Honours Final Project Report

"Investigating the use of Agile Development in a Large Scale Software Development"

By

Penelope Pitstop

BHCG5_1

(20030xxxx)

Project Supervisor: Dr. Richard Foley.
Second Marker: Mr. Edwin Gray.

Except where explicitly stated, all work in this document is my own.

Signed: ______________________________ Date: ______________

Abstract

Despite the introduction of different software process models and associated development methods, significant problems still remain within the software engineering industry. These include late system delivery, projects being over budget or not to specification and poor software quality, all leading to significant numbers of project failures. The most recent approach to combat this has been the development of methodologies collectively termed Agile Methods which promise to address the limitations of traditional methods through the use of a lightweight methodology and specific practices promoting adaptability to change and improved software quality.

This project focuses on the most commonly used agile method, extreme programming (XP) which utilises 12 principles for software development. Advocates of XP indicate that all practices must be implemented to obtain the related benefits from the method. This project investigates if all benefits could be achieved from implementing only a subset of principles.

A case study of a large organisation with both traditional and XP based development teams is undertaken, where the XP team does not implement the full set of principles. The study utilises metrics based analysis within its approach as well as interview and questionnaire based qualitative methods.

The results show that all benefits associated with a full implementation of XP are still experienced through the subset of principles. These also provide specific evidence of substantial increases in indicators of code quality and a reduction in project risk. These findings further suggest that organisations may benefit from a phased approach to the introduction of agile techniques when considering changes aimed at improving their software development process.

Acknowledgements

I would like to express my thanks to the supervisor of this project, Dr. Richard Foley, for his support and guidance in the completion of the project.

I would also like to express thanks to Company X for the help and assistance they have provided throughout the case study, especially Stuart and Doug.

Finally, I would like to thank Douglas for his advice and support throughout the investigation of this project.

Contents

List of Tables

1. Introduction

This section will introduce the project area of software engineering. The section also discusses the main problems within the software engineering industry and the software development processes in place to address those problems.

This section will also introduce the nature of the project including the research question, objectives and hypothesis which will be used to investigate the project. A summary of the primary research method of a case study will also be introduced.

Finally, this section will provide an overview to the main content of this document.

1.1 Background

Software engineering is defined by Sommerville (2004) as: *"an engineering discipline whose focus is the cost effective development of high-quality software systems"*. The author explains that this includes the selection and application of an appropriate development method to successfully create software.

A report by Krishnan et al (1999) has shown that in the last 30 years, the process of developing and maintaining software has become a major concern for the software engineering industry. The authors further detail that this crisis has emerged from reports of projects not being delivered on time, on budget or with a satisfactory product quality. This is supported in a report by SRI International which has found that less than 1% of projects are completed on time, on budget and to the original project specification (Dewey, 1988 as cited in Krishnan et al, 1999). A further report (Standish Group International, 1994 as cited in Glass, 2006) states project failure rates of 70% or more and claims software projects are always over budget, behind schedule or unreliable. More recently, a study by Sandmeier & Gassmann (2006) has found that almost two thirds of new products fail after launch due to rapidly changing customer requirements. These reports all collectively suggest that the problems common within the software engineering industry have not been properly addressed as these problems have not improved over time.

Sommerville (1996) suggests the main approach to tackling the problems in the industry is by adopting the use of a software development model. The use of such models can increase the efficiency and effectiveness of developing a software system (Balsamo et al, 2004; Guntamukkala et al, 2006) and therefore lead to the production of a successful project as Corbin reports (1991). It is also discussed by Krishnan et al (1999) that the use of a software process model can improve the productivity of the project team and increase product quality, thus tackling one of the main problems in the software engineering industry of projects being of poor quality.

Existing Approaches

Sommerville (1996) explains that software development models include the activities and processes needed to create a software system. There are many existing approaches categorised by Guntamukkala et al (2006) as heavyweight, middleweight and lightweight models. The authors explain:

- Heavyweight models refer to the initial software process models, introduced in the 1960s-1970s, including *"code and fix"* and the *"waterfall model"*.
- Middleweight models are more flexible, such as *"rapid prototyping"*, the *"V model"* and the *"spiral model"* which were introduced in the 1980s to overcome the problems experienced by the waterfall model.
- Lightweight models refer to a group of more recent models, termed Agile Methods - such as *"extreme programming"* and *"scrum"* which are flexible. Graham (2004) explains that agile methods were developed to reduce the high project failure rates common with traditional plan-driven approaches.

Why Agile?

Willoughby (2005) explains that agile methods are a group of software development models which produce *"high-quality, functioning software at a fraction of the time and cost of traditional methods"*. Agile methods originated in the late 1990s and focus on flexibility (Guntamukkala et al, 2006) and adaptability to change (Copeland, 2002; Lindstrom & Jefferies, 2004; Maurer & Martel, 2002a; Turk et al, 2005). Highsmith and Cockburn (2001) explain that agile methods have been proposed as a solution to overcome the difficulties experienced when developing software. As a report by Willoughby (2005) suggests, they are significantly more beneficial to implement than a traditional model. For this reason 10% of corporate IT organisations were utilising agile methods in 2002, with a further 25% investigating their use as reported by Sliwa (2002). This study also predicted that in the following 18 months more than two thirds of all corporate IT organisations would use some form of agile method.

Common characteristics of agile methods include breaking software into small, manageable chunks (Willoughby, 2005) and utilising an intensively iterative approach to develop software (Graham, 2004). Graham (2004) further explains the agile approach allows the customer to see small, functioning pieces of software much earlier in the development of the system. Agile methods cover many software process models including lean, scrum and crystal (Cohn & Ford, 2003; Guntamukkala et al, 2006). Maurer and Martel (2002a) explain that the most commonly implemented agile method is extreme programming (XP) and so this is the agile method which this project will focus on. XP is the most widely used agile method as it is not only the most mature (Nord & Tomayko, 2006) but also the most commonly known agile method (Turk et al, 2005).

There have been several significant benefits suggested through implementing XP as a development methodology. Ambler (2000a) explains that using XP allows an organisation to get the best value from a project as it focuses on what is most important to the project's development on a daily basis. A further benefit suggested by Maurer and Martel (2002a) is the increase in customer satisfaction due to the release of small, working parts of the application every few weeks. Moreover, Ambler (2000a) and Turk et al (2005) explain that XP is beneficial as it is flexible and allows a project to easily change direction when necessary without any difficulty.

Use of XP

XP is based on 12 principles as defined by Beck (2000): the planning game, small releases, metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40-hour week, on-site customer and coding standards. Turk et al (2005) explain that XP is renowned for its adaptability to change, however a study by Ambler (2000a) reports that XP is widely recognised as an *"all or nothing software process"*, as these reports explain that all 12 principles must be followed fully to successfully implement a project in XP. On the other hand, XP has also been recognised by Copeland (2001) and English (2002a) as an agile method which should be adapted to suit the development needs of an organisation. This adaptation would involve implementing a subset of principles which are most applicable to the software development.

Since conflicting viewpoints exist over which implementation of XP provides the most lucrative benefits, this project will investigate the issue through a case study of an organisation which currently implements a subset of XP principles. The study will therefore gain an insight into the benefits which the organisation receives from its implementation of XP and the extent to which these benefits are achieved.

1.2 Project Aims and Methodology

1.2.1 Project Method

To investigate the research area a case study will be used to measure the success of implementing a subset of XP principles. The case study will be focused on a real life, typical large scale software development organisation which shall be known throughout this report as Company X for the purposes of anonymity.

Company X develops large scale information based systems, based on a 24 hour emergency service. The development of their software has taken a traditional waterfall approach; however, more recently the company has adopted the use of agile methods within 2 core development teams. The agile approach implemented by Company X involves utilising a subset of the 12 XP principles: the planning game, small releases, simple design, testing, refactoring, collective ownership, continuous integration, 40-hour week and coding standards. As only a subset of 9 XP principles are implemented, it will be beneficial to uncover the success which Company X has encountered when using this adapted version of XP in comparison to the benefits reportedly gained from implementing the full methodology.

In order to measure the benefits obtained by Company X metric calculations will be performed on code listings, interviews will be performed with developers and customer representatives, and questionnaires will be administered to developers of two software systems during the case study. In order to gain a before and after view of the effect of implementing a subset of XP principles on Company X, two projects will be investigated including a traditional project (pre-agile) and a project utilising a subset of XP principles (post agile).

1.2.2 Initial Overview

A report by Krishnan et al (1999) has highlighted that there have been many problems in the software engineering industry regarding projects being delivered late, over budget or without the required functionality. Willoughby (2005) explains that agile methods have been introduced to tackle these problems. The most popular agile method is XP (Maurer & Martel, 2002a), however confliction exists as to how XP should be implemented as some studies report that all 12 principles must be implemented for success (Ambler, 2000a; Turk et al, 2005) as opposed to studies which suggest a tailored implementation should be adopted for optimum results (Cao et al, 2004; Copeland, 2001; English, 2002a; Fitzgerald et al, 2006; Layman et al, 2006). Therefore, a study on the most beneficial implementation of XP would prove valuable to the software engineering industry. By investigating the implementation of a subset of XP principles, this project will provide results on the success or otherwise of this development approach, thus drawing a conclusion as to the most effective implementation of XP. For these reasons the research question which the project will investigate is:

Does implementing a subset of the 12 extreme programming principles provide Company X with the substantial benefits associated with the full implementation of the XP methodology?

1.2.3 Objectives

In order to investigate the research question the following objectives will be met within a literature review:

1. **Agile methods:** an investigation of agile methods will provide an overview of the methodology and the benefits it presents to an organisation. This review will introduce the development method which is under investigation and will detail the initial benefits presented to an organisation when implementing an agile method.
2. **Typical organisation:** an investigation of the characteristics of an organisation that would benefit from utilising agile methods will allow Company X to be assessed for its suitability of implementing an agile method. This discussion will ensure Company X is a suitable candidate for the case study and will validate that the company will benefit from utilising agile methods.
3. **Assessment of extreme programming principles**: an assessment of the XP principles will include an explanation of XP, the 12 principles and the benefits presented to an organisation from implementing each of these principles. This objective will provide an initial understanding of the development process undertaken by Company X. Furthermore, from this discussion the potential benefits obtained through implementing XP will be identified for analysis within the case study.
4. **Use of metrics:** identifying suitable metrics which can be used to measure the benefits of XP and the implementation of these metrics will be undertaken. The results of this objective will be utilised within the case study to investigate the benefits which Company X have achieved from implementing a subset of the XP principles.

The following objectives will be met within the methods section of this report and will be exercised during the case study:

1. **Case study techniques:** suitable case study techniques which will be required to investigate the software development process currently utilised in Company X will be identified. Case study techniques are required as documentary analysis will be necessary to uncover the precise implementation followed by Company X. Interviews will also be conducted with system stakeholders to expose the impact of the XP implementation on the stakeholders and questionnaires will be administered to developers as a method to gather data in order to calculate project risk.
2. **Data analysis techniques:** data analysis techniques will be investigated to understand which can be utilised to analyse the results generated from the case study. Data analysis techniques will be required as these will be performed on the data gathered during the case study to evaluate the results in relation to the research question and hypothesis.

Based on the initial study of literature detailing the background of the project documented in Section 1.1, the aim of the project is to evaluate the hypothesis that:

An organisation will still obtain tangible benefits in software development productivity through the implementation of a subset of extreme programming principles.

As discussed previously, in order to investigate the hypothesis, a case study will be performed on a typical large scale software development organisation to uncover if significant benefits associated with the full implementation of XP can still be obtained through the implementation of a subset of XP principles.

1.3 Report Structure

This report will follow the structure detailed below.

1.3.1 Literature Review

The literature review, documented in Section 2, provides an understanding of agile methods, extreme programming and typical organisations which may benefit from implementing these development methods. This section also defines specific benefits which are achieved by implementing each XP principle. These benefits are particularly significant within the project as each individual benefit is investigated within the case study. The literature review then documents suitable metric calculations which could be performed during the case study to gain measurable results of the occurrence of the identified benefits.

1.3.2 Methods

Section 3 details the research methods used to investigate the occurrence of each benefit associated with XP. The primary research method of a case study is justified and its logistics explained in depth. This section also details the precise nature of the research conducted and the subjects involved in this process, including metric based quantitative research and documentary analysis, interview and questionnaire based qualitative research. Section 3 also discusses the process of data analysis which was undertaken on the data gathered from the case study in order to provide an evaluation of the research.

1.3.3 Presentation of Results

The results of the research conducted are presented within Section 4. This section details the results of the case study in the order of performance of research, thus metric based quantitative research followed by interview and questionnaire based qualitative research.
This section discusses and evaluates the results presented in relation to the benefits being experienced or not by Company X and also in relation to the work of others within this field as highlighted in the literature review.

1.3.4 Conclusions and Discussion

Finally, Section 5 provides a review of the aim of the project before drawing together the conclusions which have arisen from the results presented in Section 4. These conclusions are again analysed in relation to the work of others in this area as well as the overall research question and hypothesis of the project. This section also explains how Company X has obtained the benefits associated with the principles which the organisation has not implemented, before discussing limitations which have been placed on the project. Finally, this section provides possible further areas of research which have arisen from the evaluation of the results of the project before concluding the research.

2. Literature Review

This section will fulfil the objectives of the literature review as detailed in Section 1.2.3. This includes an introduction to agile methods, extreme programming and typical organisations that may benefit from implementing these methodologies. This section will also define specific benefits which can be achieved through the implementation of each XP principle. Finally, suitable metric calculations which could be performed during the case study will be discussed.

2.1 Overview of Agile

Agile methods are a collection of lightweight software process models (Alshayeb & Li, 2006; English, 2002b; Zhang, 2004). Highsmith & Cockburn (2001) explain that agile methods were developed to overcome the problems faced when implementing a traditional model for software development. This is reinforced by Baskerville et al (2006) who explain that agile methods are less formal than traditional methods. Common characteristics of agile methods as reported by Cao et al (2004) include developing iteratively, providing working pieces of functionality often and working in close consultation with the customer, thus showing that agile methods rely heavily on the input of the developer and customer. It is reported by Cohen (2005) that the common characteristics of agile methods make them effective in producing *"smaller, faster, cheaper"* software, thus suggesting agile methods are useful in combating the problems experienced by the software engineering industry which were discussed in the introduction.

Glass (2001) describes agile methods as *"human focused"* due to working practices such as a 40 hour working week. This human focused aspect of agile methods is further reinforced by Guntamukkala et al (2006), who explain that this development method requires a high level of involvement with the customer. Graham (2004) supports this by explaining customers can see a working system earlier and more often in development due to principles such as small releases. These human characteristics described provide agile projects with high success rates and satisfied customers as Lindstrom & Jeffries (2004) report. Agile methods also increase the satisfaction of developers as a study by Maurer & Melnik (2006) found twice as many agile developers were more satisfied with their job than non-agile developers. It has also been reported that agile methods help boost employee morale and allow developers to enjoy their job (*"Extreme programming: The Zero G experience"*, 2003).

Productivity has also been reportedly increased through the use of agile methods as Cohn and Ford (2003) discuss. A more recent report by Layman et al (2006) shows an improvement of up to 50%. These reports of increases in productivity would suggest that agile methods could address the problem in the software engineering industry of projects not being delivered on time.

A report by Maurer and Martel (2002a) highlights that agile projects are aimed to increase the responsiveness of an organisation and decrease the overheads associated with software development. This is accomplished by several factors including frequent iterations (*"Extreme programming: The Zero G experience"*, 2003; Sliwa, 2002), an accelerated life cycle (Cohn

& Ford, 2003) and by responding to changing requirements (Copeland, 2002). Ambler (2000a) reports that by utilising agile methods an organisation will receive the utmost value from their project.

Within this study, as Company X has only recently adopted the use of agile methods it will be beneficial to compare the results gained in both its agile and traditional projects. This will investigate if Company X has encountered the benefits discussed thus far and which lifecycle has provided the organisation with the most lucrative results.

2.2 Characteristics of a Typical Organisation

From the advantages discussed so far it could be envisaged that agile methods would be utilised across-the-board; however this is not the case as it is reported by Ambler (2001) that agile methods are only effective within the correct environment. Copeland (2001) reinforces this by explaining that agile methods are not suited to every software project.

Agile methods were designed for small development teams (Ambler, 2000a; Ambler, 2002; Williams et al, 2004). Cao and Ramesh (2007) define a small development team as typically less than 10 people. Small teams are reported by Cohn & Ford (2003) and Sliwa (2002) as being beneficial as they provide high levels of communication. It is suggested by Talby et al (2006) that the development team should operate a *"same room policy"*, a point confirmed by Williams et al (2004), who explain that there should be an open space working environment. By enforcing these policies, developers can work closely together and thus further increase face-to-face communication (Maurer & Martel, 2002a). These characteristics are reinforced by Williams & Kessler (2000), who report that 96% of programmers find the office layout is crucial to the success of an agile project.

Guntamukkala et al (2006) describe agile methods as a flexible process. This is confirmed by several reports (Ambler, 2000a; Ambler, 2002; Cohn & Ford, 2003; Highsmith & Cockburn, 2001; Sliwa, 2002) which state that optimum results are produced in development environments which are constantly changing or alternatively, as Ambler (2002b) explains, where requirements are ambiguous. These environments are favourable for a typical organisation as agile methods allow the project to adapt to change (Guntamukkala et al, 2006). As one of the main causes of project failure in software engineering is the inability to change requirements (Sandmeier & Gassmann, 2006), the literature reviewed suggests that agile methods address this problem as they work optimally in these conditions, thus signifying a possible reduction in project failure due to changing requirements.

