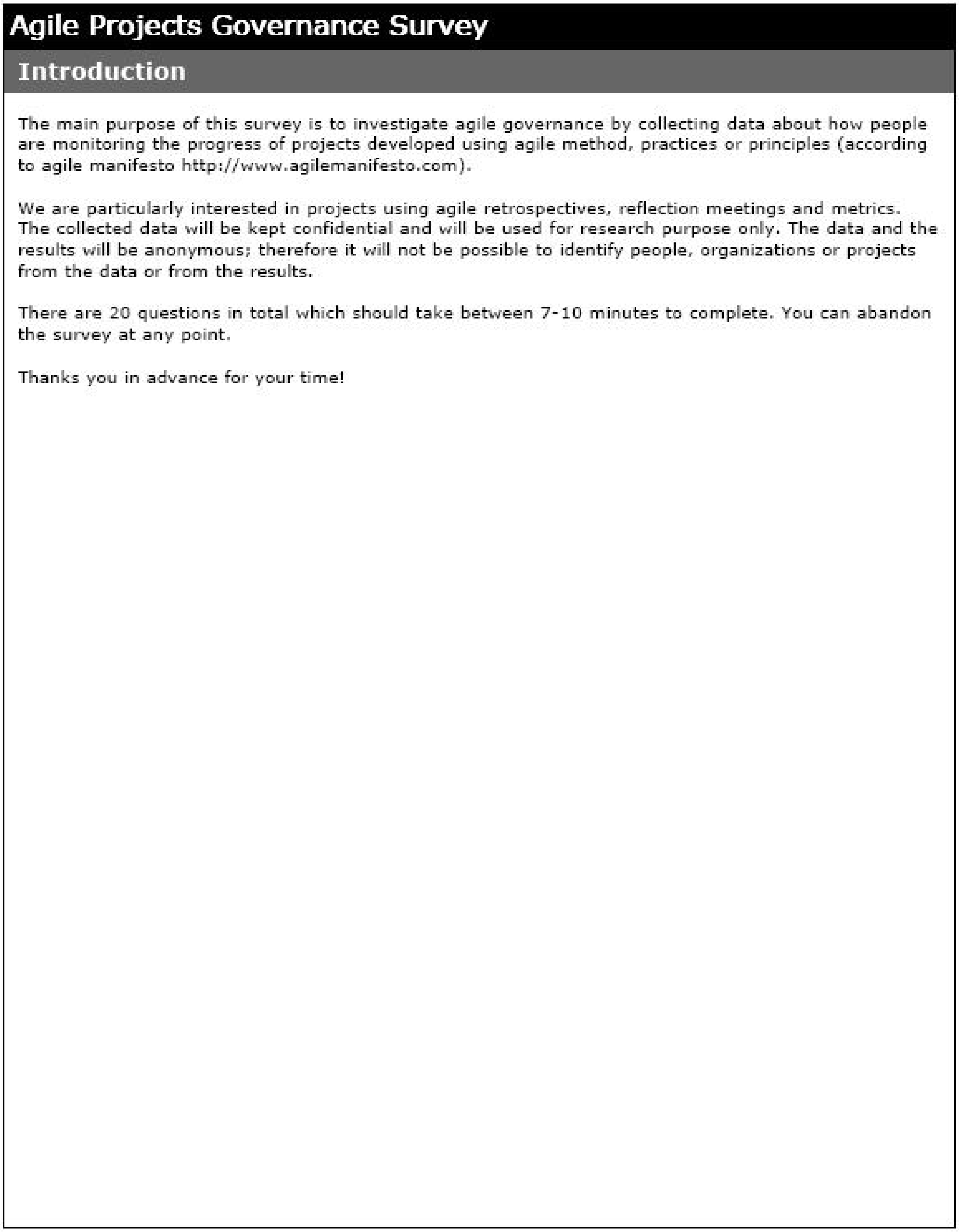
Table F-3 The pattern matrix

	Component														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
initial agile architectural modelling	.831														
initial agile requirements modelling	.817														
evolutionary design	.600					.464									
proved architecture early	.583													.404	
flexible architecture	.527			-.430		.446									
gantt chart details		.875													
gantt chart highlevel		.827													
case tool modelling		.657													
architecture specification detailed		.634								-.511					
requirements specification details		.606													
use cases details		.532									-.495		.425		
Velocity			.763												
burn down chart			.746												
planning game			.696												
daily stand up meeting			.641												
iteration tast list			.580		-.401										
regular status report			.512								-.463				
architectural spikes			.405												
database refactoring				-.834											
continuous database integration				-.819											
database testing				-.817											
data naming conventions				-.577								-.527			
UI refactoring				-.520								-.478			
whiteboard sketches					-.800										
whiteboard sketching modelling					-.799										
continuous code integration						.689									
code refactoring						.675									
test driven development						.649									
developer tests						.557									
simple design						.533								.461	
collective ownership						.508			.411					.464	
paper based modelling							.757								
paper models					-.407		.675								
pair programming						.449	.534								
static code analysis								.748							
code inspection								.714							
independent confirmatory exploratory testing									.639						
customer acceptance tests									.614						
model document reviews	.414								.558						
architecture specification highlevel	.497									-.648					
configuration management										.481					
test plan											-.731				
defect reports											-.607				
source code											-.566				
defect trend metrics			.454								-.535				
coding standard												-.730			
requirements specification highlevel													.596		

use cases light													.529		
UI testing				-.408									.440		
incremental delivery														.852	
iterative development						.403								.814	
small releases														.805	
sustainable pace														.670	
self organising teams			.424											.457	
co located team															-.786
active stakeholder participation														.528	-.592
working demoable software					-.496									.484	-.523

Table F-4 The structure matrix

Appendix G: Agile Projects Governance Survey Design and Extra Results



Agile Projects Governance Survey

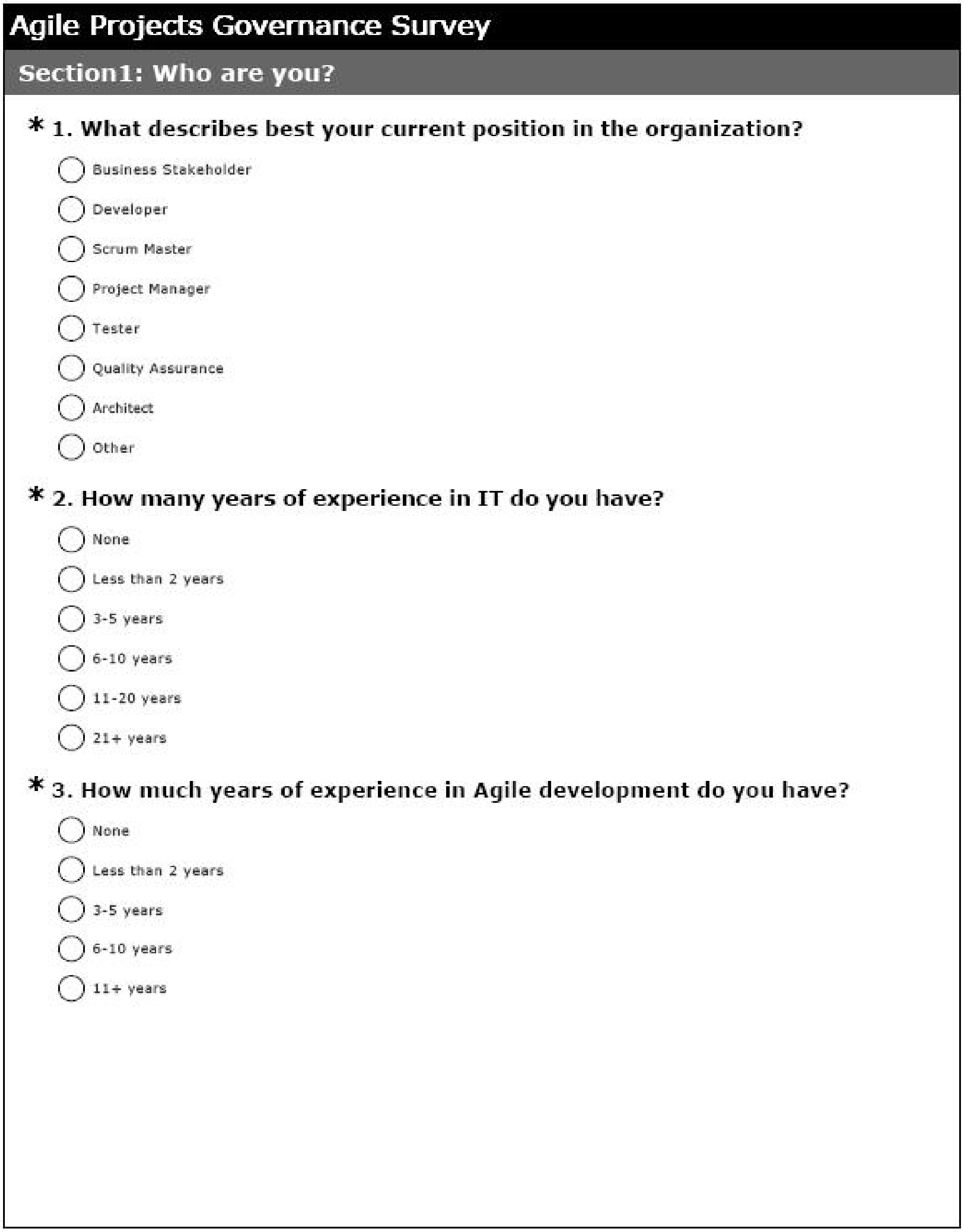
Introduction

The main purpose of this survey is to investigate agile governance by collecting data about how people are monitoring the progress of projects developed using agile method, practices or principles (according to agile manifesto http://www.agilemanifesto.com).

We are particularly interested in projects using agile retrospectives, reflection meetings and metrics. The collected data will be kept confidential and will be used for research purpose only. The data and the results will be anonymous; therefore it will not be possible to identify people, organizations or projects from the data or from the results.

There are 20 questions in total which should take between 7-10 minutes to complete. You can abandon the survey at any point.

Thanks you in advance for your time!



Agile Projects Governance Survey

Section1: Who are you?

* 1. What describes best your current position in the organization?

○ Business Stakeholder

○ Developer

○ Scrum Master

○ Project Manager

○ Tester

○ Quality Assurance

○ Architect

○ Other

* 2. How many years of experience in IT do you have?

○ None

○ Less than 2 years

○ 3-5 years

○ 6-10 years

○ 11-20 years

○ 21+ years

* 3. How much years of experience in Agile development do you have?

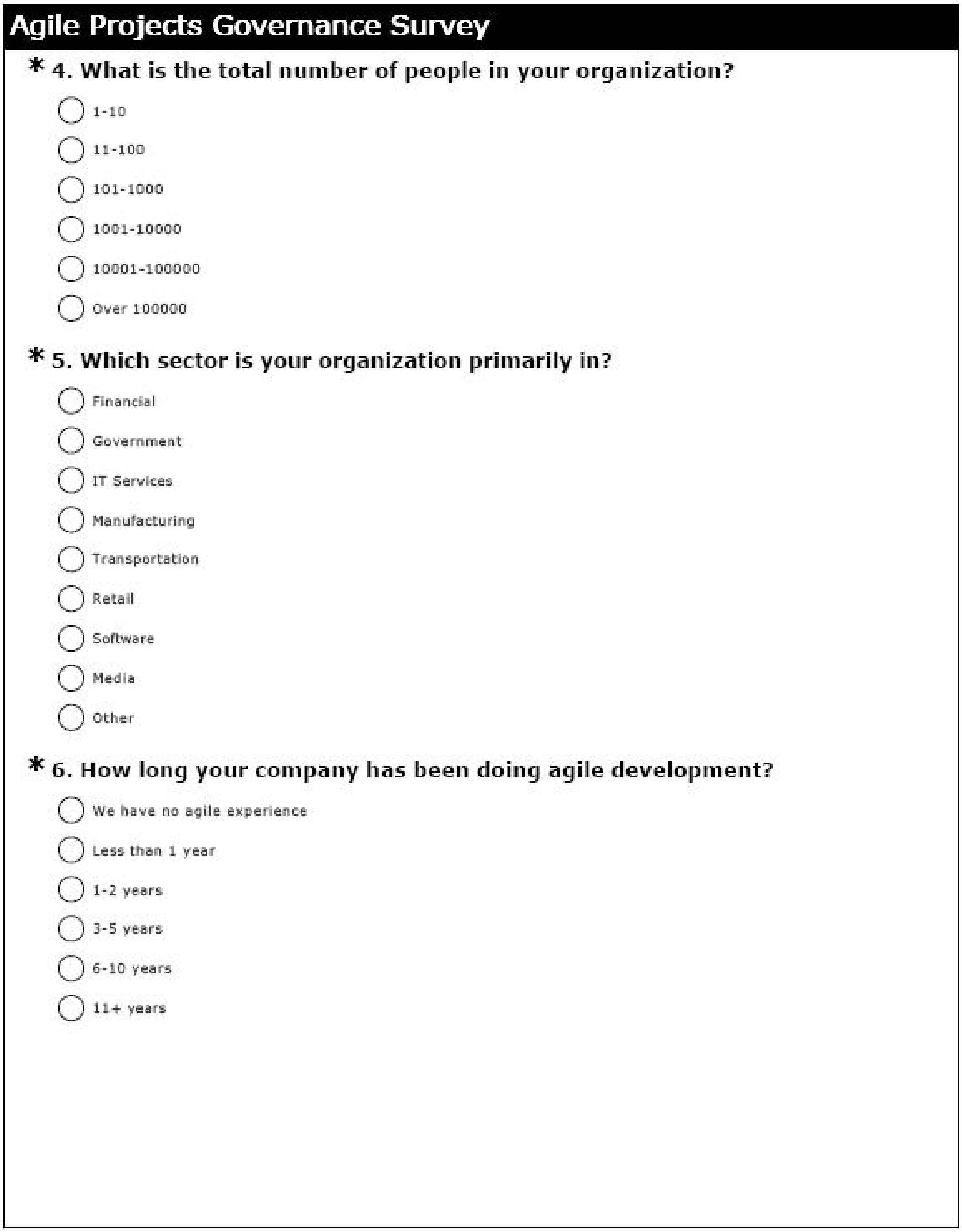
○ None

○ Less than 2 years

○ 3-5 years

○ 6-10 years

○ 11+ years



Agile Projects Governance Survey

*4. What is the total number of people in your organization?

○ 1-10

○ 11-100

○ 101-1000

○ 1001-10000

○ 10001-100000

○ Over 100000

*5. Which sector is your organization primarily in?

○ Financial

○ Government

○ IT Services

○ Manufacturing

○ Transportation

○ Retail

○ Software

○ Media

○ Other

*6. How long your company has been doing agile development?

○ We have no agile experience

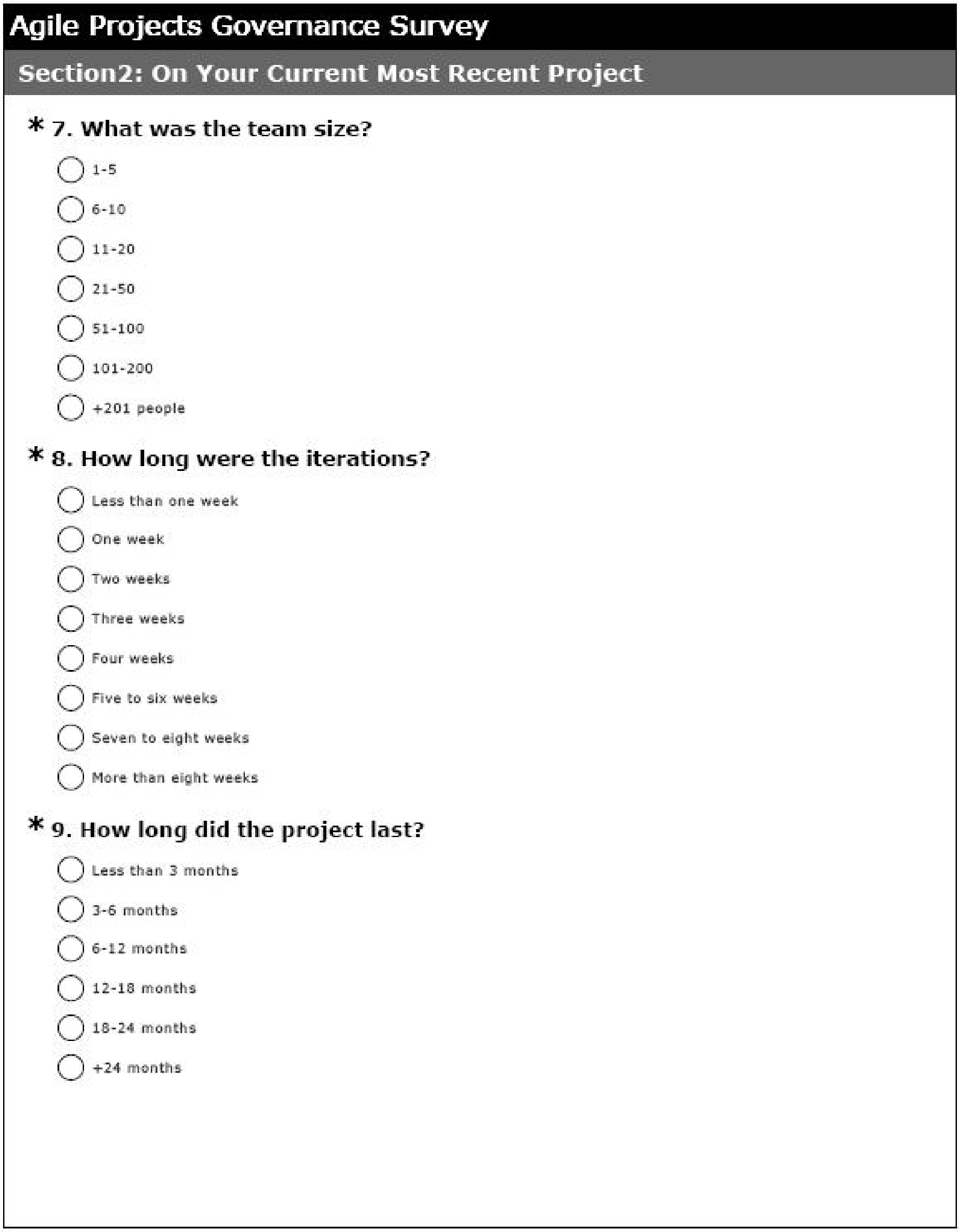
○ Less than 1 year

○ 1-2 years

○ 3-5 years

○ 6-10 years

○ 11+ years



Agile Projects Governance Survey

Section2: On Your Current Most Recent Project

* 7. What was the team size?