With respect to project size, it is reported by Cao et al (2004), that agile methods can be utilised within large scale projects. English (2002b) also explains that they can be implemented in a variety of programming languages.

Initial research suggests that Company X meets the characteristics of a typical agile organisation. Therefore, it will be beneficial to the software engineering industry to study the benefits which the company has received as this will provide an insight into the potential benefits which other 'typical' organisations may achieve when adopting an agile method.

2.3 Assessment of Extreme Programming

Maurer & Martel explain that XP is the most commonly used agile method. In a further report (Maurer & Martel, 2002b) the authors suggest that XP is most commonly used as it is the most practical agile method. It is also possible that XP is widely utilised as the authors report that XP produces large productivity gains.

The review of several reports suggests that by implementing XP common problems within the software engineering industry can be addressed. It is reported by Ambler (2000a) that implementing XP leads to successfully delivered software projects, which suggests XP addresses the problem of traditional projects not meeting requirements. Furthermore, Copeland (2001) explains that XP can provide early product delivery, thus addressing the problem of traditional software projects not being delivered on time.

2.3.1 Extreme Programming Principles

As introduced previously, XP is based around the implementation of 12 principles (English, 2002b; Levy, 2003). The principles as defined by founder Kent Beck (2000) are detailed below. The benefits associated with each principle are also discussed.

1. The Planning Game - *"Quickly determine the scope of the next release by combining priorities and technical estimates. As reality overtakes the plan, update the plan"* (Beck, 2000).

As Copeland (2002) explains, this principle allows projects to have less time devoted to planning as this task is undertaken at the start of each iteration. From implementing the planning game Turk et al (2005) explain that the competitiveness of an organisation can be enhanced due to the flexibility of the project and its ability to adapt to change, thus confirming that planning takes place throughout the project. Wood & Kleb (2003) also report that by implementing this principle it is easier to incorporate further principles such as continuous integration as planning helps define *"small chunks"*. This suggests these *"small chunks"* can be used as software modules during continuous integration. Finally, English (2002b) explains that by implementing the planning game the risk of a project can be reduced as the project will focus on developing the most complex feature of the requirements first.

2. Small Releases -*"Put a simple system into production quickly, then release new versions on a very short cycle"* (Beck, 2000).

This principle allows complete reviews of the system every few weeks (*"Extreme programming: The Zero G experience"*, 2003). English (2002b) explains that these reviews take place every 2-3 weeks and allow the customer to *"see and touch"* the working system often. Highsmith & Cockburn (2001) support this as they report that regular reviews with the customer provide rapid feedback to developers, thus suggesting that the developers are aware of the customer's judgement of the system on a regular basis. Furthermore, Maurer & Martel (2002a) explain that by releasing small parts of the system regularly, the customer is provided with working functionality frequently and so the risk of a project is reduced, possibly as the customer is familiar with the application. Another benefit of implementing small releases is suggested by Alshayeb & Li (2006) who explain that a project can respond to changes in requirements which arise from the customer reviewing functional parts of the system. This is reinforced further by Graham (2004) who discusses that small releases ensure the requirements of a system are being met. Again, these reports suggest that agile methods can produce systems which meet the requirements of the user and thus potentially minimise the risk and opportunity for project failure.

Finally, it is reported by Maurer & Martel (2002a) that a customer can monitor the development of a project through small releases and in doing so this increases the satisfaction of the customer as they are aware of the progress of the project.

3. Metaphor -*"Guide all development with a simple shared story of how the whole system works"* (Beck, 2000).

Maurer & Martel (2002a) explain that the metaphor clarifies what the project team is working towards by giving a reliable view of the system. This is reinforced in a report by Lindstrom & Jeffries (2004) who discuss that a metaphor ensures everyone on the project team understands what the system will do and has an overview of how the system will operate and where parts of the system functionality can be located within it. The authors also explain that each team member can relate to the system as they are aware of their individual part in the project's development. Glass (2001) describes this principle as beneficial as it allows the project team to describe the system in everyday language. Wood & Kleb (2003) are also of this opinion as they state a metaphor should not be written in technical phraseology. In a report by Williams et al (2004), the authors explain that developing a metaphor of the system improves team communication and also guarantees understanding is spread throughout the team, thus reducing the risk of the project.

Although the metaphor provides advantages for its use, it is reported by Rumpe & Schrder (2002) that many XP teams do not utilise it as they are unclear how it should be implemented successfully. This suggests that this principle may not be exercised should an organisation be implementing a subset of XP principles such as those described in reports by Copeland (2001) and English (2002a).

4. Simple Design – "*The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered"* (Beck, 2000).

Ambler (2001) states that simple design produces *"clean and clear code"*. This is supported by Highsmith & Cockburn (2001), who report that *"clean and clear code"* allows changes within a project to be integrated more easily and also suggests that the number of changes required within a project will be less. A further advantage of implementing simple design was reported by English (2002b), who explains that the complexity of developed code is reduced through the implementation of this principle. Alshayeb & Li (2006) confirm this as they report that the number of errors found in a project is reduced should the development utilise simple design. Therefore, this literature suggests that by implementing this principle the system developed will be more robust as code is not as technically complex and also features fewer errors.

5. Testing -*"Programmers continually write unit tests, which must run flawlessly for development to continue. Customers write tests demonstrating that features are finished"* (Beck, 2000).

This principle is used to present developers with feedback quickly (Ambler, 2000b; Rasmusson, 2003). Further advantages of the testing principle include an increase in software and project quality as detailed in two reports by Williams et al (2003; 2004). It is also reported by English (2002b) that testing ensures the code and system are validated earlier. This is supported by Highsmith & Cockburn (2001) as they discuss that testing leads to the earlier detection of errors within the project, possibly as the code and system are validated earlier. Talby et al (2006) also report a reduction in the number of defects and the time taken to fix those defects, thus suggesting that this reduces the possibility of system failure and therefore reducing project risk. Testing is also reported by Thomas (2005) as a means to promote software evolution.

6. Refactoring -*"Programmers restructure the system without changing its behaviour to remove duplication, improve communication, simplify, or add flexibility"* (Beck, 2000).

Refactoring provides many benefits to organisations that choose to implement the principle. Firstly, refactoring reduces the complexity of code as reported by Capiluppi et al (2007) and Mens & Tourwe (2004). It is possible the complexity of code is reduced as refactoring produces *"clean and clear code"* which is easy to understand as Ambler (2001) reports. Furthermore, as code is *"clean and clear"* this not only supports the findings of Marurer & Martel (2002a) who explain that the structure of code is improved but also supports the findings of Alshayeb & Li (2006) who indicate that code is easier to read. These characteristics reviewed suggest that the use of refactoring produces well designed code which was also reported in several studies (Lindstrom & Jeffries, 2004; Rainsberger, 2007; Turk et al, 2005). Moreover, as code is well designed and easy to read this suggests that future maintenance work on code would be easier to perform as found by Levy (2003) and Neill & Gill (2003). This further indicates that refactoring increases the flexibility and quality of a system as also reported by two papers (Levy, 2003; Mens & Tourwe, 2004), possibly as changes can be implemented more easily into code which is readable and maintainable. Therefore the literature reviewed suggests that refactoring simplifies code and makes code easier to use, which confirms the findings of Cao et al (2004) who explain that by implementing refactoring the developed system is more robust.

7. Pair Programming -*"All production code is written with two programmers at one machine"* (Beck, 2000).

Conrad (2000) explains that pair programming is a *"controversial aspect"* as two developers are doing the work of one person. On the other hand, several reports state that pair programming leads to high productivity gains (Dybå et al, 2007; Griffin, 2002; Turk et al, 2005; Wood & Kleb, 2003; Williams & Kessler, 2000; Williams et al, 2000). This is supported by the findings of Cao et al (2004) who explain that the same functionality can be implemented in 80% of the time when pairing. Although pair programming leads to a reduction in development time, several studies (Cao et al, 2004; Griffin, 2002; Holmström et al, 2006; McMahon, 2003; Turk et al, 2005; Williams et al, 2000; Williams et al, 2004) report the quality of the code produced in this reduced time frame is significantly higher than individually produced code. This is supported in three studies (Fitzgerald et al, 2006; Williams et al, 2000; Williams & Kessler, 2000) which report that pair developed code exhibits a reduced count of defects. Furthermore, it has been reported by McDowell et al (2006) and Wood & Kleb (2003) that pair developed code is easier to read.

Pair programming has also been reported beneficial to developers by Williams & Kessler (2000) as 96% of pair programmers enjoy their job more whilst pairing. The principle is also seen to improve the confidence of developers (McDowell et al, 2006; Williams et al, 2000), make employees more enthusiastic (Fitzgerald et al, 2006) and increase job satisfaction amongst those who practice it (Dybå et al, 2007; Maurer & Martel, 2002a; Williams & Kessler, 2000). Furthermore, pair programming also improves communication between team members as found by Biggs (2000) and Williams et al (2004). This is confirmed by several studies (Biggs 2000; Dybå et al, 2007; McDowell et al, 2006; Williams et al, 2004), which report that pair programming ensures knowledge of the system and the development methods used are spread throughout the team. As knowledge is spread throughout the team and the communication levels are increased this can reduce the training required for staff as each team member can now learn from another. This was reported in several studies (Cao et al, 2004, Lindstrom & Jeffries, 2004; Rasmusson, 2003; Williams et al, 2000). Williams et al (2004) also report that this team learning approach reduces the risk of the project and as Griffin (2002) reports it also reduces the overall project cost.

Although pair programming has many potential benefits, it is reported by Copeland (2001) as difficult to practice due to team size, location or an outsourcing of work. This is supported by Glass (2001) who reports that pair programming is often not fully utilised as it requires a certain type of programmer to be successful. These studies suggest that should an organisation be implementing only a subset of XP principles this principle would possibly not be implemented.

8. Collective Ownership - *"Anyone can change any code anywhere in the system at any time"* (Beck, 2000).

A study by Fitzgerald et al (2006) reports that collective ownership introduces flexibility to a project as every team member has access to each part of the system. By allowing any member of the team to change any code in the system, knowledge of the project is spread throughout the team as Layman et al (2006) report. It is also reported by Williams et al (2004) that collective ownership is a way of reducing the risk of a project, possibly as all team members are aware of the workings of the whole system. As collective ownership allows any team member access to any piece of code, the maintainability of the system is increased as reported by two studies (Fitzgerald et al, 2006; Lindstrom & Jeffries, 2004). This is possibly because any team member can make changes to any part of the system when required, thus increasing the opportunity for system maintenance. A further advantage of collective ownership is reported by Biggs (2001) and English (2002b) as both papers explain that collective ownership reduces the number of defects in a project, possibly as every team member is aware of the workings of each part of the system and is more aware of the consequences which their code may have on other areas of the system.

9. Continuous Integration - *"Integrate and build the system many times a day, every time a task is completed"* (Beck, 2000).

Continuous integration is found advantageous in a study by Maurer & Martel (2002a) as it ensures an executable version of the system is always available. This is supported by Ambler (2000b) as having a working version of the system provides an opportunity for rapid feedback to the development team. Lindstrom & Jeffries (2004) also report continuous integration as advantageous as it helps detect errors within the project. This is reinforced by a report from Biggs (2000) who explains that this principle reduces the likeliness of errors occurring when the system is released as the errors have been detected previously when the system is integrated. Moreover, studies by English (2002a; 2002b) report that by building the system many times a day the final integration process is simplified and thus also suggests that the final integration time would be reduced.

10. 40-Hour Week -"*Work no more than 40 hours a week as a rule. Never work overtime a second week in a row"* (Beck, 2000).

This principle is described by Glass (2001) as *"human focused"* as it involves no extra working hours for the developers. This is supported by Turk et al (2005) who state that this principle will *"stimulate and motivate"* developers, possibly as they are less likely over worked. Therefore, as reported by English (2002b) and Griffin (2002), this principle enhances productivity of developers and thus suggests that the 40-hour week could address the problem in software engineering of not delivering projects on time, as developers are more motivated to complete the project.

11. On-Site Customer -"*Include a real, live user on the team, available full-time to answer questions"* (Beck, 2000).

This principle is described by Glass (2001) as *"customer focused"*. This is supported by reports of an increase in customer satisfaction (Maurer & Martel, 2002a; Williams et al, 2004). Having a customer available on site is also beneficial to developers as it helps resolve any issues which may arise in development and helps to set priorities of tasks as English (2002b) reports. This principle can also lead to a reduction in feedback time as the customer is always available (Williams et al, 2004).

Although having a customer on-site is advantageous, Cao et al (2004) report that direct access to a customer is not always available and so this principle is not always exercised within XP teams. This suggests that should an organisation be implementing a subset version of XP that this principle may not be utilised.

12. Coding Standards -*"Programmers write all code in accordance with rules emphasising communication through the code"* (Beck, 2000).

English (2002b) explains that coding standards are common rules implemented by each team member to make code appear to have been written by one developer. It is reported (English, 2002b; Griffin, 2002) that by implementing coding standards the consistency of code is increased as all developers are using the same criteria. Levy (2003) also explains that the implementation of this principle increases the readability of code as the code is more consistent. Additionally, as code is easier to read this suggests that code would therefore be easier to maintain, as discussed by Erdogmus (2007). Levy (2003) further explains that this principle produces a higher quality of code, thus suggesting that by implementing coding standards the problem within the software engineering industry of delivering low quality projects would be addressed.

2.3.2 Benefits Summary

From the key points discussed under each of the 12 XP principles, the assessable benefits can be summarised in Table 1 below. The benefits highlighted in this table are the explicit benefits which will be investigated within the case study.

The manner in which the benefits are measured is important in order to achieve accurate and rigorous results, therefore qualitative and quantitative research will be performed.
Some of the benefits in Table 1 are subjective as these involve personal opinions of system stakeholders including developer and customer representatives, for example: an increase in customer satisfaction, an increase in job satisfaction and an increase in confidence. These subjective benefits will be measured qualitatively during the case study by interviewing developers and customer representatives. However, the remaining benefits are only measurable quantitatively as these involve calculations to be performed on data obtained from Company X. Examples of these benefits include: an increase in productivity, an increase in quality and an increase in code robustness. In order to measure these benefits, metrics will be performed on code listings obtained from Company X. These potential metric calculations will be documented in Section 2.4.

Table 1 - Benefits Gained from XP Principles

XP Principle	Associated Benefit
The Planning Game	Reduction in planning time Project more flexible/can adapt to change Reduces project risk
Small Releases	Reviews often Rapid feedback gained Reduces project risk Project can adapt to change/Flexibility Ensures requirements are being met Increased customer satisfaction
Metaphor	Provides an overall view of the system Provides an understanding of the project Provides a relation to the project Improves communication Allows knowledge to be spread throughout the team Reduces project risk
Simple Design	Clean and clear code/Readability Easier to implement changes Less change within a project Reduction in complexity Reduction in number of defects
Testing	Rapid feedback provided Quality increase Earlier defect detection Reduction in number of defects Reduction in time taken to fix defects Promotes software evolution/flexibility
Refactoring	Reduces complexity Clean and clear code produced/Readability Improves maintainability Increases quality Promotes system flexibility Improves robustness

Pair Programming	Increased productivity/reduced development time Increased quality Reduced number of defects Increased readability of code Increased developer confidence Increased developer job satisfaction Improves communication Allows knowledge to be spread throughout the team Developer training reduced Reduction in project risk Reduction of project cost
Collective Ownership	Increase project flexibility Allows knowledge to be spread throughout the team Reduction in project risk Improves maintainability Reduces number of defects
Continuous Integration	Rapid feedback provided Earlier defect detection Reduces number of defects Simplifies integration process
40 Hour Week	Stimulates and motivates developers Increases productivity
On-Site Customer	Increased customer satisfaction Easy resolution of issues and setting of priorities Rapid feedback provided
Coding Standards	Increases readability of code Increases maintainability of code Increases quality

2.3.3 Implementation of Principles

As discussed previously, conflicting viewpoints exist in how to implement XP most effectively. Many reports (Ambler, 2000a; Lindstrom & Jefferies, 2004; Maurer & Martel, 2002a; Turk et al, 2005) suggest that all 12 principles must be fully implemented to successfully develop a project in XP. Although XP has also been recognised in studies (Copeland, 2001; English, 2002a) as an agile method which should be tailored - such as implementing a subset of principles - to suit the organisation's needs. Moreover, previous case studies (Cao et al, 2004; Fitzgerald et al, 2006; Layman et al, 2006; Rumpe & Schrder, 2002) have proven that using a subset of principles has delivered high success rates.

Due to these conflicting viewpoints, it will be advantageous to conduct a case study on Company X to investigate the benefits which it may have encountered whilst implementing a subset of XP principles. The investigation will provide further evidence on the requirement to implement a subset of XP principles or otherwise, in order to gain a high level of success when implementing XP.

2.4 Metrics

The following section details proposed metric calculations which could be used to measure the reported benefits of XP which are measurable quantitatively as detailed in Table 1.

Complexity

As can been seen from Table 1, the simple design and refactoring principles can reduce the complexity of a system. Capiluppi et al (2007) explain that the complexity of a system or code can be calculated using McCabe's Cyclomatic Complexity metric, therefore this metric could be used to measure if a reduction in system complexity has been experienced in Company X through its implementation of XP.