○ 1-5

○ 6-10

○ 11-20

○ 21-50

○ 51-100

○ 101-200

○ +201 people

* 8. How long were the iterations?

○ Less than one week

○ One week

○ Two weeks

○ Three weeks

○ Four weeks

○ Five to six weeks

○ Seven to eight weeks

○ More than eight weeks

* 9. How long did the project last?

○ Less than 3 months

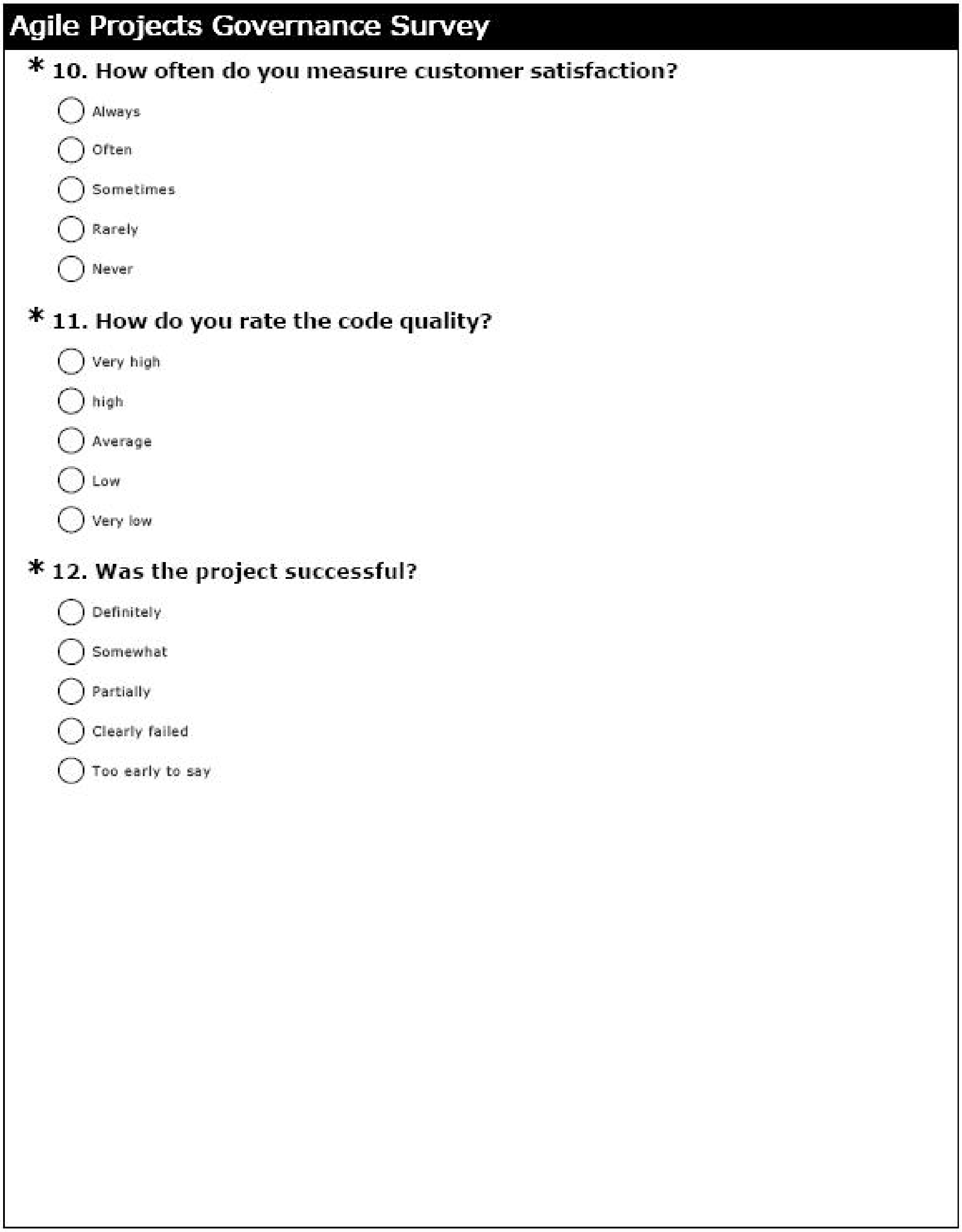
○ 3-6 months

○ 6-12 months

○ 12-18 months

○ 18-24 months

○ +24 months



Agile Projects Governance Survey

*** 10. How often do you measure customer satisfaction?**

○ Always

○ Often

○ Sometimes

○ Rarely

○ Never

*** 11. How do you rate the code quality?**

○ Very high

○ high

○ Average

○ Low

○ Very low

*** 12. Was the project successful?**

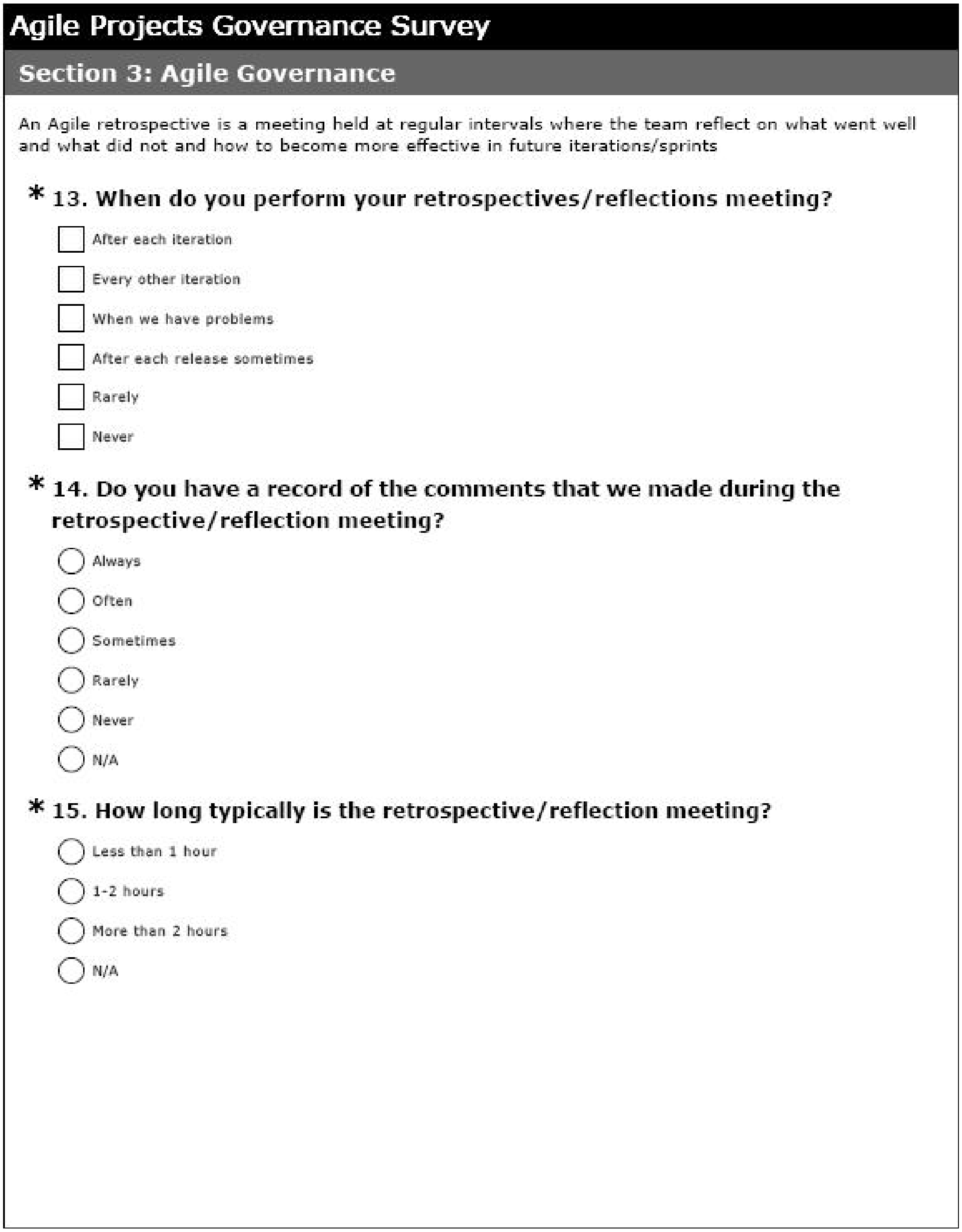
○ Definitely

○ Somewhat

○ Partially

○ Clearly failed

○ Too early to say



Agile Projects Governance Survey

Section 3: Agile Governance

An Agile retrospective is a meeting held at regular intervals where the team reflect on what went well and what did not and how to become more effective in future iterations/sprints

*** 13. When do you perform your retrospectives/reflections meeting?**

□ After each iteration

□ Every other iteration

□ When we have problems

□ After each release sometimes

□ Rarely

□ Never

*** 14. Do you have a record of the comments that we made during the retrospective/reflection meeting?**

○ Always

○ Often

○ Sometimes

○ Rarely

○ Never

○ N/A

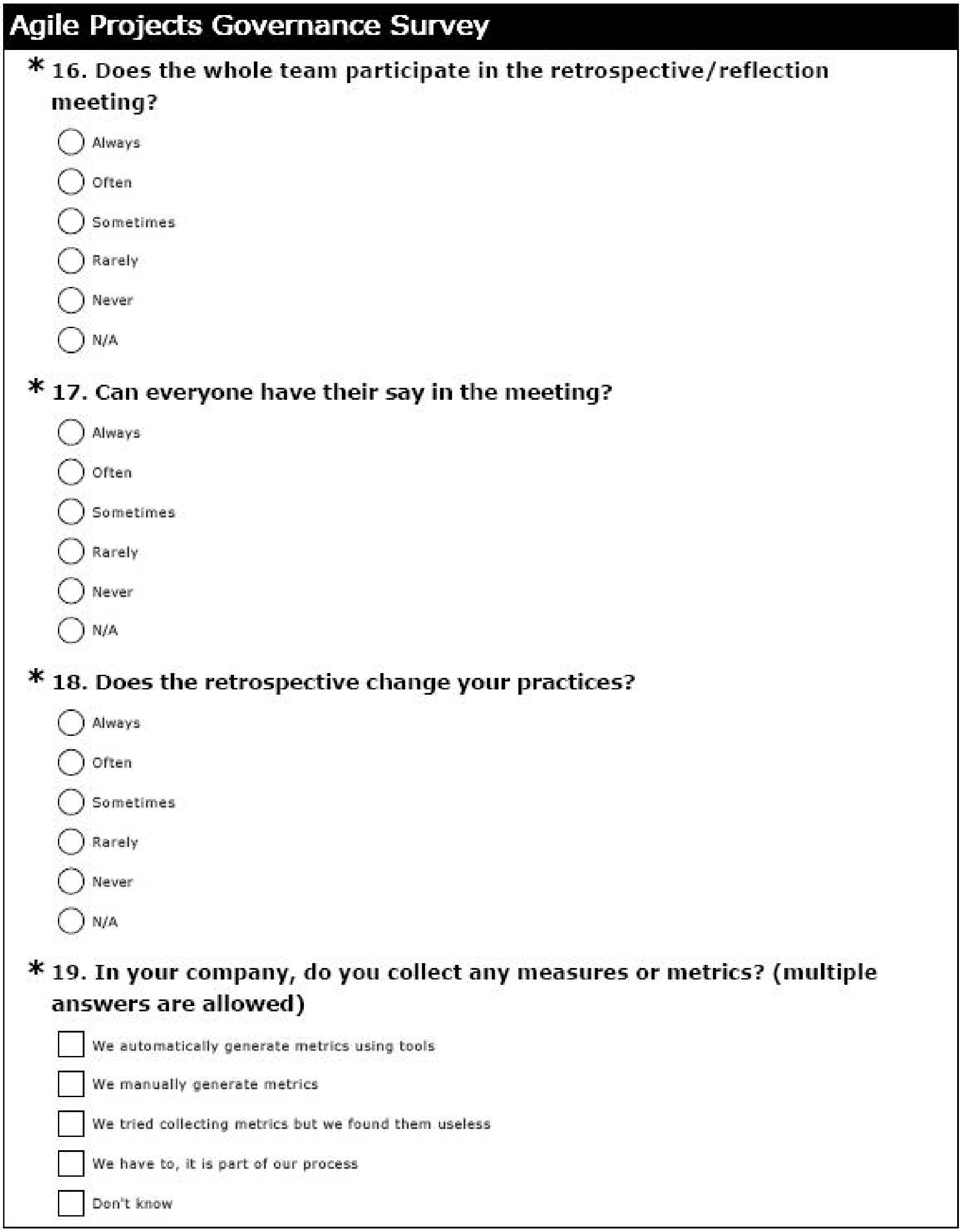
*** 15. How long typically is the retrospective/reflection meeting?**

○ Less than 1 hour

○ 1-2 hours

○ More than 2 hours

○ N/A



Agile Projects Governance Survey

*** 16. Does the whole team participate in the retrospective/reflection meeting?**

○ Always

○ Often

○ Sometimes

○ Rarely

○ Never

○ N/A

*** 17. Can everyone have their say in the meeting?**

○ Always

○ Often

○ Sometimes

○ Rarely

○ Never

○ N/A

*** 18. Does the retrospective change your practices?**

○ Always

○ Often

○ Sometimes

○ Rarely

○ Never

○ N/A

*** 19. In your company, do you collect any measures or metrics? (multiple answers are allowed)**

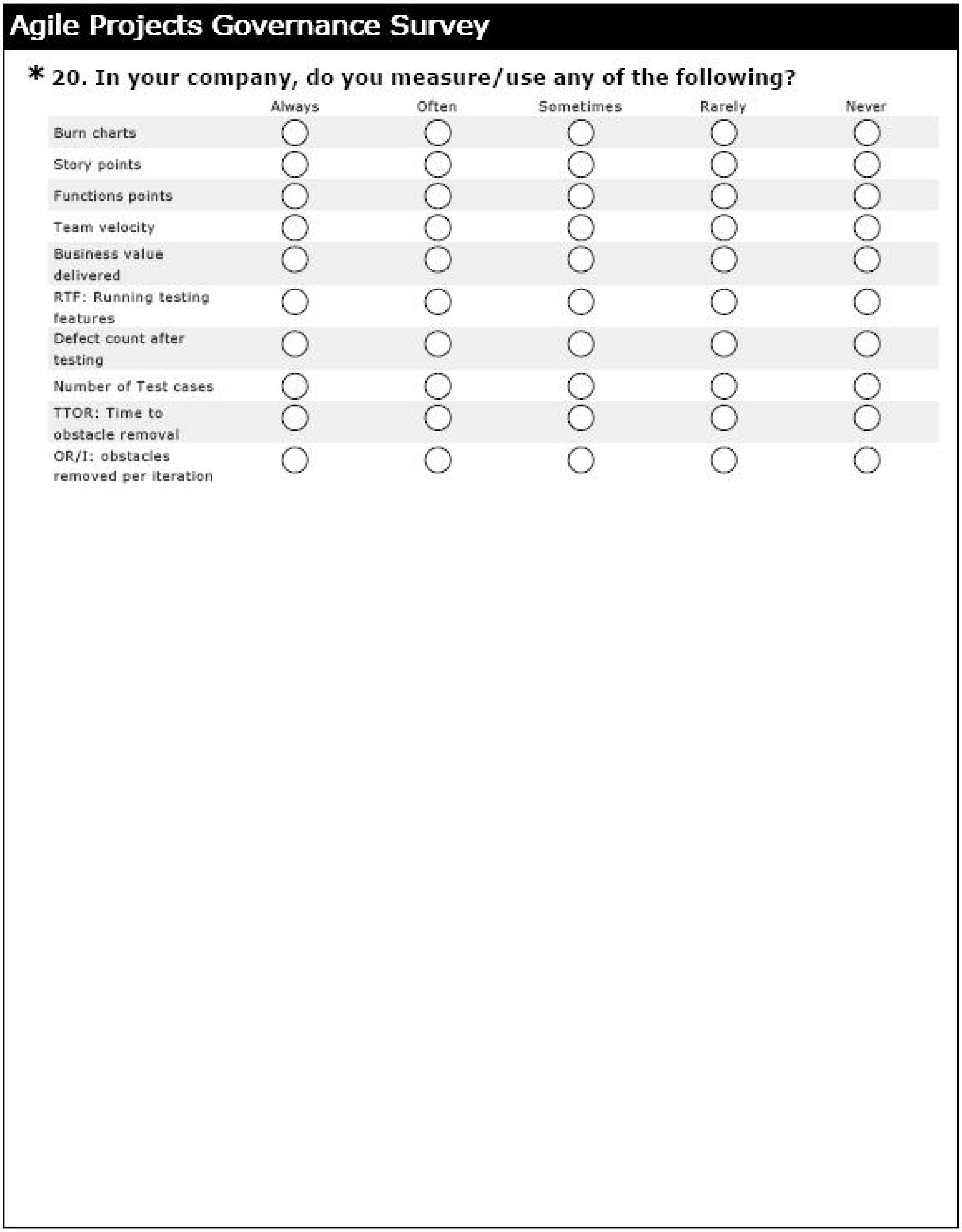
□ We automatically generate metrics using tools