The Cyclomatic Complexity metric is represented by McCabe (1976) as:

$$Complexity = (DecisionNodes) + 1$$

(Where 'DecisionNodes' includes the total number of constructs such as IF, WHILE, REPEAT in a program (Fenton, 1991)).

Rosenberg & Hyatt (1997) explain that the total complexity of a class should preferably be below 10, therefore the lowest complexity value in the pre and post agile results of Company X would signify the lowest system complexity and thus an overall reduction or increase in system complexity when using a subset of XP principles.

Cost

Table 1 shows pair programming can reduce the cost of a project. Boehm (2000) explains the cost of a project can be calculated using the COCOMO model. Thus, this metric could be utilised in the case study to calculate the cost of a traditional and agile project in Company X.

Nord & Tomayko (2006) explain that the model involves two metric calculations to produce the time taken for development (TDEV). This paper details the calculations as:

$$Effort(PM) = a(KLOC)^{\mathrm{b}}$$

$$TDEV = a(PM)^{\mathrm{b}}$$

(Where PM=person months, KLOC=thousands of lines of code and a and b are standard values dependent on the project type (Nord & Tomayko, 2006)).

Thus, should the post agile result provide a lower TDEV than the pre agile result this would indicate a reduction in the cost of the agile project.

Defect Density

As Table 1 shows, several principles lead to a reduced number of defects. These principles include: simple design, testing, pair programming, collective ownership and continuous integration. Conte et al (1986) discuss that in order to compare the defects in a software project meaningfully, such as calculating the reduction or increase in the number of defects, the defect density metric is most commonly utilised. Defect density is defined (Conte et al, 1986; Layman et al, 2006) as:

$$Density = \left(\frac{Defects}{KLOEC} \right)$$

(Where KLOEC is thousands of lines of executable code (Layman et al, 2006)).

Therefore, pre and post agile code could be measured and analysed to investigate if Company X has experienced a reduction in the number of defects within its agile project by utilising a subset of XP principles.

Defect Detection (Time)

Testing and continuous integration are principles which provide an earlier defect detection as shown in Table 1. Fenton (1991) explains that timing of errors can be measured as either the lifecycle stage of occurrence *("ordinal scale")* or as real time *("interval scale").* This would therefore be expressed either in real time or the stage of the occurrence within the projects lifecycle (i.e. pre-release, post-release). Thus, this metric could be utilised to measure the time taken to detect errors in pre and post agile code within Company X. The project with the shortest detection time would therefore indicate the development method with the earliest defect detection rate.

Error Fix (Time)

Table 1 shows that testing reduces the time taken to fix an error. As discussed previously, Fenton (1991) explains that the time taken to fix an error can be expressed in either real time, or the stage of occurrence within the project's lifecycle. This metric could be used within the case study to measure the time taken to fix an error pre and post agile in Company X. Should the post agile value be less than the pre agile value, this would show a reduction in the time taken to fix an error when using XP.

Maintainability

Refactoring, collective code ownership and coding standards all improve maintainability of code when implemented as shown in Table 1. Maintainability of a class can be determined by the values of coupling and cohesion which it exhibits (Erdogmus, 2007; Mens & Tourwe, 2004; Neill & Gill, 2003). Conte et al (1986) explain that coupling is the total number of interconnections between modules, while cohesion is the relationship of components within a module. The authors further explain that the software should exhibit levels of low coupling and high levels of cohesion.

Maintainability of software can also be calculated by the weighted methods per class (WMC) metric (Rosenberg & Hyatt, 1997). This paper explains that WMC is the sum of all complexities of methods, the result of which should be as low as possible.

These metrics could be utilised collectively within the case study to determine if the results of implementing a subset of XP principles provide an increase in maintainability over the traditional results. Values of lower coupling, higher cohesion and lower WMC would be required to achieve an increase in maintainability when using XP.

Productivity

As can be seen from Table 1, the productivity of an individual developer is increased by implementing pair programming and a 40 hour week. Productivity can be measured by calculating the size of the code developed over a period of time (Boehm, 2000; Maurer & Martel, 2002b; Williams et al, 2004). These papers explain that the following calculation would be performed:

$$productivity = \left(\frac{LinesOfCode}{Effort(time)} \right)$$

This metric could be utilised in the case study of Company X to investigate if the organisation has achieved higher productivity per developer when implementing a subset of XP principles or a traditional software development model. By analysing the pre and post agile results, the highest level of productivity would suggest if productivity had increased by implementing XP.

Project Flexibility

Table 1 describes that the flexibility of a project can be increased when utilising the planning game, small releases, testing, refactoring and collective ownership principles. Layman et al (2006) explain that the flexibility of a project can be measured by *"dynamism"*. Williams et al (2004) explain dynamism as *"the extent to which requirements change throughout a project*". This paper further explains that dynamism can be calculated by the number of user stories/requirements added or removed from a project. The higher the number of user stories/requirements added or removed, the more flexible the project is. Therefore, the pre and post agile results from Company X could be analysed to uncover the possibility of improving the organisation's flexibility by increasing the project's dynamism through implementing a subset of XP principles.

Project Risk

As specified in Table 1, planning game, small releases, metaphor, pair programming and collective ownership principles all reduce the risk of a project. Jiang et al (2002) suggest this can be measured by a combination of metrics, including *"complexity, user experience, user support and project team expertise"*. This paper also proposes a list of 21 statements which have been devised as factors of project risk that can be used to gain a measurement by ranking each of the statements with a value from 1 - 5 in accordance with agreement, where 5 is equal to a strong agreement with the statement. The paper explains that the higher the score of the statements, the higher the risk of the project. Therefore, to measure the risk of each respective project, these statements could be issued as a questionnaire to the developers of each project - the traditional and the XP project - to obtain a rank value for each statement based on the developers opinion. The scores for each completed questionnaire would then be accumulated on a per project basis, with the project with the greatest score being the project with the most risk.

Readability of Code/Clean and Clear Code

Simple design, refactoring, pair programming and coding standards are detailed in Table 1 as principles to increase the readability of code or to improve code consistency. Lindstrom & Jeffries (2004) discuss that readability can be measured by the values of coupling and cohesion found in code. The authors further explain that a level of low coupling and high cohesion would be most desirable. Readability of code can also be measured by the percentage of comments within a class as discussed by Rosenberg & Hyatt (1997). The authors explain that this would be calculated by:

$$\left(\frac{NumberOfComments}{LinesOfCode}\right) - BlankLines$$

These metric calculations could be collectively employed within the case study of Company X. Both pre and post agile code would be analysed to investigate if an increase in readability was encountered due to the use of a subset of XP principles, thus a measurement of lower coupling, higher cohesion and a higher percentage of comments within a class would be required.

Robustness

Table 1 describes refactoring as a principle which increases the robustness of software. Rosenberg & Hyatt (1997) suggest the robustness of a project can be measured by the number of children metric (NOC). This paper explains that NOC is a calculation of *"the number of immediate subclasses subordinate to a class in the hierarchy"*, where the smaller the number of children, the more robust the system. This metric could be utilised within the case study of Company X to investigate if the NOC has decreased when implementing XP in comparison to developing traditionally, thus making the system more robust.

Software Quality

Testing, refactoring, pair programming and coding standards are highlighted in Table 1 as principles which improve software quality. Software with high levels of quality has shown to have a reduced defect density (Cao et al, 2004; Zhang, 2004), thus this metric could potentially be performed during the case study to calculate the quality of the developed software in Company X. As discussed previously, defect density is defined (Conte et al, 1986; Layman et al, 2006) as:

$$Density = \left(\frac{Defects}{KLOEC}\right)$$

(Where KLOEC is thousands of lines of executable code (Layman et al, 2006)).

Therefore, pre and post agile code could be measured and analysed to uncover if Company X has experienced a reduction in defect density and thus produced a better quality system through their agile implementation.

2.5 Literature Conclusion

This section has presented the benefits of implementing agile methods, the characteristics of a typical organisation which could benefit from implementing an agile method, the benefits of each XP principle and has also discussed metric calculations which could be used to measure the proposed benefits of XP which are measurable quantitatively. Table 2 overleaf shows the benefits associated with each XP principle and the metrics which can be used to measure the quantitative benefits. Where a benefit is measurable qualitatively this has been stated in the table.

The research methods to be implemented to perform the measurements detailed in Table 2 will be discussed in depth in Section 3.

Table 2 - Benefits Gained from XP Principles and the Method of Measurement

XP Principle	Associated Benefit	Measurable
The Planning Game	Reduction in planning time	Qualitatively - Interview
	Project more flexible/can adapt to change	Quantitatively - Project dynamism
	Reduces project risk	Qualitatively - Questionnaire
Small Releases	Reviews often	Qualitatively - Interview
	Rapid feedback gained	Qualitatively - Interview
	Reduces project risk	Qualitatively - Questionnaire
	Project can adapt to change/Flexibility	Quantitatively - Project dynamism
	Ensures requirements are being met	Qualitatively - Interview
	Increased customer satisfaction	Qualitatively - Interview
Metaphor	Provides an overall view of the system	Qualitatively - Interview
	Provides an understanding of the project	Qualitatively - Interview
	Provides a relation to the project	Qualitatively - Interview
	Improves communication	Qualitatively - Interview
	Allows knowledge to be spread throughout the team	Qualitatively - Interview
	Reduces project risk	Qualitatively - Questionnaire
Simple Design	Clean and clear code/Readability	Quantitatively - Coupling, cohesion, percentage of comments
	Reduction in complexity	Quantitatively - Cyclomatic complexity
	Easier to implement changes	Qualitatively - Interview
	Reduction in number of defects	Quantitatively – Defect density
	Less change in project	Qualitatively - Interview
Testing	Rapid feedback provided	Qualitatively - Interview
	Quality increase	Quantitatively - Defect density
	Earlier defect detection	Quantitatively - Defect detection (Time)
	Reduction in number of defects	Quantitatively – Defect density
	Reduction in time taken to fix defects	Quantitatively - Error fix (Time)
	Project can adapt to change/Flexibility	Quantitatively - Project dynamism
Refactoring	Improves robustness	Quantitatively - Number of children (NOC)
	Clean and clear code produced/Readability	Quantitatively - Coupling, cohesion, percentage of comments
	Reduces complexity	Quantitatively - Cyclomatic complexity
	Improves maintainability	Quantitatively - Coupling, cohesion,WMC
	Increases quality	Quantitatively - Defect density
	Project can adapt to change/Flexibility	Quantitatively - Project dynamism
Pair Programming	Increased productivity/reduced development time	Quantitatively - Productivity
	Increased quality	Quantitatively - Defect density
	Reduced number of defects	Quantitatively – Defect density
	Clean and clear code produced/Readability	Quantitatively - Coupling, cohesion, percentage of comments
	Increased developer job satisfaction	Qualitatively - Interview
	Increased developer confidence	Qualitatively - Interview
	Improves communication	Qualitatively - Interview
	Allows knowledge to be spread throughout the team	Qualitatively - Interview
	Developer training reduced	Qualitatively - Interview
	Reduction in project risk	Qualitatively - Questionnaire
	Reduction of project cost	Quantitatively - COCOMO cost model
Collective Ownership	Project can adapt to change/Flexibility	Quantitatively - Project dynamism
	Allows knowledge to be spread throughout the team	Qualitatively - Interview
	Reduction in project risk	Qualitatively - Questionnaire
	Improves maintainability	Quantitatively - Coupling, cohesion, WMC
	Reduces number of defects	Quantitatively – Defect density
Continuous Integration	Rapid feedback provided	Qualitatively - Interview
	Simplifies integration process	Qualitatively - Interview
	Earlier defect detection	Quantitatively - Defect detection (Time)
	Reduces number of defects	Quantitatively – Defect density
40 Hour Week	Stimulates and motivates developers	Qualitatively - Interview
	Increases productivity	Quantitatively - Productivity
On-Site Customer	Rapid feedback provided	Qualitatively - Interview
	Easy resolution of issues and setting of priorities	Qualitatively - Interview
	Increased customer satisfaction	Qualitatively - Interview
Coding Standards	Clean and clear code produced/Readability	Quantitatively - Coupling, cohesion, percentage of comments
	Improves maintainability	Quantitatively - Coupling, cohesion, WMC
	Increases quality	Quantitatively - Defect density

3. Methods

This section will discuss the research methods which were utilised within the case study to investigate the benefits, highlighted in Section 2, which are presented to an organisation from implementing a subset of XP principles.

The section will also provide a more in depth discussion of the company under analysis, the case study techniques which were utilised including metric based quantitative research and documentary analysis, interview and questionnaire based qualitative research. Finally, this section will discuss data analysis and evaluation techniques which were used to analyse the results gathered from the research performed.

3.1 Overview of Case Study Approach

The aim of the project is to investigate if an organisation will still obtain tangible benefits in software development productivity through the implementation of a subset of XP principles. To investigate this fully, the primary research method utilised is a case study of Company X, which develops software by implementing a subset of XP principles.

A case study has been utilised as a large number of research papers consulted during the review of literature highlighted this method as the primary research method utilised when investigating agile methods (Capiluppi et al, 2007; Cao et al, 2004; Copeland 2002; Fitzgerald et al, 2006; Layman et al, 2006; Maurer & Martel, 2002a; Maurer & Martel, 2002b; Maurer & Melnik, 2006; McMahon, 2003; Rasmusson, 2003). Case studies have been widely used within these papers as it has not been possible to make changes to the systems currently in place in the case study organisation due to the size of the systems. For this reason case studies have been used to validate the theory of the researcher rather than test the theory. As the systems in place within Company X are similarly of a large scale it is not possible to make changes to them. Therefore, this investigation has taken a similar approach to those discussed and will validate rather than test the hypothesis.

A case study is used to *"contribute to our knowledge"* (Yin, 2003, p1). The author explains it is a comprehensive research strategy as it involves design, data collection techniques and data analysis. By undertaking a case study, this project will gain a detailed insight into the *"instance of one thing"* and the processes which it undertakes (Oates, 2006). In this project the "thing" is Company X and the "process" is the agile development implementation which Company X utilises. The case study used in this project is both direct and indirect (Yin, 2003). Direct research has been conducted through interviews, questionnaires and metric calculations and indirect research through documentary and code analysis.

The case study has taken the form of that discussed by Yin, (2003):

1. Identifying the case study problem

2. Designing the case study

3. Preparing for data collection

4. Collecting the evidence
5. Analysing the evidence
6. Reporting the case study

The remainder of this chapter will discuss these steps undertaken by grouping them into 3 main areas: identification and design of the case study, preparation and collection of case study data and finally analysing the evidence and reporting the case study.

3.2 Identification and Design of the Case Study

The first two areas of the case study as proposed by Yin (2003) include identifying the case study problem and designing the case study to be performed. The case study problem in this project is to investigate the research question:

Does implementing a subset of the 12 extreme programming principles provide Company X with the substantial benefits associated with the full implementation of the XP methodology?

The case study was carried out on a large scale software development organisation, in order to enforce anonymity this organisation has been referred to as Company X. Company X has been selected as the case study organisation because the company satisfies the criteria of a typical agile organisation, as defined in Section 2.2, namely: Company X has a small development team, operates a same room policy, operates in a dynamic environment and works on large scale software development projects. Furthermore, as Company X can be categorised as a typical agile organisation the results of the case study will be validated as it is envisaged that the results will be typical for similar companies.

3.2.1 Company X

Company X is a large organisation with over 10,000 employees and operates on behalf of 2.3 million people. The company has primarily been involved with providing a 24-hour emergency response facility, communicating with the public and providing general assistance since 1975, however specialist units are also in operation to provide more in-depth facilities.

The in-house I.T. department develops and maintains software applications for use across the whole of Scotland including the organisation’s website, intranet site, 24-hour emergency response system and bespoke systems for each specialist unit. The applications developed are typically large scale information based transactional systems which allow users to store and retrieve information from a centralised database. These types of system are typically those which would be developed using agile methods due to their large scale and dynamic nature. The overall development team consists of more than 50 developers who are split into 7 core teams for each application. The teams provide development, support and maintenance to the systems currently in place and also those in development.

The development of software has taken a traditional waterfall approach; however more recently the company has adopted the use of agile methods within 2 core development teams.

Although the organisation has not officially evaluated the performance of this development method, it has been anecdotally viewed as a success. Therefore, the results of this case study will prove beneficial to Company X as they will provide a fuller and more rigorous evaluation of the organisation's agile implementation.

Initial documentary analysis of Company X (2004; 2006) identified that the agile development approach undertaken involves implementing 9 of the 12 XP principles: the planning game, small releases, simple design, testing, refactoring, collective ownership, 40-hour week, continuous integration and coding standards. The development approach does not implement the metaphor, pair programming or on-site customer principles. As highlighted in the literature review, these three absent principles are common omissions by development teams who implement only a subset of the 12 XP principles for the following reasons:

Metaphor: Rumpe & Schrder (2002) explain that many XP teams do not exercise this principle as they are unsure how to implement it correctly.

Pair Programming: Copeland (2001) discusses that pair programming is difficult to practice due to team size, location or a subcontracting of work. It is also reported by Glass (2001) that it requires a certain type of programmer to be successful.

On-Site Customer: Cao et al (2004) report that direct access to a customer is not always available and so this principle is not always exercised within XP teams.