□ We manually generate metrics

□ We tried collecting metrics but we found them useless

□ We have to, it is part of our process

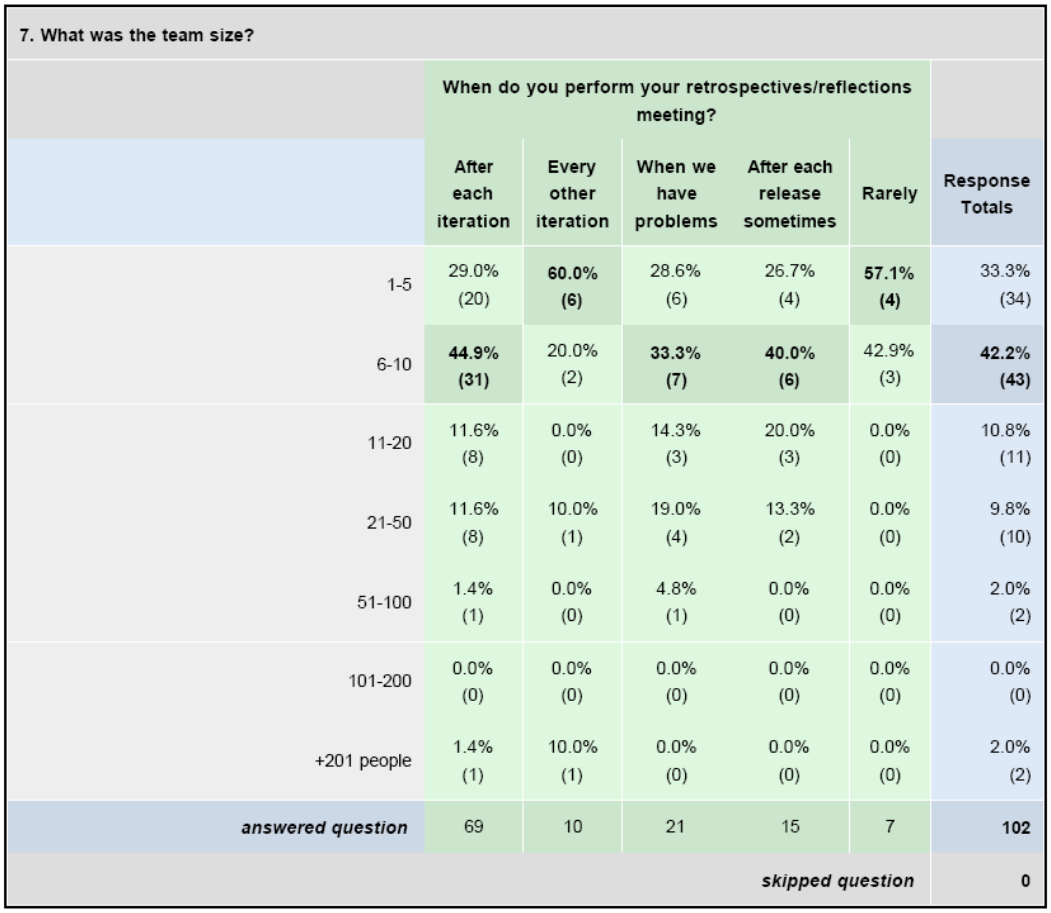
□ Don't know



Agile Projects Governance Survey

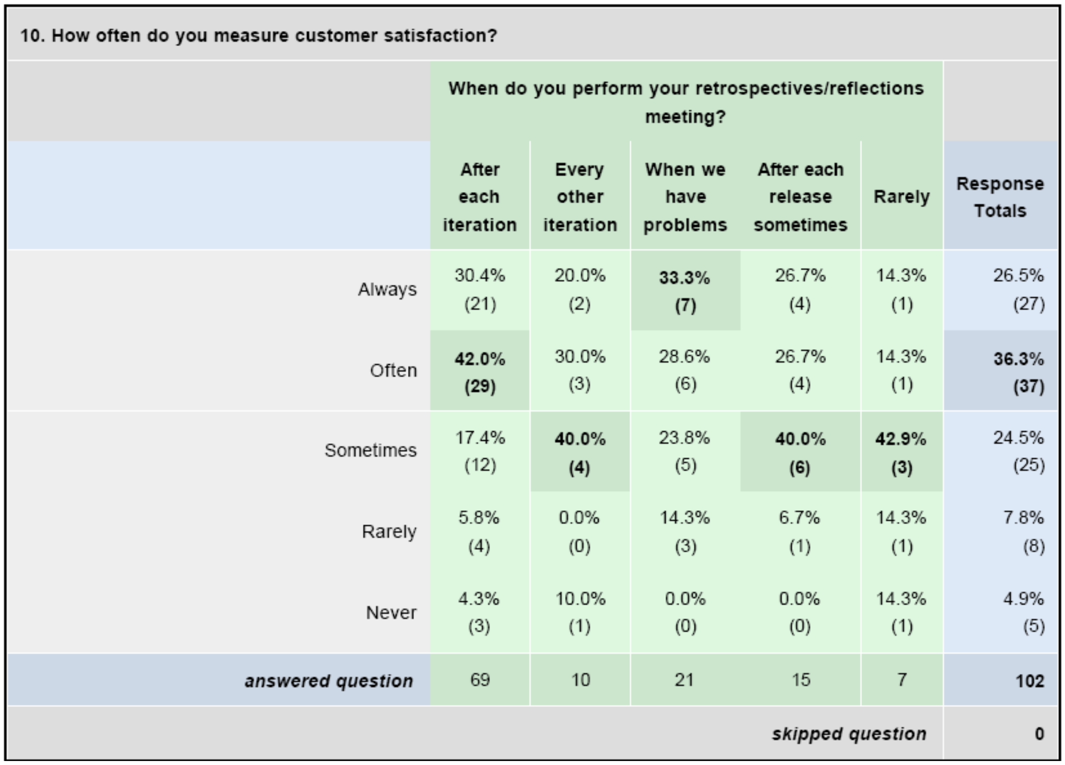
* 20. In your company, do you measure/use any of the following?

	Always	Often	Sometimes	Rarely	Never
Burn charts	○	○	○	○	○
Story points	○	○	○	○	○
Functions points	○	○	○	○	○
Team velocity	○	○	○	○	○
Business value delivered	○	○	○	○	○
RTF: Running testing features	○	○	○	○	○
Defect count after testing	○	○	○	○	○
Number of Test cases	○	○	○	○	○
TTOR: Time to obstacle removal	○	○	○	○	○
OR/I: obstacles removed per iteration	○	○	○	○	○



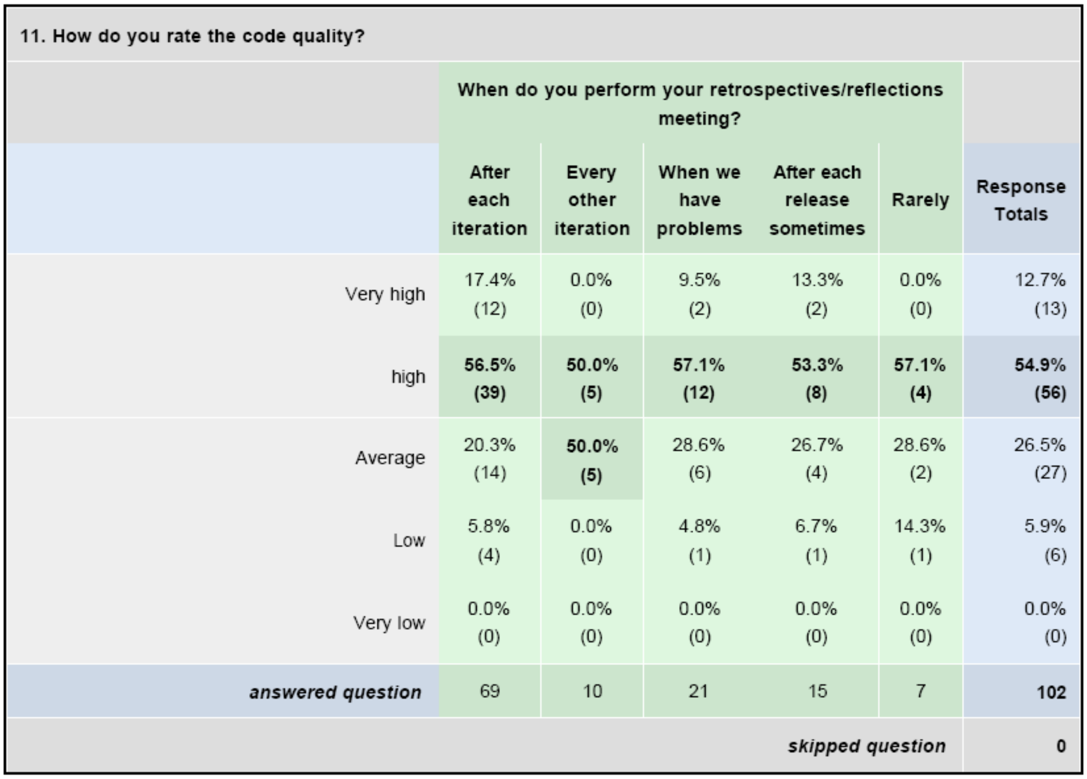
7. What was the team size?