The results of the case study will therefore prove beneficial as they will provide an indication of the typical results found from implementing a subset of XP principles. As the implementation in this study does not include the three most commonly omitted principles the findings may be typical of other organisations implementing a subset of principles, as the 9 principles utilised within this study are the most likely combination to be exercised.

Projects under Investigation

In order to investigate the benefits which Company X has achieved through utilising this agile approach, the case study has focused on two development projects, one traditional and one implementing a subset of XP principles. To ensure anonymity of the organisation is maintained, these teams will be referred to by their acronyms: PP team and ICRS Team. The two systems developed by these teams are detailed below:

- The PP team have developed the PP system using a traditional waterfall approach. The system has been in use for approximately 5 years and is developed and maintained by a team of 3 developers using the Java programming language.
  The PP system is a large scale software development and consists of 1170 Java files with a total of 264,316 lines of code. The system is a screen based data transaction system which interacts with a central database. The user accesses the system through the organisation's intranet page and inputs data via a form based screen. The information can also be retrieved and displayed on screen as a report. The PP system is used daily by more than 500 users and is used to record personal, medical and locality details of individuals.
- The ICRS team have developed the ICRS system by implementing a subset of 9 XP principles. The system has been in use for approximately 2 years and is developed and maintained by a team of 3 developers using C# programming language.

The ICRS system is a large scale software development and consists of 235 C# files with a total of 44,971 lines of code. The system is a screen based data transaction system which interacts with a central database. The user accesses the system through the organisation's intranet page and inputs data via a form based screen. The information can also be retrieved and displayed on screen as a report. The ICRS system is used daily by a specialist team of around 500 users and is used to record information, statements and locality details of events.

As can be seen, the systems which will form the core of the case study have been selected as they are two very similar systems. Both systems are large scale, screen based data transaction systems which interact with the same centralised database. The systems are also used by approximately the same number of users and have been developed in similar programming languages (as C# is based around Java). The analysis of these systems should validate the results of the case study as both systems are similar, thus also reducing the number of limiting factors placed upon the results. Although the size of the two projects differs this is because a key characteristic of agile methods is to produce smaller projects (i.e. less lines of code per project) and so it is unlikely that two similar sized projects could have been accessed within Company X in order to validate the results further.

3.3 Preparation and Collection of Case Study Data

As identified in Table 2 (p26), the potential benefits which the case study is evaluating can be classified as measurable qualitatively or quantitatively, therefore the case study has involved performing both quantitative and qualitative research. The quantitative research undertaken involved performing metric calculations on the code listings of both projects. These measurements provide precise statistical values detailing the extent to which Company X has encountered success through the implementation of its agile approach. The qualitative research conducted involved documentary analysis, interviews with stakeholders and a questionnaire of ranked statements to measure project risk. The results of this research provide information on the nature of the development process undertaken, opinions from key stakeholders on the extent to which Company X has encountered the reported benefits of XP and a measure of project risk based on defined statements which are viewed as the main factors contributing to the risk of a project.

The remainder of Section 3.3 will discuss the quantitative and qualitative research undertaken within the case study, including metric based research, interview based research and questionnaire based research.

3.3.1 Metric Based Quantitative Research

The quantitative research performed within the case study has involved the performance of metric calculations on code listings of each project. Mőller & Paulish (1993) explain that this will provide an indicator of the development process and any problem areas within it. It was therefore envisaged that this research method would provide a precise indication of the effect that the agile development approach has had on Company X. This is supported by Yin (2003) who explains that metrics produce an *"accurate, precise and definite result"*, thus the results will be used to deduce if the hypothesis has been proven or not. Moreover, Yin (2003) further describes metrics as giving a true insight of results in comparison to using other data collection techniques such as questionnaires or interviews as these tend to encounter bias or imprecise results. Furthermore, as a key practice of agile development is to produce little documentation, Company X has fewer historical data records for its agile project and so it was envisaged that a comparison of historical data records could provide an unfair advantage to the traditional project as there would be a greater volume of data for that project.

Therefore, the metrics introduced in Section 2.4 were performed on each application to gain an insight into the applicability of these metrics with regards to the results obtained being of value to the researcher. As discussed previously, both projects under investigation are large scale software developments with a large number of files in each application. It was envisaged that by using an automated metric suite the time taken to perform the metrics and also the disruption caused to Company X would be reduced. In order to identify suitable applications which could be used to automatically perform the metrics a list of required criteria were produced, this included:

1. The application should perform all of the identified metrics in order to limit the disruption to Company X.
2. The application should be available as open source/freeware/shareware in order to minimise costs within the project.
3. The application must operate on Windows XP operating system as this is the operating system in use by Company X.
4. The application must be capable of performing the metrics on a large number of files as both projects under investigation are large scale.
5. The application must be capable of performing the metrics on files within a hierarchy structure as both projects are complete projects with hierarchical file structures.
6. The application must be capable of performing the metrics on Java files as the traditional project was developed using Java.
7. The application must be capable of performing the metrics on C# files as the agile project was developed using C#.
8. The application must provide results on a per file basis in order to allow manipulation of the results to allow data standardisation.

After an initial review, a short list of 5 automated applications was constructed. These potential applications can be viewed in Table 3 below. This table also includes a grid which indicates which of the 8 criteria the application satisfies. Details of the websites where each of the applications can be accessed are included in the reference list within Appendix 1, (p65).

Table 3 – Shortlist of Automated Metric Calculation Applications

Application	Criteria 1	Criteria 2	Criteria 3	Criteria 4	Criteria 5	Criteria 6	Criteria 7	Criteria 8
CCCC		√	√			√	√	√
CodeAnalyzer		√	√	√	√	√		√
KLOC		√	√	√	√		√	√
LOC Metrics		√	√	√	√	√	√	√
SourceMonitor		√	√	√	√	√	√	√

As can be seen from Table 3, none of the applications perform all metrics required for the study (criteria 1), however a combination of LOC metrics (http://www.locmetrics.com) and SourceMonitor version 2.4 (http://www.campwoodsw.com/sourcemonitor.html) cover the greatest number of the metrics required including: number of files, number of lines of code, percentage comments, average project complexity and weighted methods per class. Furthermore, both of these applications satisfy the other remaining criteria and so an automated metrics suite was created by utilising both of these applications. Both of these applications were performed on the agile and traditional projects to gain results for most of the metric calculations. However not all metrics could be calculated in this way and so some metrics required data from Company X to calculate the result whilst the remainder of metrics were performed manually.

Manual Performance of Metrics

As not all metrics could be calculated by the automated metrics suite, 5 metrics required a combination of data from Company X and data obtained from the performance of the automated suite to calculate a value for these, including: project dynamism, defect density, defect detection, error fix and productivity. The data required from Company X (such as development time, number of errors) was collated during the interviews with the developers of each project and the information was manipulated to produce results for these metrics.

Furthermore, as the combination of the automated metrics suite and the data gathered from Company X could not provide ample data to perform the three remaining metrics: coupling, cohesion and number of children; these were calculated manually on a random sample of 20 files for each project. This provided an average representation for these measurements and again limited the disruption to Company X as a total of 40 files were manually examined.

Standardising Data

In order to perform statistical analysis, all metric data collated has been standardised by calculating a mean value. Oates (2006) explains that calculating the mean is the most popular way to describe the process of obtaining the average value when using a large volume of data such as that obtained in this study. The author further confirms the use of this technique by explaining the calculation of the mean value *"is widely used"*. Therefore, the results of the data gathered have been standardised by calculating the mean value across the number of files in each project. Standardising the data in this way this has reduced the possibility of the occurrence of biased results due to the difference in project size (i.e. the traditional project has 1170 files whereas the agile project has only 235 files). Therefore it was envisaged that this standardisation would ensure the results gathered were accurate as a representation across the whole of each project.

Applicability of Metrics

When performing the proposed metrics, constraints arose which limited the results of the metrics and therefore caused some metrics to no longer be applicable within this study. These constraints and the metrics which they have affected are detailed below:

- **Project Dynamism:** as the project teams log changes to a project after development and not during its development a measurement for this metric could not be produced. Thus, this metric was not performed as an accurate figure could not have been given as a result.
- **Software Quality, Defect Density and Time Taken to Fix an Error:** as one of the main practices of agile methods is the reduction in documentation, Company X holds no information on the number of defects which have occurred within the agile project.

Therefore, as only a traditional value could be accessed these metrics were not utilised as the result could not be used to prove or otherwise if software quality was increased, if the number of defects within a project had decreased or if the error fix time had been reduced by using the agile approach adopted by Company X.

- **Productivity:** this metric is not applicable as it calculates the productivity of a developer based on the number of days worked on a project, regardless of the number of hours worked per day. During the development of the traditional project, the developers performed several additional hours of overtime, unlike the agile developers. Therefore, this metric was not calculated as the result would provide an unfair advantage to the traditional project as the traditional developers performed more hours of work per day than the agile developers. Furthermore, as Company X does not keep a log of hours worked per day it was unfeasible to use another similar measure of productivity as the precise hours worked per day could not be provided for either project.
- **Cost:** the cost of the project could not be measured as the systems under investigation are two separate systems which were developed in different time periods with different functionality. Therefore, the cost of the two projects could not be calculated as the results could not be compared to give an accurate change in the project cost. Furthermore, in this instance the cost is irrespective of the development methodology.

As these metrics have been identified as not applicable to the study they will not be investigated as the results would not be accurate. The following table provides a summary of the metrics which were performed within the case study, the benefits which were measured by these metrics, the data which was required to compute the results and finally where the required data was obtained.

Table 4 – Metrics Performed in Case Study

Benefit Measured	Metric	Data Required	Data Obtained From
Clean/Clear Code	Coupling Cohesion % Comments	Coupling within a file Cohesion within a file Percentage of comments in a file	Manual calculation Manual calculation Metrics suite
Code Complexity	McCabe Cyclomatic Complexity	Average complexity	Metrics suite
Earlier Defect Detection	Defect Detection	Stage of occurrence when defects are detected	Company X
Robustness of Code	Number of Children	Number of children subordinate to a class	Manual calculation
Maintainability	Coupling Cohesion Weighted Methods per Class	Coupling within a file Cohesion within a file Sum of all complexities	Manual calculation Manual calculation Metrics suite

3.3.2 Documentary Analysis Based Qualitative Research

Initially in the case study documentary analysis was conducted to uncover the specific development process utilised by Company X. Although Company X has historical data records for the traditional PP application (e.g. number of defects etc.) they have no standards documentation which could be analysed as the development approach is ad hoc. Therefore, the documentary analysis performed involved the consultation of standards documentation on the design and implementation of the ICRS application, thus providing a basis of investigation of the agile methodology implemented. To investigate this, the following standards documentation were analysed: "Development Process" (Company X, 2006) and

"ICRS – System Architecture & Development" (Company X, 2004). The results of this documentary analysis show the agile approach undertaken by Company X involved implementing a subset of 9 of the 12 XP principles, including: the planning game, small releases, simple design, testing, refactoring, collective ownership, 40-hour week, continuous integration and coding standards. The development approach does not implement the metaphor, pair programming or on-site customer principles. After an initial discussion with the technical lead of the agile project the reasons for not implementing these principles became clear, these included:

- The metaphor was not implemented as the project team were unsure how to implement it successfully. This was reported by Rumpe & Schrder (2002) as a common reason for teams not to implement this principle. Furthermore, the technical lead explained that the team were unsure if they should have a separate metaphor from the customer, or if the metaphor should change throughout each of the project's iterations.
- Pair programming was not implemented as the development team only consisted of 3 developers and so implementing this principle would only lead to one pair being formed. Thus the technical lead envisaged that by implementing pair programming with such a small development team the productivity of the developers would be reduced. This supports a report by Copeland (2001) who comments that pair programming is often not implemented due to team size.
- The on-site customer principle was not implemented as the customer representative within the project was a senior member of staff who could not afford to be placed within the I.T. department for a six month secondment. This supports the findings of a study by Cao et al (2004) who report that this principle is difficult to fulfil as the customer is not always available.

As the 3 principles not utilised by Company X are commonly omitted from XP when implementing only a subset of principles, the results of this case study may prove beneficial as they can provide an indication of the results which may be encountered by similar organisations implementing only a subset of XP principles.

3.3.3 Interview Based Qualitative Research

The interviews conducted during the case study were semi-structured interviews. Semi-structured interviews ensure all question areas are covered and also allow the participant to speak freely (Oates, 2006). By utilising this interview technique, the researcher was provided with the opportunity to ask further questions based on the answers provided.

A total of 6 participants were interviewed, including 2 developers and the customer representative from each project. The developer interviews were conducted at the premises of Company X to minimise disruption to the organisation. Due to locality constraints, the interviews with the two customer representatives were conducted via email at a pre arranged time with the participants. This involved the initial set of questions being emailed to the customer representative who then provided answers which after analysis by the researcher led to the development of further questions.

Two developers from each project were interviewed to provide an insight into the development process and to gain each developer's view on the benefits which they have received from their development method. It was envisaged that by interviewing two developers from each project the results obtained would be validated as this accounts for more than half of the project's development team and so would provide an average sample. Should the interviews only have been conducted with one developer from each team this could have led to the results not being wholly representative of the overall project team.

The customer representative from each project was also interviewed to gain the customer's view of the system and their involvement in its development. The customer was interviewed to detail the extent to which the benefits presented to the customer match the benefits proposed by a full implementation of XP. Due to the approach undertaken by all development teams in Company X, there is only one customer representative consulted during the development process for each core application, therefore only one customer representative of each project was available for interviews.

As discussed, the semi-structured approach of the interviews highlights questions and topic areas of discussion. These questions/topic areas can then be extended upon during the interview process depending on the answers obtained from the participant. The topic areas and questions for discussion within the interviews have been formulated from the findings of the literature review, thus the benefits detailed in Table 2 (p26), which are measurable qualitatively. There are 18 specific benefits which can be investigated by interviews. These benefits are not all applicable to one project team member and so the interview questions differ for each type of participant including traditional developer, agile developer and customer representative. The question areas relating to each participant type are further detailed below:

- The questions for the traditional developers were formulated from the results of the review of literature. As this project aims to provide a before and after view of the implementation of the agile approach followed by Company X it was important to record if the traditional project experienced the benefits reportedly gained through utilising a subset of XP principles before the implementation of that method. Should this have occurred, this would indicate that the agile approach adopted by Company X was not the cause of the reported benefit as this had been found before the implementation of this method. The initial questions/topic areas for the traditional developer interviews can be referenced in appendix B1 (p72). The questions were derived from the benefits reportedly gained from utilising agile methods which can be found in Table 2 (p26). Table 2 was referenced and each benefit which was measurable qualitatively was addressed within the questions. Questions relating to the problems associated with the software engineering industry were also included to deduce if the traditional project encountered these. The actual interview transcripts from the traditional developers can be referenced in appendices C1 (p78) and C2 (p83).
- The questions for the agile developers were formulated from the results of the review of literature. In order to investigate the results of implementing a subset of XP principles it was important to record if the reported benefits were experienced. This would indicate that the agile approach adopted by Company X could be the cause of the reported benefits as the benefits had been found after the implementation of this method. The initial questions/topic areas for the agile developer interviews can be referenced in appendix B2 (p74). These questions were derived from the benefits reportedly gained from each XP principle which can be found in Table 2 (p26). The actual interview transcripts from the agile developers can be referenced in appendices C3 (p87) and C4 (p93).

- The questions for the customer representatives were derived from the literature review. In order to investigate the results of implementing a subset of XP principles it was important to record if the customer had experienced the benefits which were reported. This could indicate that by implementing a traditional development model or a subset of XP principles the customer representative had been provided with these benefits. The initial questions/topic areas for the customer representatives can be referenced in appendix B3 (p76). These questions were derived from the benefits reportedly gained from each XP principle which can be found in Table 2 (p26). The actual interview transcript of the traditional customer representative can be referenced in appendix C5 (p97) whilst the interview transcript of the agile customer representative can be referenced in appendix C6 (p99).

Applicability of Interview Questions

When performing the proposed interviews with the developers it became clear that the integration process used by the traditional team was the same process used by the agile team in which the system was built frequently every day. It was therefore not applicable to measure if Company X had experienced a simplified integration process through the use of their agile implementation as both the pre-agile and post-agile methods used were the same, thus this benefit has not been investigated within this study.

The following table summarises the benefits which were addressed within the interviews and which interview participant was questioned on this benefit.

Table 5 – Benefit Measured by Interviewing Specific Participant

Benefit Measured	Traditional Developer	Agile Developer	Customer Representative
Reduction in planning time	√	√	
Reviews often	√	√	√
Rapid feedback provided	√	√	
Ensures requirements being met	√	√	√
Increased customer satisfaction	√	√	√
Provides overall view of the system	√	√	√
Provides an understanding of the project	√	√	√
Provides a relation to the project	√	√	√
Improves communication	√	√	√
Allows knowledge to be spread throughout the team	√	√	
Easier to implement changes	√	√	
Less change within a project	√	√	
Increased developer job satisfaction	√	√	
Increased developer confidence	√	√	
Developer training reduced	√	√	
Stimulates and motivates developers	√	√	
Easy resolution of issues and setting of priorities	√	√	

3.3.4 Questionnaire Based Qualitative Research

As discussed in the literature review, to establish the risk of each respective project a questionnaire of ranked statements was issued to both the traditional and agile developers. This questionnaire featured 21 statements which are defined by Jiang et al (2002) as common factors of project risk such as the availability of users or the complexity of the system functionality. Each of the statements were ranked by the participants on a scale of 1-5 in order with agreement where the highest value is equal to a strong agreement with the statement. The values obtained were then accumulated to provide a risk value for each project where the greater the value the greater the risk of the project, thus the project with the highest risk level was the project which had the largest accumulated value. The questionnaires and the results from each developer can be viewed in appendix D (p101).