	When do you perform your retrospectives/reflections meeting?					
	After each iteration	Every other iteration	When we have problems	After each release sometimes	Rarely	Response Totals
1-5	29.0% (20)	**60.0%** **(6)**	28.6% (6)	26.7% (4)	**57.1%** **(4)**	33.3% (34)
6-10	**44.9%** **(31)**	20.0% (2)	**33.3%** **(7)**	**40.0%** **(6)**	42.9% (3)	**42.2%** **(43)**
11-20	11.6% (8)	0.0% (0)	14.3% (3)	20.0% (3)	0.0% (0)	10.8% (11)
21-50	11.6% (8)	10.0% (1)	19.0% (4)	13.3% (2)	0.0% (0)	9.8% (10)
51-100	1.4% (1)	0.0% (0)	4.8% (1)	0.0% (0)	0.0% (0)	2.0% (2)
101-200	0.0% (0)	0.0% (0)	0.0% (0)	0.0% (0)	0.0% (0)	0.0% (0)
+201 people	1.4% (1)	10.0% (1)	0.0% (0)	0.0% (0)	0.0% (0)	2.0% (2)
answered question	69	10	21	15	7	**102**
					skipped question	**0**



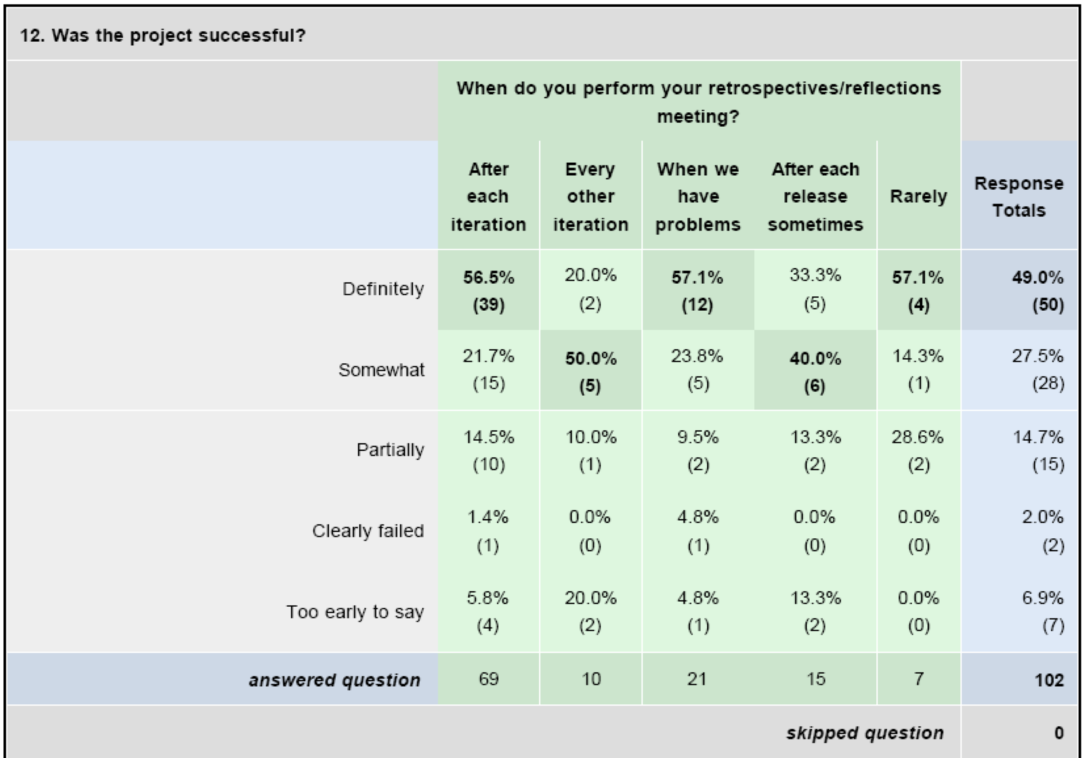
10. How often do you measure customer satisfaction?

	When do you perform your retrospectives/reflections meeting?					
	After each iteration	Every other iteration	When we have problems	After each release sometimes	Rarely	Response Totals
Always	30.4% (21)	20.0% (2)	**33.3%** **(7)**	26.7% (4)	14.3% (1)	26.5% (27)
Often	**42.0%** **(29)**	30.0% (3)	28.6% (6)	26.7% (4)	14.3% (1)	**36.3%** **(37)**
Sometimes	17.4% (12)	**40.0%** **(4)**	23.8% (5)	**40.0%** **(6)**	**42.9%** **(3)**	24.5% (25)
Rarely	5.8% (4)	0.0% (0)	14.3% (3)	6.7% (1)	14.3% (1)	7.8% (8)
Never	4.3% (3)	10.0% (1)	0.0% (0)	0.0% (0)	14.3% (1)	4.9% (5)
answered question	69	10	21	15	7	**102**
					skipped question	**0**



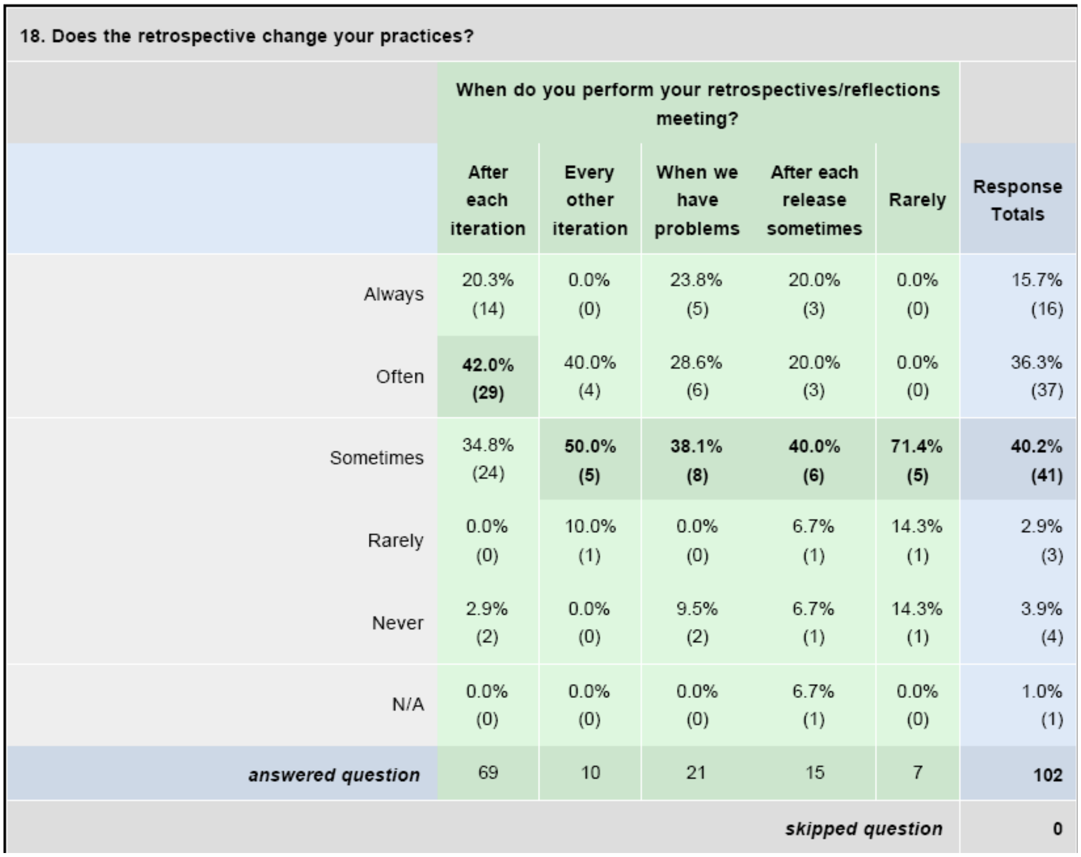
11. How do you rate the code quality?