3.4 Analysing the Evidence and Reporting the Case Study

The results of both the quantitative and qualitative research were analysed using data analysis techniques before the results could be presented or commented on. To analyse the quantitative research a statistical comparison was conducted on the results of the metrics. In order to analyse the qualitative research, transcripts of the interviews conducted with participants were reviewed using pattern matching techniques.

3.4.1 Statistical Comparisons

Oates (2006) explains that statistical analysis can show if links occur or if patterns are simply coincidental. In order to statistically compare the quantitative data gathered, the percentage change from the pre-agile to post-agile values were used to show the extent to which benefits have been encountered. As these results clearly show gains or reductions in the values per project more complex calculations are not required, although should these have been required t-tests could have been performed on the results gathered to test if a significant difference had occurred between the two projects. As this case study is validating the hypothesis rather than testing the hypothesis this method of statistical analysis is not required. Furthermore, as Babbie et al (2007) explain, hypothesis testing uses a sample of the population and so would not be a valid measurement as this study uses populations of data (i.e. two complete software projects) rather than a sample of these projects.

In order to collate and manipulate the raw metric data the data was entered into SPSS and manipulated in order to produce the mean and percentage change values.

3.4.2 Pattern Matching

To analyse the results of the qualitative research, pattern matching was used as Yin (2003) describes this as one of the most enviable analysis techniques to perform on case study data. Pattern matching compares an experimental based pattern with a predicted one (Yin, 2003) to calculate how similar one measurement is to another (Churbuck, 1992). This meant that the interview transcripts were analysed for patterns and trends. These could involve similar interview results from two developers which would indicate the possibility of the result gathered being accurate for the project. Finally, any patterns or trends uncovered are assessed in relation to the research question and hypothesis, to prove or otherwise the validity of the research hypothesis.

The results of the analysed data will now be reported in Section 4. The results will be documented in order of performance: metric based quantitative research, interview based qualitative research and finally questionnaire based qualitative research.

4. Presentation of Results

This section will detail the results of the case study in the order of performance of research, thus the benefits researched by metric calculations will be addressed first, followed by the benefits researched by the interviews and finally the benefit researched by the questionnaire. This section will also discuss and evaluate the results presented in relation to the benefits being experienced or not by Company X and will provide initial conclusions based on the results found.

4.1 Benefits Measured by Metrics

As shown in Table 4 (p33) the following benefits were measured by metric calculations:

1. Clean and clear code/Readability
2. Reduction in complexity
3. Earlier defect detection
4. Increased robustness
5. Increased maintainability

The results of these benefits will now be discussed in detail. The raw data which was obtained during the performance of the metrics can be accessed in appendix E (p104).

4.1.1 Clean and Clear Code/Readability

As concluded in the literature review, the benefit of producing clean and clear code and the subsequent increase in readability was measured using a combination of metrics. These included the percentage of comments found in code, the coupling within a file and the cohesion within a file. The results of these metrics can be shown in Table 6 below.

Table 6 – Clean and Clear Code Results

Project		The Percentage of Comments in Code	The Average Coupling in a File	The Average Cohesion in a File
Agile	Mean	15.1762	2.5500	26.4000
Traditional	Mean	16.8241	4.2500	22.0500
	Percentage Change	**-9.7948%**	**-40%**	**+19.7278%**

As can be seen from Table 6, the percentage of comments in code has reduced by 9.79% when utilising the agile approach. It is possible that this result could be inaccurate due to the size of projects measured as the traditional project involved performing this metric on 1170 files, whereas the agile project only had 235 files, thus providing the possibility that this could be the reason for a decrease in the number of comments. On the other hand, the average coupling found in all files across the project has decreased considerably by 40%. As Lindstrom & Jeffries (2004) explain, a project should strive for the lowest possible coupling and the highest possible cohesion values as these are most desirable. The authors also discuss that these are one of the main characteristics of well designed code. Furthermore, the cohesion in the files within the agile project has increased by almost 20% in comparison to when developing traditionally, another indicator of well designed code. It is possible that Company X is now experiencing favourable coupling and cohesion values as it is focusing more on simplicity and code structure. By implementing principles like simple design, refactoring and coding standards the developers are ensuring the least complex solution is implemented in a way which is simplistic and understandable to all developers, thus causing the quality of the code to be increased which is shown by respective increases and decreases in cohesion and coupling values.

Overall the use of the agile approach in Company X has increased the readability of the code through producing clean and clear code in two of the three categories: percentage of comments in code, coupling within a file and cohesion within a file. Therefore, these results support those reported in studies highlighted in the literature review including: Ambler, 2001; Highsmith & Cockburn, 2001; Levy, 2003; McDowel et al, 2006; Wood & Kleb, 2003. These reports collectively suggest the use of simple design, refactoring, pair programming and coding standards can cause an increase in the readability of code. As Company X implements simple design, refactoring and coding standards these results indicate a link between the use of these principles and an increase in code readability. However, as pair programming is not implemented these results suggest that the benefits associated with this principle are achieved regardless of the principle's implementation.

4.1.2 Reduction in Complexity

As concluded in the literature review, the benefit of a reduction in the complexity of code can be measured using McCabes Cyclomatic Complexity metric. The results of this metric are be shown in Table 7 below.

Table 7 – Cyclomatic Complexity Code Results

Project		The Average Complexity of the Project
Agile	Mean	2.2500
Traditional	Mean	2.8700
	Percentage Change	**-21.6027%**

As Table 7 shows, the complexity of the project has reduced by almost 22% when implementing a subset of XP principles. This is a significant reduction in a critical measurement of project success, as Kan (1995) reports a strong correlation exists between complexity and defect rates thus should the project have high complexity it would be common for it to possess a high number of defects. It is possible that the reduction in complexity within Company X is due to the higher focus by the developers on simplicity and code structure through the use of principles such as simple design, refactoring and coding standards. As discussed previously, by implementing these principles the developers are focusing on implementing the simplest solution, thus causing the complexity to be reduced. Furthermore, as the developers have frequent consultations and reviews with the customer through the small releases principle, the developers are now more aware of the functionality to be implemented which may lead to its implementation being less complex.

As the results show a decrease in complexity of code by almost 22% it can be suggested that using a subset of XP principles has led to a reduction in complexity of code within Company X. This result supports those reported in studies highlighted in the literature review (Capilippi et al, 2007; English, 2002b; Mens & Tourwe, 2004) as these suggest the use of simple design and refactoring reduce the complexity of code and Company X utilises both of these principles within its subset implementation.

4.1.3 Earlier Defect Detection

As described previously, the benefit of an earlier defect detection rate has been measured in project lifecycle stage by defect detection. The results of this metric were obtained from Company X and show the traditional project to have a post-release detection and the agile project to have a pre-release detection, thus indicating the agile project has an earlier defect detection rate. This is a significant benefit posed to an organisation as it prevents the opportunity of errors occurring when the project is released to the customer. It is possible that implementing principles such as testing, continuous integration and refactoring has led to this earlier detection rate as these principles would involve regular reviews of code by developers and the performance of automated tests on a daily occurrence during integration of the system. These characteristics would therefore lead to a detection of errors at several stages within the system's integration, thus suggesting the largest number of errors would be detected during the development of the system before being released to the customer.

As the results show that Company X has achieved an earlier defect detection rate, this supports studies highlighted in the literature review (Highsmith & Cockburn, 2001; Lindstrom & Jeffries, 2004) which suggest through implementing testing and continuous integration an earlier defect detection rate can be obtained.

4.1.4 Increased Robustness

The benefit of increasing the robustness of code can be measured using the number of children (NOC) metric. The results of this metric can be shown in Table 8 below.

Table 8 – Number of Children Results

Project		The Average Number of Children Subordinate to a File
Agile	Mean	0.8500
Traditional	Mean	0.9500
	Percentage Change	**-10.5263%**

Rosenberg & Hyatt (1997) explain that the smaller the number of children, the more robust the system is as there are less classes dependent on another. As Table 8 shows there has been a reduction of almost 11% in the number of children subordinate to a file when implementing a subset of XP principles, thus an increase in robustness of almost 11%. This is a significant increase in a critical measurement of project success as the more robust the system the smaller the probability of system failure. It is possible that this increase in robustness is due to the implementation of principles such as refactoring and simple design as these encourage developers to write the system in the simplest code possible, thus using a small number of classes which are not interdependent. The use of refactoring also promotes the restructuring of code into the simplest form which again leads to simplistic code, which in turn is highly robust.

These results show an increase in code robustness by almost 11% within Company X which suggests support for the findings of Cao et al (2004) who explain the use of refactoring leads to this increase. Therefore, as Company X implements refactoring it could be suggested that the implementation of this principle may lead to an increase in the robustness of code.

4.1.5 Increased Maintainability

As described in the literature review, an increase in the maintainability of code can be measured using a combination of metrics. This includes weighted methods per class (WMC), the coupling within a file and the cohesion within a file. The results of these metrics can be shown in Table 9 below where the coupling and cohesion values are those reported in Table 6 (p39).

Table 9 – Maintainability of Code Results

Project		The Average Weighted Methods per Class	The Average Coupling in a File	The Average Cohesion in a File
Agile	Mean	12.1702	2.5500	26.4000
Traditional	Mean	14.9846	4.2500	22.0500
	Percentage Change	**-18.7819%**	**-40%**	**+19.7278%**

Rosenberg & Hyatt (1997) explain that the weighted methods per class should be as low as possible. As can be seen from Table 9, Company X has achieved a reduction in weighted methods per class of almost 19% when utilising its agile approach. Rosenberg & Hyatt (1997) also explain that the greater the number of WMC, the greater the potential consequence on the children of that class, thus suggesting that a project with a small weighted methods per class is more robust and less at risk of project failure. It could be that Company X has achieved a decrease in WMC through principles such as simple design, refactoring, collective ownership and coding standards as these principles encourage developers to write code in the simplest way possible, thus minimising complexity which in turn increases the robustness of code as the code is easier to understand. As discussed within the clean and clear code results the average coupling has decreased favourably by 40%, whilst the average cohesion has increased advantageously by almost 20%, possibly due to the implementation of principles such as simple design, refactoring and coding standards which encourage code to be of the simplest form.

Overall, the results of the implementation of a subset of XP principles in Company X indicate an increase in the maintainability of the project as it has experienced a reduction of almost 19% in weighted methods per class, a 40% decrease in coupling of files and an increase of almost 20% in the cohesion of files – all of which are indicators of well designed code and a maintainable project. These results support those reported in studies highlighted in the literature review including: Erdogmus, 2007; Fitzgerald et al, 2006; Levy, 2003; Lindstrom & Jeffries, 2004; Neill & Gill, 2003. As the reports suggest that the maintainability of a project can be increased through utilising refactoring, collective ownership and coding standards the results highlight a possible link in the use of these principles and in increase in the maintainability of code within Company X.

4.2 Benefits Measured by Interviews

As shown in Table 5 (p36), the following benefits were investigated by interviews:

1. Reduction in planning time
2. Reviews often
3. Rapid feedback provided
4. Ensures requirements are being met
5. Increased customer satisfaction
6. Provides overall view of system
7. Provides an understanding of the project
8. Provides a relation to the project
9. Improves communication
10. Allows knowledge to be spread throughout the team
11. Easier to implement changes
12. Less change within a project
13. Increased developer job satisfaction
14. Increased developer confidence
15. Developer training reduced
16. Stimulates and motivates developers
17. Easy resolution of issues and setting of priorities

The results of the interviews will now be discussed in detail.

4.2.1 Reduction in Planning Time

The initial planning time of the traditional project was reported by the developers to be roughly one month, although this was said to be significantly reduced through using the agile implementation as one agile developer commented planning was: *"quite short in comparison to a traditional project"*. It was proposed by the agile developers that the reduction in planning time was due to the implementation of the planning game principle, where planning of the project takes place at the beginning of each iteration, thus 4 or 6 times throughout the project. This meant that in comparison to the traditional system where all planning was carried out at the beginning of the project, the agile project had no "up-front" planning phase, thus reducing the time taken to plan the system. Furthermore, one agile developer also explained the planning *"was easier"* thus suggesting this also lead to a decrease in the time taken to plan the project.

As highlighted in the literature review, Copeland (2002) explains that the implementation of the planning game principle leads to a reduction in planning time. The results of this study support this and indicate a link in using this principle and obtaining a reduction in the time taken to plan a project within Company X.

4.2.2 Reviews Often

The customer of the traditional project explained that they were only given one opportunity to review the system before it was released and this was at the pilot stage of the system, one week before it was due to be released, thus no major changes could be implemented in the system at this late stage. This was indicated by the customer as not often enough: *"Yes we got to see the system before it was released when it was piloted, but ...I wouldn't say regularly"*.

Now, through utilising a subset of XP principles the customer has been provided with the opportunity to review the system at the end of each iteration, approximately every 4-6 weeks. This was found beneficial by the agile developers who commented that the customer had system reviews: *"quite a lot"* during the system development. This is a major improvement on the traditional system as the customer is now aware of how the system will look and operate when released and suggests that the number of changes implemented after release would be reduced as these can now be addressed after each customer review. It is probable that Company X has achieved the benefit of regular customer reviews through their implementation of the small releases principle which has caused it to break the system into small pieces of functionality which are: *"signed off"* by the customer before moving onto the next iteration. Furthermore, the high emphasis placed on customer collaboration within an agile project may have led to the increase in the number of system reviews as customer feedback is now more critical to the success of the project.

There is an indication therefore that Company X has experienced an increase in the number of customer reviews which take place within a project through implementing a subset of XP principles. This result also supports studies highlighted previously including: *"Extreme programming: The Zero G experience"*, 2003; Highsmith & Cockburn, 2001. These studies suggest that by utilising the small releases principle customer reviews are performed more frequently on a project.

4.2.3 Rapid Feedback Provided

As the traditional project only performed one customer review when the system had been developed, the developers were only given feedback at this one point. This is supported by the comments of one traditional developer who explains: *"we got feedback on requirements and changes to requirements but not the actual system because the customer didn't see the system until nearer the release date"*. The other traditional developer also discussed that the development team were given feedback at the pilot stage when *"it was the first time that the customer could see the system"*. As the review was performed at the pilot stage, this suggests that the feedback given by the customer could not be implemented before the system was released one week later.

On the other hand, the agile project received feedback from the customer after every review, roughly every 4-6 weeks during development. One agile developer commented that this was a quicker way to get feedback and on the whole more feedback was given because: "*you are putting the customer in the spotlight all the time and you get information out of them all the time"*, also because the regular reviews allows the system to be: "*more fresh in their mind and the feedback is more to the point"*. This is a great advantage to the agile project as it allows feedback to be acted upon and provides a system which is more like the customer specification due to the frequent refinement. Rapid feedback has been provided to the agile developers due to the utilisation of the small releases principle which has encouraged the full development team, including the customer representative, to have regular system reviews of the system thus allowing feedback to be generated.

As Company X have experienced rapid feedback, this supports several studies (Ambler, 2000b; Highsmith & Cockburn, 2001; Rasmusson, 2003; Williams et al, 2004) that indicate that the use of small releases, testing and on-site customer principles provide developers with rapid feedback on the system. As Company X utilises the small releases and testing principles it could be concluded that the results show a link between utilising these principles and achieving rapid customer feedback. On the other hand, as Company X does not utilise the on-site customer principle which is reported by Williams et al (2004) to produce rapid feedback, it is possible to conclude that the agile approach in this study has achieved this benefit regardless of the principle being implemented.

4.2.4 Ensures Requirements are being met

On release, the traditional project was said to be incomplete by the traditional developers: *"the initial release had some bits missing"* and so it did not meet the initial requirements. This was also reinforced by the customer representative: "*I was told the whole system would be released at once but it wasn't"*. This caused the customer to believe that the system did not meet its requirements: "*the system did meet the requirements once it was fully implemented but not in its first release"*. This was mainly as the system had to be implemented for legislative reasons and so only part of the system was developed in time for the release. As stated within the literature review, requirements of a project not being met are one of the main problems within the software engineering industry (Dewey, 1988 as cited in Krishnan et al, 1999).

In contrast, the developers and customer of the agile project agreed that the developed system met the requirements of the customer as one developer commented: "*it does meet the requirements"*. This was because the use of the small releases principle allowed the development methodology to adapt to changes from the customer within each iteration and so the system evolved along with the requirements. This is supported by one of the agile developers who explained: *"we could incorporate changes they wanted throughout its development"*.

By implementing a subset of XP principles Company X has achieved the benefit of the project fulfilling the requirements. This result supports a study by Graham (2004) who reports that the use of the small releases principle ensures the developed system meets the customer's requirements. Furthermore, as prior to utilising the agile approach the traditional project did not initially meet the customer's requirements it can be said that through utilising the agile approach Company X may have minimised one of the problems within the software engineering industry of producing a system which does not meet the customers requirements.

4.2.5 Increased Customer Satisfaction

Within the traditional project the customer and traditional developer both reported that the customer was not entirely satisfied with the project as the customer explained they were: "*not* [satisfied] *when the system was first released"*. This was supported by one developer who commented that at release the customer was: "*I would say 75% satisfied"*. This was because the project was delivered with pieces of functionality missing as it was not fully implemented on time.

The agile project reported a high level of customer satisfaction as the customer was constantly kept informed of project progress and could see working parts of the system regularly through principles such as small releases. Furthermore, the agile customer reported that in comparison to a traditional project, on which he had previously been a customer representative, he has definitely experienced an increase in satisfaction as the system was developed quicker: "*I have been a customer representative before for a traditional project and I can definitely see an improvement…it's not at all what I'm used to"*. The agile developers also commented that the customer is: "*definitely"* more satisfied because they could see progress in the system and were constantly kept up to date.

Since Company X has achieved an increase in customer satisfaction, possibly through utilising a subset of XP principles, this supports reports by Maurer & Martel (2002a) and Williams et al (2004) which collectively suggest that the use of the small releases and on-site customer principles provide an increase in the satisfaction of the customer. However, as Company X does not utilise the on-site customer principle which is reported by Williams et al (2004) to increase customer satisfaction, it could be concluded that this principle has been achieved regardless of the required principle being implemented.

4.2.6 Provides Overall View of the System

At the beginning of the traditional project the project team did not fully have an overall view of the system and what it would do as both developers were unfamiliar with the business logic of the system and so had to discover this as development progressed. As one developer explained: "[I] *didn't really know a lot about what the system was doing, the actual business logic of it"*. The customer representative was also not aware of what all parts of the system would do as she would not use all functions available in the system: "*there are parts of the system that I don't use and I don't know how they work*". This was supported by one developer who commented: "*at the start of the project the customer didn't really know what the system had to do so we found it hard to get requirements from them"*.

Alternatively, the agile project team reported having an overall view of the system at the beginning of the development possibly as the customer was aware of what the system was required to do: "*we had a rough outline of what had to be completed in order to finish the project"*. One developer also commented: "*I knew all the different parts of the system…so yes I did have an overall view of the system"*. It is possible that the use of principles such as collective ownership have allowed the agile developers to have an overall view of the system as they can work on any part of the system at any time and thus are aware of other areas of the system which are being developed in parallel with their own areas of functionality.

The result suggests that Company X has achieved the benefit of having an overall view of the system at the beginning of the systems development. These findings support those of Maurer & Martel (2002a) who report that implementing the metaphor principle provides the project team with an overall view of the system at the beginning of development. As Company X does not utilise the metaphor principle it could be concluded that the benefit associated with this principle has been achieved regardless of its implementation.

4.2.7 Provides an Understanding of the Project

At the beginning of development, the traditional project team did not fully understand the project. This was reported as being mainly because the customer was unsure of what the system was required to do. The customer also explained that she would not use all aspects of the system and so found it difficult to understand those parts: "*there are parts of the system that I don't use...so I didn't understand them*". This led to the developers not fully understanding the project as they were not given specific requirements for all parts of the system. When asked about the understanding one developer had of the project he commented: "*not 100%, no*".

Conversely, the agile team reported having an understanding of the project and the development approach undertaken. Both agile developers explained that they: *"definitely"* understood the project. This was due to the high level of user involvement within the agile project. Initially, the project team and customer representative discussed the approach that would be followed and the input which would be required from the customer representative. This ensured the customer was aware of the details of the project and his role in specifying requirements: *"I did understand how the project was being developed...and what would be expected of me"*. The project team also discussed everything which would be included within the system in simple terms in order to prioritise requirements. This suggests that not only the small releases principle but also the planning game may have led to the project team having an understanding of the project as the full scope of the project had to be understood initially in order to prioritise requirements into iterations. Furthermore, by having regular reviews through the small releases principle the development team have ensured that this understanding has been maintained throughout the project.

As Company X has experienced an understanding of the project throughout its agile team this result reinforces the findings of Lindstrom & Jeffries (2004) and Williams et al (2004). These reports further detail that the use of the metaphor principle leads to this benefit being achieved. Therefore, as Company X does not utilise this principle the result indicates that the organisation may have achieved this benefit regardless of the corresponding principle being implemented.

4.2.8 Provides a Relation to the Project

The development team of the traditional project reported that they did not all have a relation to the project as they were unsure of the full system requirements and what they would be required to implement. One developer commented: *"I couldn't exactly relate to all the functionality because I didn't know everything that would be implemented inside out"* this is supported by "*I didn't know absolutely everything that I would be responsible for*". Furthermore, the customer of the project reported that they were also unsure of their role in the project: "*I'm not exactly sure of the role*" thus suggesting that their role may not have been fulfilled properly.

On the other hand, the agile project reported that all project team members had a relation to the project and their role within it. The customer explained that he could relate to the project as he had a very important role within it: "*I could relate to it because it was me who was telling the developers what it should do*". Furthermore, as the customer had a large input in the development the developers could relate to the project more: "*I was aware of what I would be doing…* [and what] *I was expected to implement*". Therefore, as the customer had a large input within the project this has led to the overall team having a clearer understanding of the project and thus a clearer relation to it. This suggests that principles such as small releases and collective ownership may have led to the project team having a relation to the project. By attending regular reviews at the end of each 'small release' the project team have been kept up to date with the project and their role within it. Furthermore, by practicing collective ownership, the developers have obtained a clearer view of the overall system and their position within its development.

It could be concluded that by utilising a subset of XP principles Company X has increased the relation to the project within the agile team, thus supporting the findings of Lindstrom & Jeffries (2004). The authors detail that the use of the metaphor principle leads to this benefit being achieved. As Company X does not utilise the metaphor principle this would suggest that the agile approach in this study has achieved the benefit of providing a relation to the project regardless of the metaphor principle being implemented.

4.2.9 Improves Communication

The communication within the traditional project between the developers and the customer was mainly via email and telephone. One developer commented that this could have been improved: "*the communication with the customer could have been better, we didn't talk to the customer too much and to be honest with hindsight I think that was a mistake*". It is possible that this lack of communication with the customer was because the customer was unsure of her role within the project or indeed the requirements of the system. The customer also explained: "*I didn't ever really have a need to phone that much*" which indicates the customer was unsure of her position within the team and how critical it is to the success of the project. This lack of communication with the customer was highlighted by the one developer who discussed that the customer was: "*quite hostile*" and "*very apprehensive*". These characteristics all suggest that there was a low level of communication between the traditional development team.

Conversely, the agile development team all reported an increase in communication through utilising the agile approach. The communication between the agile team involved email, telephone calls and customer review meetings. Both developers explained that they could easily have communicated with the other developers and this helped in meeting deadlines for iterations. One developer explained: "*even the communication with the customer was very good*". Furthermore, the customer representative of the agile project explained that the communication had improved in relation to a traditional project: "*before I never got to see the system but this way I got to see the developers and the system every month. The communication has definitely increased*". It is possible that this increase in communication is due to the implementation of XP being more customer oriented as it focuses on high customer involvement which would not be possible without frequent communication. Furthermore, principles such as small releases which encourage regular customer reviews may have promoted a higher level of team communication.

As the agile team have experienced an increase in communication it could be suggested that by utilising a subset of XP principles Company X has experienced an increase in communication throughout the project team. This result would then support the findings of Biggs (2000) and Williams et al (2004). The authors report that the use of the metaphor and pair programming principles lead to this benefit being achieved. As Company X does not utilise these principles the result indicates that the organisation may have achieved this benefit despite the related principle being utilised.

4.2.10 Allows Knowledge to be spread Throughout the Team

Both developers of the traditional team conceded that the knowledge of the system was not spread throughout the team as they could not work on the code of another: *"I don't think we knew enough about each others code or functionality"*. This was mainly because the developers were not aware of the functionality or business logic of the full system: *"I don't think they fully knew what functionality my code did"*. Furthermore, as the traditional developers do not have access to all system code the developers are restricted only to knowledge of their own area of functionality. Additionally, as the traditional developers develop code using their own style this may have made it more difficult to understand the work of another.

Conversely, the agile developers both reported that the knowledge of the system was definitely spread throughout the team as developers could work on the code of another, "*we both knew what each other was working on and could have worked on each other's code no problem"*. This is perhaps because when planning each iteration the developers were informed of the functionality and business logic of the full requirements to be implemented within that iteration and not only the developers specific implementation area. Also, as the agile team develop code using coding standards all code is written to a specific set of criteria, thus enabling any developer to understand the code of another. It is also possible that the use of the collective ownership principle has provided the agile developers with a greater access to the code of the full system and provided an opportunity for the developers to learn about other areas of code when required.

As Company X has experienced the knowledge of the system being spread throughout the development team it could be concluded that this has been due to the implementation of a subset of XP principles. This would support the findings of several studies identified within the literature review (Biggs, 2000; Dybå et al, 2007; McDowell et al, 2006; Williams et al, 2004) which report that the use of the metaphor and pair programming principles lead to this benefit. As Company X does not utilise these principles, these results may indicate that this benefit has been achieved despite the associated principles being utilised.

4.2.11 Easier to Implement Changes

The traditional development team reported some difficulties in implementing changes to the system due to the technical complexity involved: "*some changes involved quite difficult* *business logic so that took a bit longer than usual*". The developers also commented on the scale of some of the changes which lead to the changes being time consuming: "*it was the size and time consuming*". This suggests that initially Company X had an element of difficulty in implementing changes imposed on the traditional system, possibly as they did not have regular contact with the customer throughout development which meant after the system was released there were major changes imposed.

This was not the case however for the agile project where the developers reported that changes were incorporated into the project much easier than when developing traditionally: "*it is a lot easier because you don't need to change a lot to make the change"*. The developers explained that the iterative approach undertaken made changes easy to incorporate as changes were only imposed on small areas of code as opposed to the full system: *"any changes were only implemented into one part of the system so it didn't take as long and it wasn't as complicated"*. Furthermore, as the testing principle involves the developers creating tests up front, the altered code could easily be retested by using the tests which were already developed, thus decreasing the time taken to implement any changes: *"the tests had already been written so these could easily have been run"*. Therefore, by utilising principles such as testing and small releases the changes imposed on the system may have been implemented more easily. Also, as there is a greater focus on customer collaboration and system reviews in the agile approach, changes which have been imposed by the customer have been implemented at an earlier stage in the project's development (i.e. before release) which has caused integration to have less impact on the overall project.

Therefore, as Company X has experienced easier implementation of changes this result supports the findings of Highsmith & Cockburn (2001) who discussed the use of the simple design principle leads to this benefit being achieved. This result may indicate that the use of this principle could have led to the easier implementation of changes within Company X.

4.2.12 Less Change in a Project

The traditional project reported having a large number of changes throughout its development due to changes in legislation and problems with the system not initially being fully implemented. One developer commented: *"there were a lot of changes throughout the whole project"*. It is probable that this was also due to the lack of requirements specified by the customer and the lack of regular reviews which the customer had. This meant the only opportunity the customer had to detail any changes was one week before the system deployment, when it was too late. Should the customer have had regular updates and reviews of the system it is possible most of the changes required could have been implemented during the systems development.

On the other hand, the agile project reported that the number of changes were less than when developing traditionally as iterative development allows the customer to see what has been developed and what is planned for future iterations and thus allows a more overall view of the system: *"the customer is more aware of what is going to be developed in each iteration"*. This indicates that the agile project has experienced less change within the project due to principles such as small releases, where the system is broken into iterations and the customer is given the opportunity to review the system at the end of each iteration. Also, as the customer is exposed to creating requirements throughout the project due to the planning game, many changes are implemented as new requirements for new iterations as opposed to being implemented as changes to the system. This is supported by the comment of one developer who explains that changes are easier to manage than when developing traditionally: *"I do think that agile is able to accommodate the changes better"*.

These results suggests that Company X has achieved less change within their agile project and supports the findings of Highsmith & Cockburn (2001) who reported that using XP leads to this benefit through the implementation of the simple design principle. This indicates that Company X may have achieved less change within its agile project due to the implementation of this principle.

4.2.13 Increased developer Job Satisfaction

The developers of the traditional project commented that they did not have a high level of job satisfaction, mainly as they were not satisfied with the final developed system: *"I wasn't satisfied in the way it has been developed"*. One developer also commented on his dissatisfaction with the final system: *"I wasn't happy with what had been developed"*. The developers suggested that they may have increased their job satisfaction through working together during the development: *"I think we could have worked as a team more"* thus suggesting an XP principle such as pair programming may have increased the developer's job satisfaction as this would have involved the developers working together more closely.

In contrast, the agile developers reported a high level of job satisfaction due to the increase in customer satisfaction and also as they were able to see progress within the project: *"you can see development and progress and it makes the customer happier"*. One developer also commented that: *"you do communicate more, it keeps you busy and you see the results of your work more quickly…so that way it is satisfying"*. This suggests that having close contact with the customer and regular system reviews has increased the satisfaction of the developers as they can now see regular progress of their work and gain rapid feedback from the customer. Also, as the customer is satisfied this has increased the satisfaction of the developers: *"if you know the customer is happy then you know you are doing it right"*.

It could be concluded that the developers within the agile team have reported a greater job satisfaction than the developers of the traditional team. This supports the findings of several studies (Dybå et al, 2007; Maurer & Martel, 2002a; Williams & Kessler, 2000) that report the use of pair programming leads to an increase in job satisfaction. As this principle is not utilised within Company X this result could indicate that the company may have experienced this benefit through the implementation of another principle as the principle associated with this benefit has not been utilised.

4.2.14 Increased Developer Confidence

The traditional project reported that neither of the two developers were confident in the system they produced or the development method that they used to create it, as one developer explained: *"I wasn't confident in our system"*. This was mainly because the system was not completed on time and so only part of the system was released: *"*[we] *didn't manage to get the whole system ready in time"*.

This was not the case for the agile project which reported a high level of developer confidence due to positive customer feedback: *“probably because I was getting feedback from the customer and I knew I was making a system that they wanted, yeah I would say I was more confident”*. This indicates that by having close collaboration with the customer due to principles such as small releases, the agile developers have achieved a higher level of confidence as they are aware of the progress of the project and also the opinion and the satisfaction of the customer.

From these results it could be concluded that Company X may have experienced an increase in developer confidence through utilising XP, as also reported by McDowell et al (2006) and Williams et al (2000). These studies suggest that the use of the pair programming principle leads to an increase in developer confidence, thus suggesting that in this study Company X may have achieved this benefit through the implementation of other XP principles as pair programming has not been utilised.

4.2.15 Developer Training Reduced

The traditional developers reported that they received a small amount of training on the business logic of the system to be developed: *“we did get training from the department on what they do with regards to the business logic of the requirements”*. This was administered by the department where the system would be deployed and was aimed at showing the developers how the job was performed prior to the development of the system. It is possible that this training was required as there would be little customer contact throughout the development of the project which may suggest that the developers would not be able to refine requirements with the customer easily.

The developers of the agile project were given no additional training; one developer explained that the approach taken was more: *“training on the job”* as they learnt as they went along through the experience of other developers and high levels of communication with the customer. This indicates that by having regular reviews, through the small releases principle, the agile developers have been able to clarify requirements with the customer without the need for training to take place. Furthermore, as the developers are now encouraged to share code with other developers through the collective ownership principle this may have led to the developers learning from the experience of others. Also, as the simple design principle encourages code to be written in the simplest form this may have reduced the need for training as only simple coding structures would be utilised.

These characteristics imply that Company X may have experienced a reduction in the training required for developers through its implementation of a subset of XP principles. This would support the work of several authors (Cao et al, 2004, Lindstrom & Jeffries, 2004; Rasmusson, 2003; Williams et al, 2000) who report a decrease in developer training through utilising the pair programming principle of XP. As Company X has not implemented this principle this result could indicate that the organisation may have achieved this benefit through utilising other XP principles.

4.2.16 Stimulates and Motivates Developers

The developers of the traditional project reported that they were motivated by the project at the beginning of its development but as the project evolved the developers became unmotivated and under pressure: *"not motivated but pressurised"*. It is probable that this was due to the stress placed on the developers to ensure the system was delivered on time and also led to the developers working long periods of overtime, decreasing motivation.

In contrast, the agile developers reported that they were motivated by the project because as development advanced they could see progress in the system which encouraged them to work harder to complete the project. This is supported by the comments of one developer: *"because you can see the system developing it encourages you to keep moving on with the project and put more work in to get it finished"*. The second agile developer also explained that using XP motivates the whole project team: *"yes it's more motivating because you see progress"*. These comments suggest that the whole project team (including the customer) are motivated throughout the project as they can see progress frequently through regular reviews which are common with the small releases principle. Moreover, principles such as continuous integration allow an executable version of the system to be available at all times which providing further opportunities for the team to evaluate the progress of the project and also evaluate the progress in relation to project completion.

It is possible to conclude that the agile developers may have received motivation and stimulation through utilising a subset of XP principles. This would then support the work of Turk et al (2005) who reports this benefit is obtained through the implementation of the 40 hour week principle. As Company X utilises this principle it may be suggested that the use of this principle has caused this benefit to be achieved.