	When do you perform your retrospectives/reflections meeting?					
	After each iteration	Every other iteration	When we have problems	After each release sometimes	Rarely	Response Totals
Very high	17.4% (12)	0.0% (0)	9.5% (2)	13.3% (2)	0.0% (0)	12.7% (13)
high	**56.5%** **(39)**	**50.0%** **(5)**	**57.1%** **(12)**	**53.3%** **(8)**	**57.1%** **(4)**	**54.9%** **(56)**
Average	20.3% (14)	**50.0%** **(5)**	28.6% (6)	26.7% (4)	28.6% (2)	26.5% (27)
Low	5.8% (4)	0.0% (0)	4.8% (1)	6.7% (1)	14.3% (1)	5.9% (6)
Very low	0.0% (0)	0.0% (0)	0.0% (0)	0.0% (0)	0.0% (0)	0.0% (0)
answered question	69	10	21	15	7	**102**
					skipped question	**0**



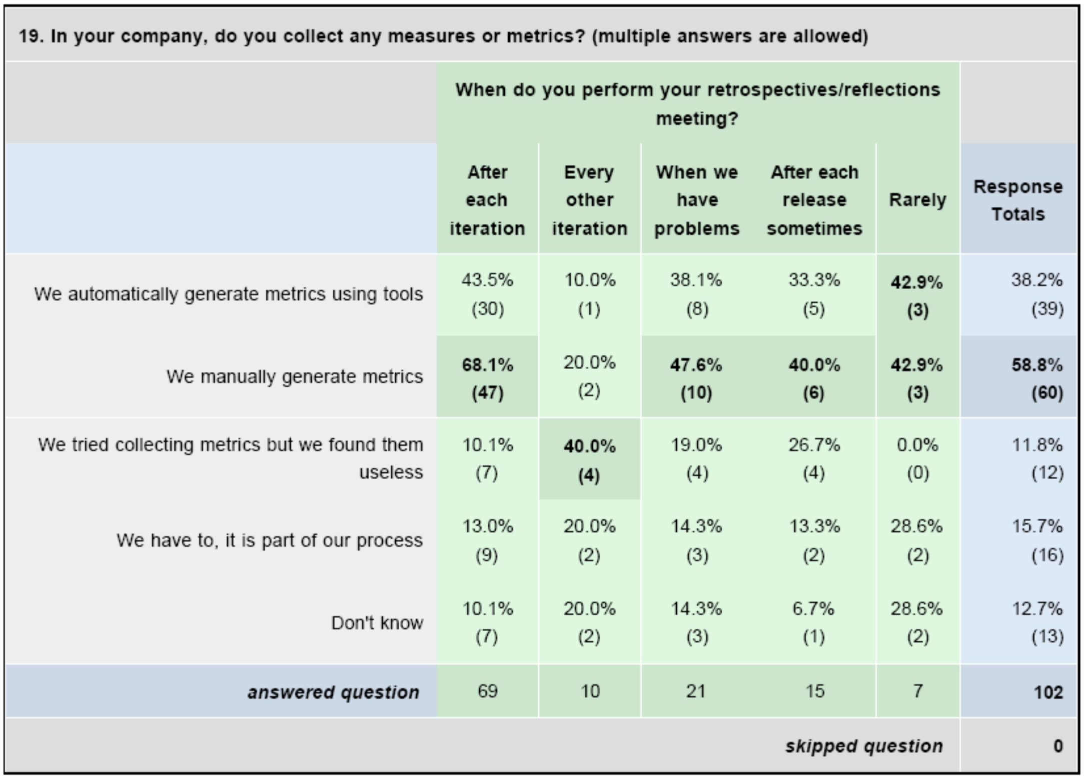
12. Was the project successful?

	When do you perform your retrospectives/reflections meeting?					
	After each iteration	Every other iteration	When we have problems	After each release sometimes	Rarely	Response Totals
Definitely	**56.5%** **(39)**	20.0% (2)	**57.1%** **(12)**	33.3% (5)	**57.1%** **(4)**	**49.0%** **(50)**
Somewhat	21.7% (15)	**50.0%** **(5)**	23.8% (5)	**40.0%** **(6)**	14.3% (1)	27.5% (28)
Partially	14.5% (10)	10.0% (1)	9.5% (2)	13.3% (2)	28.6% (2)	14.7% (15)
Clearly failed	1.4% (1)	0.0% (0)	4.8% (1)	0.0% (0)	0.0% (0)	2.0% (2)
Too early to say	5.8% (4)	20.0% (2)	4.8% (1)	13.3% (2)	0.0% (0)	6.9% (7)
answered question	69	10	21	15	7	**102**
					skipped question	**0**



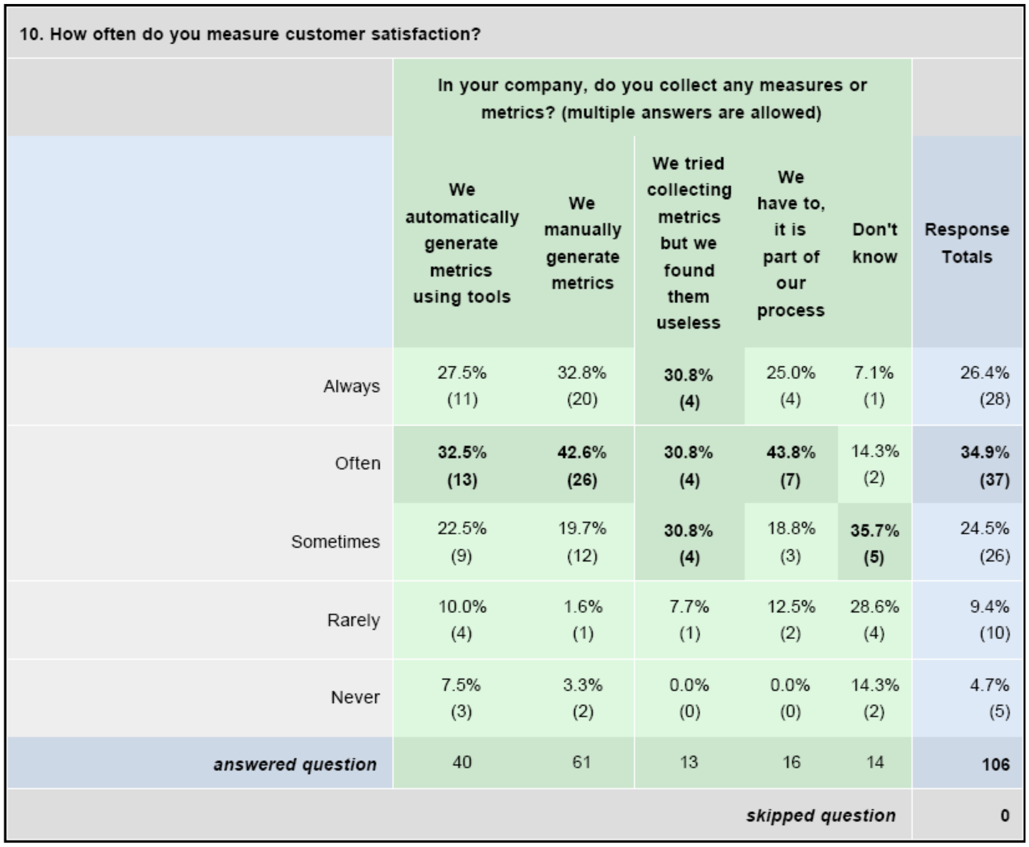
18. Does the retrospective change your practices?

	When do you perform your retrospectives/reflections meeting?					
	After each iteration	Every other iteration	When we have problems	After each release sometimes	Rarely	Response Totals
Always	20.3% (14)	0.0% (0)	23.8% (5)	20.0% (3)	0.0% (0)	15.7% (16)
Often	**42.0%** **(29)**	40.0% (4)	28.6% (6)	20.0% (3)	0.0% (0)	36.3% (37)
Sometimes	34.8% (24)	**50.0%** **(5)**	**38.1%** **(8)**	**40.0%** **(6)**	**71.4%** **(5)**	**40.2%** **(41)**
Rarely	0.0% (0)	10.0% (1)	0.0% (0)	6.7% (1)	14.3% (1)	2.9% (3)
Never	2.9% (2)	0.0% (0)	9.5% (2)	6.7% (1)	14.3% (1)	3.9% (4)
N/A	0.0% (0)	0.0% (0)	0.0% (0)	6.7% (1)	0.0% (0)	1.0% (1)
answered question	69	10	21	15	7	**102**
					skipped question	**0**