4.2.17 Easy Resolution of Issues and Setting of Priorities

The traditional project reported having a few problems resolving issues as the customer only reviewed the system one week before its deployment which caused changes to arise at a stage within the project when it was too late to implement them. This is concluded by one developer: *"they didn't see the system until it was complete, we tended to do what we wanted"*. This also suggests that the developers implemented the functionality in the manner which they preferred, possibly indicating one of the causes of issues arising with the customer. One traditional developer also explained that issues arose as changes could not be implemented into the system in time for release: *"changes they wanted were too late to be implemented because they didn't tell us until the pilot stage"*. These characteristics indicate that the traditional developers may have benefited from frequent reviews with the customer; such as those associated with the small releases principle. This would then have provided the developers with customer feedback earlier in the system's development and minimised the issues which have arisen in this project. Furthermore, the traditional project team also explained that in relation to setting priorities of the project these were not set in consultation with the customer but instead set by the developers: *"we set the priorities ourselves because we were developing the system"*. It is probable this was due to the lack of customer involvement within the project and should the customer have been involved in this task the number of issues which arose may have been reduced.

In contrast to the traditional system, the agile team found it: *"very easy to deal with"* and thus resolve any issues with the customer as they could address the problem in a future iteration, or as one developer explained: *"you can always offer an alternative"*. These characteristics were possibly only feasible through the implementation of the small releases principle which allows development to be iterative. Through this iterative development the developers have now been able to incorporate any changes to the system in future iterations which would therefore reduce the number of issues which would arise with the customer. Furthermore, as the customer is highly involved in the development of the project it is likely that the requirements would be refined by the developers and be to the customers expectation, thus again reducing the possibility for the occurrence of issues with the customer. Throughout the agile project the developers also found it easy to set priorities with the customer as these were defined at the planning stage of each iteration. This meant that as priorities changed throughout the project the development could reflect this. One developer explained this as: *"we discussed what should be in each iteration when planning it…so we could see what was most important to the customer then because it may have changed from when the project started"*. Therefore the use of the small releases and planning game principles may have led to the setting of priorities with the customer being easier as these principles have split the development of the project into iterations and allowed the customer to only focus on what is most important on a per iteration basis.

The characteristics discussed have highlighted that the agile development team have found it easier to resolve issues and set priorities with the customer. It could be proposed that this has been due to the implementation followed by the agile team - the utilisation of a subset of XP principles. This result would support the findings of English (2002b) who reports that the use of the on-site customer principle leads to this benefit being achieved. As Company X does not utilise this principle, it is possible to indicate that by utilising other XP principles the organisation may have achieved this benefit.

4.3 Benefits Measured by Questionnaire

As discussed previously, the benefit of a reduction in project risk was measured using a questionnaire of ranked statements which are proposed by Jiang et al (2002) as measurements of project risk. The questionnaires were administered to 2 developers of each project, traditional and agile, during the interviews. The scores for each ranked statement were then accumulated and the results can be shown in Table 10 below. The raw data results for the questionnaire can be referenced in appendix D (p101).

Table 10 – Project Risk Results

Project		The Risk of the Project
Agile	Sum	107
Traditional	Sum	115
	Percentage Change	**-6.9565%**

Jiang et al (2002) explain that the higher the measure of project risk, the higher the risk which is involved in the development of that project. As can be seen from Table 10, the agile project has experienced a decrease in project risk of almost 7% in comparison to the traditional project. It is possible that this reduction in risk could be due to a higher consultation with the customer which may have increased the developer’s awareness of the system requirements. This decrease could also have been because of an earlier defect detection rate within the agile system which would cause most errors to be reported before the system is released. Furthermore, it may be that an increase in system robustness has reduced the chance of project failure and thus decreased the risk of the project, or that the use of iterative development has ensured that changes can be incorporated throughout the development of the project and reduced the risk of the project in comparison to if these changes were requested after system deployment. These factors have all arisen from the utilisation of principles such as small releases, testing and continuous integration as these have ensured rigorous testing is performed on iterations of working software which has been deployed to the customer for review.

It could be concluded that the reduction in project risk which Company X has encountered may be due to the implementation of a subset of XP principles as also reported by several studies (English, 2002b; Maurer & Martel, 2002a; Williams et al, 2004). These studies suggest the risk of a project can be decreased through the implementation of the planning game, small releases, metaphor, pair programming and collective ownership principles. As Company X has utilised the planning game, small releases and collective ownership this may indicate that the organisation has achieved a reduced project risk through the implementation of these principles.

4.4 Summary of Results

The results presented within this section have indicated that Company X has achieved all of the benefits measured within table 2 (p26). Substantial evidence of the occurrence of these benefits has also been presented with reports of decreases in critical measurements such as a decrease of 40% in coupling within a file, a decrease of almost 22% in code complexity and a decrease of almost 7% in project risk. These results show the extent to which Company X has achieved the benefits of XP through implementing only a subset of XP principles. Not only has the organisation experienced the benefits associated with the principles which it has utilised as part of its subset implementation, but it has also experienced benefits which are associated with principles which have not been utilised as part of this study. This result therefore indicates that not all principles of XP must be implemented in order to obtain the benefits associated with a full XP implementation.

The following chapter will now discuss the results presented within this section and provide conclusions on the overall results in comparison to the work of others within this field.

5. Conclusions and Discussion

This section will provide a summary of the aim of the project before drawing together the conclusions which have arisen from the results presented in Section 4. These conclusions will be analysed in relation to the work of others in the area as well as the overall research question and hypothesis of the project. The section will then discuss limitations placed on the study and possible further areas of research which have arisen from the evaluation of the results of this project before providing a final conclusion to the research.

5.1 Project Résumé

The aim of the project was to investigate the claim made by Ambler (2000a) and Turk et al (2005) that to obtain significant benefits associated with XP all 12 principles must be applied within a software development. Thus, the research question investigated in this study was as follows:

Does implementing a subset of the 12 extreme programming principles provide Company X with the substantial benefits associated with the full implementation of the XP methodology?

The research question was investigated using a case study of Company X, which has recently implemented a subset of 9 XP principles including: the planning game, small releases, simple design, testing, refactoring, collective ownership, continuous integration and coding standards. Company X does not implement the metaphor, pair programming or on-site customer principles. The organisation does not implement the metaphor as the development team are unsure how it should be implemented effectively. Pair programming is not practiced as the development team only consists of 3 developers and so this principle would only provide one set of pairs and thirdly, the on-site customer principle is not utilised as the customer representative is a senior member of staff and the organisation considered it would be unfeasible for him to join to the I.T. department for a period of 6 months.

The performance of the case study has utilised metric calculations on code listings, documentary analysis of company documents, interviews with developers and customer representatives and the administering of a questionnaire in order to investigate if all benefits associated with the full implementation of XP have been encountered. The case study investigated two development teams from Company X, one traditional team and one team implementing a subset of XP principles, to provide pre and post agile results.

The data gathered from the case study research was analysed using pattern matching and statistical analysis techniques and the results have shown that Company X has received substantial benefits through implementing a subset of XP principles, as all benefits measured have been experienced by the agile development team. These results will now be discussed in more depth in the following section, which will also include an analysis of the results in relation to the work of others in this area.

5.2 Final Discussion of Results

As can be seen from the presentation of results documented in Section 4, Company X has achieved all of the benefits reportedly obtained when utilising a full XP implementation regardless of the organisation only implementing a subset of principles. Although some of the results offer substantial increases or decreases in corresponding measures, it can be said that all benefits have been achieved to a favourable extent. Examples of this degree of obtainment include small results such as a decrease in project risk of almost 7%, but also more beneficial results such as a reduction in the coupling within files by almost 40%. These results are critical measures of project success and quality and thus would be beneficial to an organisation should the decrease be small or substantial. Therefore it could be concluded that any obtainment of these benefits would be advantageously acquired by any software engineering organisation.

5.2.1 Research Question and Hypothesis

As the results have shown that through utilising a subset of XP principles Company X have achieved all of the benefits associated with a full implementation of XP to a favourable extent, this can be evaluated in relation to the research question and hypothesis. As described in Section 3, the aim of the case study was to investigate the research question by validating the hypothesis that an organisation can still obtain tangible benefits in software development productivity through the implementation of a subset of XP principles. Thus in relation to this study it can be said that the results from Company X have validated this hypothesis as it has achieved benefits in software development productivity through the implementation of a subset of 9 XP principles.

In relation to the hypothesis that the benefits obtained should be tangible, it could be concluded that this has been the case within Company X due to the range of benefits obtained. Company X has not only received benefits relating to the quality of the code produced such as a decrease in the coupling of files by 40%, a decrease of almost 22% in code complexity and an increase in the cohesion of files by almost 20%, but the organisation has also received benefits relating to the satisfaction of the developers and customer representatives and further benefits which may address the problems within the software engineering industry. In relation to the satisfaction of the developers involved within the project the study has found that not only the job satisfaction of the developers has increased but also the confidence, stimulation and motivation of the developers has increased as the developers can now see progress much easier within the project. The satisfaction of the customer representative has also been increased through the implementation of the agile project as the customer is now more aware of the progress within the project and can see regular executable pieces of the system. The customer is also more satisfied as he is able to incorporate changes into the requirements without delaying the project. Finally, the problems within the software engineering industry of projects not being delivered on time or to the customer's specification have been addressed. During the study it became clear that Company X had encountered these problems within its traditional project as only part of the system was delivered on time, thus meaning that the system did not meet all of the customer's requirements. Through the agile implementation these problems have now been addressed as the system was developed both on time and to the customer's specification. Therefore in relation to the range of benefits which have been obtained by Company X these could be categorised as tangible. Thus validating the hypothesis that through the implementation of a subset of 9 XP principles, Company X has achieved tangible benefits in software development productivity.

As the hypothesis within this study has been validated, this also proves the research question to be true that implementing a subset of the 12 extreme programming principles does provide Company X with the benefits associated with the full implementation of the XP methodology. The results also indicate that the benefits achieved by Company X have included some substantially favourable gains/reductions. Some highlights of these results will now be presented.

5.2.2 Notable Benefits Achieved

The study has not only indicated that all benefits associated with a full implementation of XP can be achieved by implementing a subset of XP principles, but also that substantial gains/reductions can be achieved in these benefits. The results of the project have shown that by utilising the agile approach within Company X the project has now been delivered on time and to the customer's specification, therefore increasing the satisfaction of both the customer and the developer as the system meets the criteria set out at the beginning of development. This result is advantageous as the project has also uncovered that when implementing the traditional approach to software development Company X experienced some of the problems which are commonly found with this approach. These include the project not being completely delivered on time and the developed system not being to the customer's specification, thus highlighting that in this instance the use of a subset of XP principles may have addressed these problems which are common throughout the software engineering industry.

Further advantageous results uncovered include a 40% reduction in the coupling of code within a file and an increase of almost 20% in the cohesion of code within a class has been achieved. These measures are defined by Lindstrom & Jeffries (2004) as the main characteristics of well designed code, thus showing the substantial advantages gained by Company X through the implementation of a subset of XP principles. As the code can be classified as well designed this may allow easier maintenance on the project in the future and should also ensure the system is robust. In relation to this, Company X has also experienced a reduction in the complexity of code by almost 22%. This is a highly favourable benefit presented to the organisation as the code produced is now easier to understand and thus suggests that other developers could easily work on this code as the complexity is now reduced. This provides an added benefit to Company X as it was uncovered during the interviews that the organisation previously had a problem in code sharing when developing traditionally as the developers could not understand the code of another. This suggests that by implementing a subset of XP principles this problem may have been addressed as the code is now easier to understand and thus share between developers. The reduction in complexity also indicates that the risk of the project may have been reduced as the system has been developed using the simplest functionality and coding structures possible which minimises the risk of project failure due to highly complex code. This is supported by a reduction in project risk of almost 7%. This is an important decrease as it helps address one of the main problems within the software engineering industry of project failure (Graham, 2004). The statements within the questionnaire utilised have been proposed by Jiang et al (2002) as common indicators of project risk, thus by utilising this questionnaire a precise measurement has been gained into the risk of each respective project which further validates the result. By reducing the risk of the project the organisation has been presented with a more stable project and possibly a more relaxed development team.

Further notable benefits include an increase of almost 11% in code robustness. An increase in robustness ensures any problems or errors within the code are less likely to propagate throughout the system as fewer classes are dependant on others. This also indicates a possible reduction in the number of defects which may have occurred within the project. Moreover, an earlier defect detection rate further increases the robustness of the system as most errors are found before the system is released to the customer, thus also providing an increase in customer satisfaction as less defects now occur after project release. Company X has also experienced a decrease in weighted methods per class of almost 19% which has made the XP system more maintainable. As the system is now more maintainable this also signifies that the code within the system is easier to use and understand which suggests that productivity could now be optimised as it may take less time to make changes to the system in future releases as code is easier to manipulate.

Not only has the project uncovered substantial benefits through utilising a subset of XP principles but it has also highlighted that the benefits of associated principles which are not implemented can still be achieved. As this study has not utilised three of the most commonly omitted principles: the metaphor, pair programming and on-site customer; the results show that should an organisation not implement these principles, the associated benefits may still be achieved.

5.2.3 Obtainment of Benefits Associated with the Principles that were Not Utilised in this Study

As discussed in the literature review, organisations may be unsure of how to implement certain XP principles, or the implementation of some principles may not be feasible. An organisation may be apprehensive in implementing the metaphor principle as they may be unsure of how to implement the principle effectively, as found in this study and also reported by Rumpe & Schrder (2002). An organisation may also be nervous of implementing the pair programming principle due to the size of the development team as found in this study and also discussed by Copeland (2001). The third most commonly omitted principle, on-site customer, may not be implemented by an organisation as it is not always possible to have direct access to a customer as Cao et al (2004) report and as also confirmed in this study. Therefore the results of this study indicate that should an organisation be looking to implement XP by omitting the metaphor, pair programming and on-site customer principles, the benefits associated with these principles may still be obtained through the implementation of other principles, as this study suggests. Table 11 overleaf shows the benefits associated with the three omitted principles and which of the remaining XP principles have provided the corresponding benefits within this study.

Table 11 – Obtainment of Benefits Associated with the Principles Not Utilised in this Study

Principle Omitted	Associated Benefit	Obtained By
Metaphor	Providing an overall view of the system	Collective ownership
	Providing an understanding of the project	Planning game, small releases
	Providing a relation to the project	Small releases, collective ownership
	Improving team communication	Small releases
	Knowledge spread throughout the team	Planning game, coding standards, collective ownership
	Reducing the risk of the project	Small releases, testing, continuous integration
Pair Programming	Increased readability of code	Simple design, refactoring, coding standards
	Increased developer job satisfaction	Small releases
	Increased developer confidence	Small releases
	Improving team communication	Small releases
	Knowledge spread throughout the team	Planning game, coding standards, collective ownership
	Reduced developer training	Simple design, small releases, collective ownership
	Reducing the risk of the project	Small releases, testing, continuous integration
On-site Customer	Increased customer satisfaction	Small releases
	Easy resolution of issues/setting priorities	Planning game, small releases
	Providing rapid feedback	Small releases

This study therefore suggests that the benefits associated with principles which are not utilised within a subset implementation can still be achieved through the use of other principles. As can be seen from Table 11, to obtain the benefits associated with the 3 omitted principles (metaphor, pair programming and on-site customer), the following 8 principles should be utilised: the planning game, small releases, simple design, testing, refactoring, collective ownership, continuous integration and coding standards. However should all benefits associated with a full implementation of XP be required, one additional principle should also be utilised - the 40-hour week principle. This indicates that in order to obtain all the benefits associated with the full implementation of XP, including the 3 omitted principles, a subset of 9 principles such as those featured in this study could be suggested for implementation.

5.2.4 Relation to Previous Work in this Area

As this project has indicated that by utilising a subset of XP principles substantial benefits can be gained, this has supported the work of Copeland (2001) and English (2002a) who report that XP should be adapted to suit the development needs of an organisation. This has been demonstrated in this study as both the research question and hypothesis have been proven, thus indicating that an organisation can achieve substantial benefits through only implementing principles which are suited to that organisation's development needs. It could also be concluded that this study suggests that XP is a development method which should be tailored to the needs of an organisations software development. In contrast to this, the results of this study have conflicted with the results of Ambler (2000a) and Turk et al (2005) who report that all 12 XP principles must be implemented fully to successfully produce a project in XP. As the results show a wide range of tangible benefits have been achieved, it is unrealistic to support the findings of these reports as this study has achieved a high level of success through implementing only 9 of the 12 XP principles.

5.3 Limitations of the Study

Although the results of the study are favourable towards implementing a subset of XP principles, it is important to note that some limitations were placed on the study which could possibly provide a restraint on the results.

As the functionality of the two projects were not the same it may be possible that one of the featured projects had a more complex functionality, suggesting that results such as a reduction in code complexity could be invalid should the business logic of the agile project be less complex than that of the traditional project. If this was the case this would mean the project with the least complex functionality would have been easier to develop as it would involve functionality which was not as difficult to implement.

Secondly, as the principles which have been implemented as part of the subset of XP principles could not be manipulated or altered within this study, it is possible that similar results may not be found should a different subset of XP principles be investigated. As this study only focuses on the implementation of 9 principles it is not able to provide comment on the success or otherwise of a different subset implementation.

Thirdly, as the metric calculations performed on the code listings of each respective project were not performed on similar sized projects (i.e. the traditional project contained 1170 files and the agile project contained 235 files) it is possible that the results may provide a biased result to either project. For example, decreases found within the agile code could be due to the significantly smaller number of files tested. In order to minimise the possibility of this occurrence the mean value has been used to standardise all results and thus to minimise the possibility of this limitation.