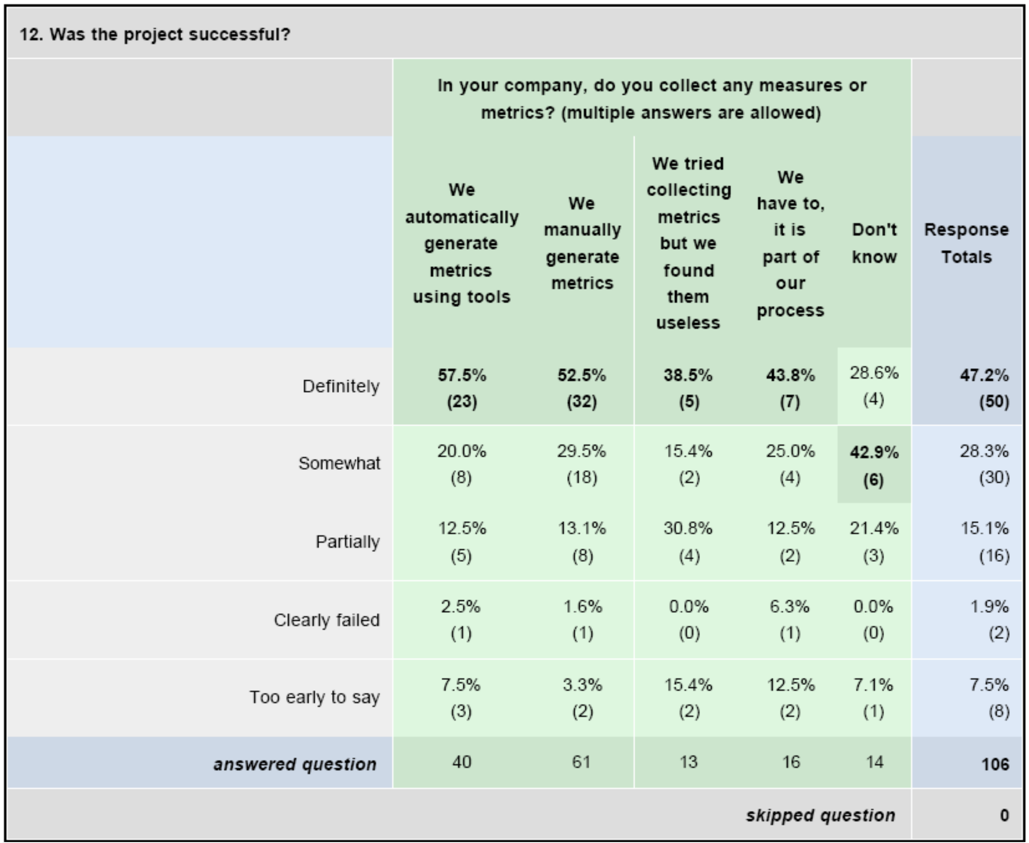
19. In your company, do you collect any measures or metrics? (multiple answers are allowed)

	When do you perform your retrospectives/reflections meeting?					
	After each iteration	Every other iteration	When we have problems	After each release sometimes	Rarely	Response Totals
We automatically generate metrics using tools	43.5% (30)	10.0% (1)	38.1% (8)	33.3% (5)	**42.9%** **(3)**	38.2% (39)
We manually generate metrics	**68.1%** **(47)**	20.0% (2)	**47.6%** **(10)**	**40.0%** **(6)**	**42.9%** **(3)**	**58.8%** **(60)**
We tried collecting metrics but we found them useless	10.1% (7)	**40.0%** **(4)**	19.0% (4)	26.7% (4)	0.0% (0)	11.8% (12)
We have to, it is part of our process	13.0% (9)	20.0% (2)	14.3% (3)	13.3% (2)	28.6% (2)	15.7% (16)
Don't know	10.1% (7)	20.0% (2)	14.3% (3)	6.7% (1)	28.6% (2)	12.7% (13)
answered question	69	10	21	15	7	**102**
					skipped question	**0**



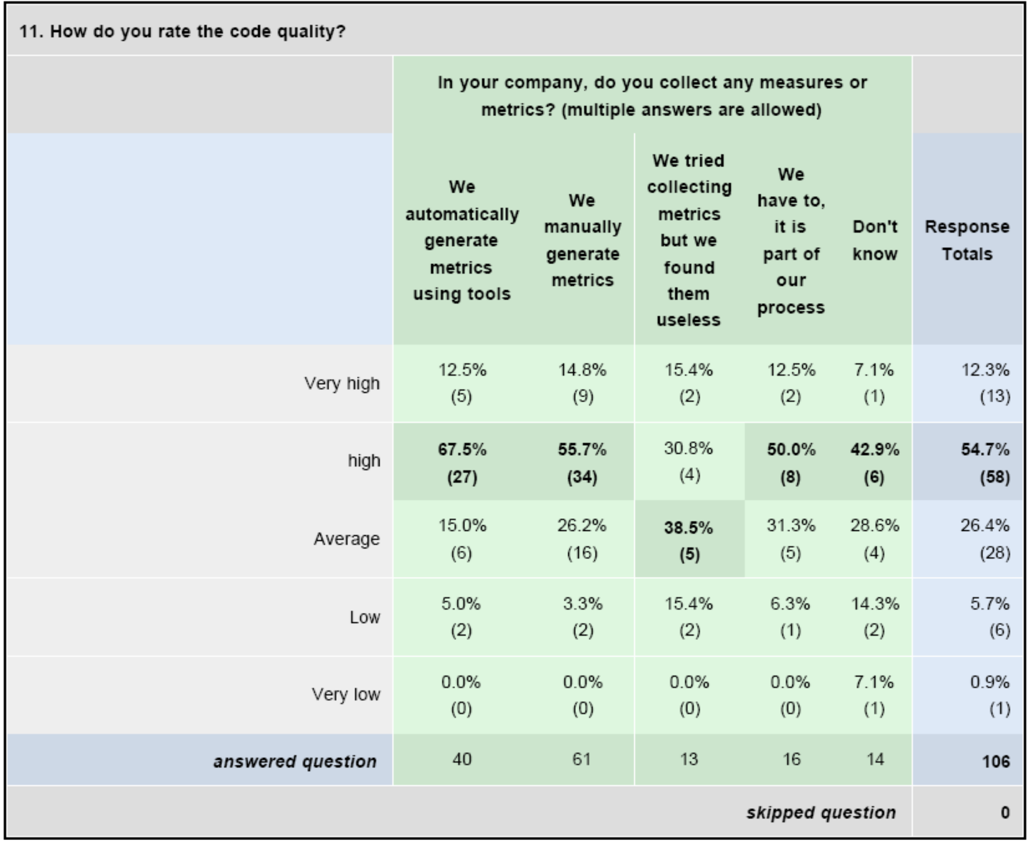
10. How often do you measure customer satisfaction?

	In your company, do you collect any measures or metrics? (multiple answers are allowed)					
	We automatically generate metrics using tools	We manually generate metrics	We tried collecting metrics but we found them useless	We have to, it is part of our process	Don't know	Response Totals
Always	27.5% (11)	32.8% (20)	**30.8%** **(4)**	25.0% (4)	7.1% (1)	26.4% (28)
Often	**32.5%** **(13)**	**42.6%** **(26)**	**30.8%** **(4)**	**43.8%** **(7)**	14.3% (2)	**34.9%** **(37)**
Sometimes	22.5% (9)	19.7% (12)	**30.8%** **(4)**	18.8% (3)	**35.7%** **(5)**	24.5% (26)
Rarely	10.0% (4)	1.6% (1)	7.7% (1)	12.5% (2)	28.6% (4)	9.4% (10)
Never	7.5% (3)	3.3% (2)	0.0% (0)	0.0% (0)	14.3% (2)	4.7% (5)
answered question	40	61	13	16	14	**106**
					skipped question	**0**



12. Was the project successful?

	In your company, do you collect any measures or metrics? (multiple answers are allowed)					
	We automatically generate metrics using tools	We manually generate metrics	We tried collecting metrics but we found them useless	We have to, it is part of our process	Don't know	Response Totals
Definitely	**57.5% (23)**	**52.5% (32)**	**38.5% (5)**	**43.8% (7)**	28.6% (4)	**47.2% (50)**
Somewhat	20.0% (8)	29.5% (18)	15.4% (2)	25.0% (4)	**42.9% (6)**	28.3% (30)
Partially	12.5% (5)	13.1% (8)	30.8% (4)	12.5% (2)	21.4% (3)	15.1% (16)
Clearly failed	2.5% (1)	1.6% (1)	0.0% (0)	6.3% (1)	0.0% (0)	1.9% (2)
Too early to say	7.5% (3)	3.3% (2)	15.4% (2)	12.5% (2)	7.1% (1)	7.5% (8)
answered question	40	61	13	16	14	**106**
					skipped question	**0**



11. How do you rate the code quality?

	In your company, do you collect any measures or metrics? (multiple answers are allowed)					
	We automatically generate metrics using tools	We manually generate metrics	We tried collecting metrics but we found them useless	We have to, it is part of our process	Don't know	Response Totals
Very high	12.5% (5)	14.8% (9)	15.4% (2)	12.5% (2)	7.1% (1)	12.3% (13)
high	**67.5% (27)**	**55.7% (34)**	30.8% (4)	**50.0% (8)**	**42.9% (6)**	**54.7% (58)**
Average	15.0% (6)	26.2% (16)	**38.5% (5)**	31.3% (5)	28.6% (4)	26.4% (28)
Low	5.0% (2)	3.3% (2)	15.4% (2)	6.3% (1)	14.3% (2)	5.7% (6)
Very low	0.0% (0)	0.0% (0)	0.0% (0)	0.0% (0)	7.1% (1)	0.9% (1)
answered question	40	61	13	16	14	**106**
					skipped question	**0**