It is also possible that by investigating two projects within the same organisation, limitations may have been placed on the results as the organisation may enforce regulations which restrict the development of the projects. To minimise the possibility of this, the development process undertaken by each development team was investigated through discussion and documentary analysis to uncover if any organisational policies would affect the development of each respective project.

Another possible limitation could be that the development of both projects utilised a different programming language (although C# is based around Java). This could have provided one project with an unfair advantage as one development language may be implemented more easily, thus causing the development of that respective project to be less complex. However, as both languages are similar and as English (2002b) explains that XP can be implemented irrespective of the development language, it was envisaged that the possibility of this limitation was minimised.

Finally, as both the traditional and agile projects involved different developers and customer representatives from each respective project it may be possible that the data gathered from each project team may possess a bias towards their development method due to lack of experience with the other development methodology. This could particularly be the case with the traditional developers who have only used this development methodology and may have a hostile opinion of agile development.

5.4 Future work

Further research could be carried out in this area to address the limitations of this study and to explore the issues in more depth.

In order to further validate the results, research could be carried out on two projects with the same functionality, thus minimising the possibility of inaccurate results as both projects within this study have differing functionality. As Company X does not have two projects with the same functionality this cannot be undertaken within this organisation, however should future work be allowed to manipulate code it is possible this type of investigation could be utilised. Alternatively, this investigation could be implemented on a small scale experimental project which could involve the development of two systems with the same functionality by developing one traditionally and also by utilising the subset of principles involved within this study. This would provide a valid comparison which could then be used validate the results found in this study.

Secondly, the research performed within this study could be conducted on a project which utilises a different subset of the XP principles than those investigated, to validate if the reported benefits of a full XP implementation are also encountered when utilising a different subset of principles. This could include implementing a different combination of principles by omitting principles separate to those which were omitted within this study. Again, this would not be feasible within the current implementation followed by Company X as it has highlighted reasons for omitting the metaphor, pair programming and on-site customer principles. However, it may be possible to implement these omitted principles within a different context in Company X. If the literature reviewed within this study was utilised it may be possible to implement the metaphor principle successfully by developing a single metaphor for the entire project team to work towards. Furthermore, should the agile approach which Company X undertakes be adopted by another of the organisation’s core development teams, it may be possible to implement the pair programming principle if this development team were of a larger scale than that investigated within this study. Moreover, should a future implementation of XP by Company X be undertaken, it could be possible to allocate a customer representative who may be able to fulfil the on-site customer principle. The implementation of these principles would then provide an opportunity to investigate the effects on the organisation of the utilisation of a different subset of principles. Should it be possible for Company X to implement these three omitted principles as discussed, it may also be possible to investigate a full implementation of XP within one of the organisation’s software developments. This would provide an opportunity to evaluate to what extent the organisation has achieved the benefits of XP through utilising a subset, by comparing the range of these results to the results found when implementing the full methodology. This could provide Company X with a measurement of which implementation is most productive for its software developments.

Another area of future work which could be conducted would be an investigation into the smallest number of principles which could be implemented in order to obtain all the benefits associated with a full implementation of XP. As this study has only investigated if all benefits can be achieved through the implementation of a subset of principles, by implementing this future work it may be possible to uncover a smaller number of principles which can be implemented in order to receive the same benefits. These results could then be used to reduce the development time of a project should a smaller number of principles be recommended for optimum use. As Company X tailors its subset implementation of XP to the development team which is utilising the development method, it may be possible to carry out this study on a different software development team within the organisation. This would again provide the company with the opportunity to optimise its development method by comparing the results of future work to the results of this study.

Finally, as detailed in the literature review agile methods are a collection of software development models. Therefore, future work could investigate the implementation of a subset of XP principles in parallel with common characteristics of other agile methods such as SCRUM or Crystal. This future work could investigate if using a combination of agile methods could maximise the benefits which have already been reported within this study. Again, as the development method undertaken by Company X cannot be modified within the current software development this could not be undertaken in this instance. It is, however, possible that this investigation could be utilised within an experimental based study of two projects with the same functionality. This would provide a valid comparison and the results could then be presented to Company X should the benefits experienced be maximised through this implementation.

5.5 Conclusions

This study has investigated the use of implementing a subset of XP principles within a large scale software development in order to investigate if tangible benefits related to the full implementation of XP can be achieved. The study has involved a case study of Company X, a large scale organisation whose I.T. department develop software applications for use within the organisation. The case study has involved metric based calculations on code listings, interviews with developers and customer representatives and the administering of a questionnaire based on project risk to uncover if Company X has achieved substantial benefits in implementing a subset of XP principles in comparison to one of the organisation's traditional developments. The subset of principles implemented has included the planning game, small releases, simple design, testing, refactoring, collective ownership, continuous integration, 40-hour week and coding standards. The 3 principles not utilised were the metaphor, pair programming and on-site customer principles.

The results have indicated that substantial benefits have been achieved as all of the benefits reportedly gained through a full XP implementation have been experienced by the subset implementation. The results also suggest that the benefits associated with the implementation of the 3 omitted principles can be obtained through the implementation of the planning game, small releases, simple design, testing, refactoring, collective ownership, continuous integration and coding standards principles.

Although all benefits measured have been reportedly gained through the implementation of a subset of principles, the occurrence of some benefits has been substantial. The most notable of these benefits include a 40% reduction in the coupling of code and a 22% reduction in the complexity of code. Both of these measures are critical indicators of code quality and indicate that these benefits have been gained significantly. Further notable benefits include an increase in code robustness of 11% and a decrease in project risk of almost 7%. These measurements suggest a reduced possibility of project failure as the system is now more robust. As discussed previously, the findings of this study challenge the findings of Ambler (2000a) and Turk et al (2005) who report that all principles of XP need to be implemented for success. As the results of this study indicate that the benefits of a full implementation can be achieved through a subset implementation this supports the work of Copeland (2001) and English (2002a).

It is envisaged that the findings of this project will be beneficial to Company X as they provide an overview and comparison of the two development methodologies which it implements and the associated success of these. From the findings of the study it may be possible for Company X to review areas of their software development process and consider implementing its agile approach across a wider scale within the I.T. department. Furthermore, the results of this study will also prove beneficial to those within the software engineering industry who may currently be experiencing problems associated with the implementation of a traditional development model. As the results show that all XP principles do not need to be implemented for success, an organisation in this position may find it useful to utilise the results of this study to alter their own development approach in order to minimise the problems which they are currently experiencing.

Finally, the results of this study not only suggest that a subset of XP principles can be implemented to produce high levels of project success, but also that by following this implementation an organisation can address the current problems within the software engineering industry. As this project has reported that Company X has experienced these problems previously, including projects not being delivered on time or to the customer specification, the results indicate that these problems may have been eliminated through the implementation of a subset of XP principles and thus could be used within the wider software engineering industry as a way to tackle these problems. The findings also suggest that organisations may benefit from a phased approach to the introduction of agile techniques when considering changes aimed at improving their software development process.

Appendix A: Reference List

Alshayeb, M. & Li, W. 2006, "An empirical study of relationships among extreme programming engineering activities", *Information and Software Technology,* vol. 48, no. 11, pp. 1068.

Ambler, S.W. 2000a, “Adopting extreme programming”, *Computing Canada,* vol. 26, no. 8, pp.26.

Ambler, S. 2000b, "The extreme programming software process explained", *Computing Canada*, vol. 26, no. 5, pp. 24.

Ambler, S.W. 2001, "Debunking eXtreme programming myths", *Computing Canada,* vol. 27, no. 25, pp. 13.

Ambler, S. 2002, "Agile development best dealt with in small groups", *Computing Canada,* vol. 28, no. 9, pp. 9.

Babbie, E., Halley, F., Zaino, J., 2007, “Adventures in Social Research: Data Analysis using SPSS 14.0 and 15.0 for Windows”, 6th Edition, Pineforge Press: California.

Balsamo, S., Marco, A.D., Inverardi, P. & Simeoni, M. 2004, "Model-Based Performance Prediction in Software Development: A Survey", *IEEE Transactions on Software Engineering,* vol. 30, no. 5, pp. 295.

Baskerville, R., Ramesh, B., Levine, L., Pries-Heje, J., 2006, “High-Speed Software Development Practices: What Works, What Doesn’t”, *IT Professional Magazine,* vol. 8, no. 4, pp.29

Beck, K., 2000, “Extreme Programming Explained”, 1st Edition, Addison Wesley: Reading MA.

Biggs, M. 2000, "Pair programming: Development times two", *InfoWorld,* vol. 22, no. 30, pp. 62.

Biggs, M. 2001, "Profit from programming practices", *InfoWorld,* vol. 23, no. 25, pp. 64.

Boehm, B. 2000, "Safe and simple software cost analysis", *IEEE Software,* vol. 17, no. 5, pp. 14.

Cao L., Mohan, K., Peng X., Ramesh, B., 2004, "How extreme does extreme programming have to be? Adapting XP practices to large-scale projects," *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference*, pp. 10 pp.-, 5-8 Jan.

Cao, L. & Ramesh, B. 2007, "Agile Software Development: Ad Hoc Practices or Sound Principles?", *IT Professional Magazine*, vol. 9, no. 2, pp. 41.

Capiluppi, A., Fernandez-Ramil, J., Higman, J., Sharp, H.C., Smith, N., 2007, “An Empirical Study of the Evolution of an Agile-Developed Software System”, *Software Engineering, 2007. ICSE 2007. 29th International Conference*, pp.551-518, 20-26 May 2007

CCCC, [online], Available from World Wide Web: http://sourceforge.net/projects/cccc [last accessed 18th April, 2008].

Churbuck, D.C. 1992, "Learning by Example", *Forbes*, vol. 149, no. 12, pp. 130.

Code Analyzer, [online], Available from World Wide Web: http://www.geocities.com/sivaram_subr/codeanalyzer/description.htm [last accessed 18th April, 2008].

Cohen, B. 2005, "Agile Programming Not For the Risk Averse", *InformationWeek*, no. 1052, pp. 74.

Cohn, M., Ford, D., 2003, “Introducing an agile process to an organization [software development]”, C*omputer*, vol.36, no.6, pp.74-78, June 2003.

Company X, 2004, “ICRS – System Architecture & Development”.

Company X, 2006, “Development Process”.

Conrad, B. 2000, "Taking programming to the extreme edge", *InfoWorld*, vol. 22, no. 30, pp. 61.

Conte, S.D., Dunsmore, H.E., Shen, V.Y., 1986, "Software Engineering Metrics and Models", Benjamin/Cummings Publishing Company Inc: California.

Copeland, L., 2001, “Developers approach extreme programming with caution”, *Computerworld*, vol. 35, no. 43, pp.7.

Copeland, L. 2002, "Caterpillar digs into agile development", *Computerworld*, vol. 36, no. 2, pp. 14.

Corbin, D.S. 1991, "Establishing the Software Development Environment", *Journal of Systems Management*, vol. 42, no. 9, pp. 28.

Dewey, R.H. 1988, “Software Engineering Technology and Management”, *SRI International, Report No. 762 1988*, pp. 2.

Dybå, T., Arisholm, E., Sjøberg, D.I.K., Hannay, J.E. & Shull, F. 2007, “Are Two Heads Better than One? On the Effectiveness of Pair Programming”, *IEEE Software,* vol. 24, no. 6, pp.12.

English, A. 2002a, "Extreme Programming: It's Worth a Look", *IT Professional Magazine,* vol. 4, no. 3, pp. 48.

English, A. 2002b, "Extreme programming", *Network World*, vol. 19, no. 4, pp. 60.

Erdogmus, H. 2007, "What's Good Software, Anyway?", *IEEE Software*, vol. 24, no. 2, pp. 5.

"Extreme programming: The Zero G experience", 2003, *Technology review,* vol. 106, no. 9, pp. 38.

Fenton, Norman E., 1991, *“Software Metrics, A Rigorous Approach”,* 1st Edition, London, Chapman and Hall, p280.

Fitzgerald, B., Hartnett, G. & Conboy, K. 2006, "Customising agile methods to software practices at Intel Shannon", *European Journal of Information Systems,* vol. 15, no. 2, pp. 200.

Glass, R.L. 2001, "Extreme programming: The good, the bad, and the bottom line", *IEEE Software,* vol. 18, no. 6, pp. 112.

Glass, R.L. 2006, "The Standish report", *Association for Computing Machinery.Communications of the ACM,* vol. 49, no. 8, pp. 15.

Graham, D.I. 2004, “Eye on IT - Agile approach to software is the way forward. “, *The Belfast Newsletter*, Nov 2, 2004, p2

Griffin, M. 2002, "Extreme programming: An alternative life cycle model part 1", *Information Executive,* vol. 6, no. 5, pp. 7.

Guntamukkala, V., Wen, H.J. & Tarn, J.M. 2006, "An empirical study of selecting software development life cycle models", *Human Systems Management,* vol. 25, no. 4, pp. 265.

Highsmith, J. & Cockburn, A., 2001 "Agile software development: the business of innovation," *Computer*, vol.34, no.9, pp.120-127

Holmström, H., Fitzgerald, B., Ågerfalk, P.J. & Conchúir, E.Ó, 2006, "Agile Practices Reduce Distance in Global Software Development", *Information Systems Management,* vol. 23, no. 3, pp. 7.

Jiang, J.J., Klein, G. & Ellis, T.S. 2002, "A measure of software development risk", *Project Management Journal*, vol. 33, no. 3, pp. 30.

Kan, S.H., 1995, “Metrics and Models in Software Quality Engineering”, Addison-Wesley: Reading, Massachusetts.

KLOC, [online], Available from World Wide Web: http://www.analogx.com/contents/download/program/kloc.htm [last accessed, 18th April, 2008].

Krishnan, M.S., Mukhopadhyay, T., Zubrow, D. 1999, "Software Process Models and Project Performance", *Information Systems Frontiers*, vol. 1, no. 3, pp. 267.

Layman, L., Williams, L. & Cunningham, L. 2006, "Motivations and measurements in an agile case study", *Journal of Systems Architecture*, vol. 52, no. 11, pp. 654.

Levy, J.V. 2003, "If extreme programming is good management, what were we doing before?", *EDN*, vol. 48, no. 25, pp. 81.

Lindstrom, L. & Jeffries, R. 2004, "Extreme Programming and Agile Software Development Methodologies", *Information Systems Management*, vol. 21, no. 3, pp. 41.

LocMetrics, [online], Available from World Wide Web: http://www.locmetrics.com/ [accessed 22nd February, 2008].

Maurer, F., Martel, S., 2002a "Extreme programming. Rapid development for Web-based applications," *Internet Computing, IEEE* , vol.6, no.1, pp.86-90, Jan/Feb 2002.

Maurer, F., Martel, S., 2002b "On the Productivity of Agile Software Processes: An Industrial Case Study," [*online*] Available from World Wide Web: http://ase.cpsc.ucalgary.ca/ase/uploads/Publications/MaurerMartel2002c.pdf [accessed 13th December, 2007]

Maurer, F., Melnik, G., 2006 “Comparative analysis of job satisfaction in Agile and non-Agile Software Development Teams” *[online]*. Available from World Wide Web: http://ase.cpsc.ucalgary.ca/ase/uploads/Publications/XP2006_Melnik_Maurer.pdf [accessed 12th December, 2007].

McCabe, T.J., 1976, "A Complexity Measure," *Software Engineering, IEEE Transactions on,* vol.SE-2, no.4, pp. 308-320.

McDowell, C., Werner, L., Bullock, H.E., Fernald, J. 2006, "Pair programming improves student retention, confidence, and program quality", *Association for Computing Machinery.Communications of the ACM*, vol. 49, no. 8, pp. 90.

McMahon, J. 2003, "5 Lessons from Transitioning to eXtreme Programming", *Control Engineering,* vol. 50, no. 3, pp. 59.

Mens, T. & Tourwe, T. 2004, "A Survey of Software Refactoring", *IEEE Transactions on Software Engineering,* vol. 30, no. 2, pp. 126.

Mőller & Paulish, 1993, “Software Metrics, A Practitioner’s Guide to Improved Product Development” 1st Edition, Chapman & Hall: London.

Neill, C.J. & Gill, B. 2003, "Refactoring Reusable Business Components", *IT Professional Magazine,* vol. 5, no. 1, pp. 33.

Nord, R.L. & Tomayko, J.E. 2006, "Software Architecture-Centric Methods and Agile Development", *IEEE Software,* vol. 23, no. 2, pp. 47.

Oates, B.J., 2006, “Researching Information Systems and Computing”, London: Sage.

Rainsberger, J.B. 2007, "Rescuing Code", *IEEE Software,* vol. 24, no. 5, pp. 26.

Rasmusson, J. 2003, "Introducing XP into Greenfield projects: Lessons Learned", *IEEE Software,* vol. 20, no. 3, pp. 21.

Rosenberg. L.H, Hyatt. L.E, 1997, “*Software Quality Metrics for Object-Orientated Environments*”, Crosstalk, Journal of Defense Software Engineering, April 1997.

Rumpe B. & Schrder A., 2002 “Quantitative Survey on Extreme Programming Projects.” In: Third International Conference on Extreme Programming and Flexible Processes in Software Engineering, XP2002, May 26-30, Alghero, Italy, pg. 95-100.

Sandmeier, P. & Gassmann, O. 2006, "Extreme Innovation", *European Business Forum,* no. 26, pp. 44.

Sliwa, C. 2002, "Users warm up to agile programming", *Computerworld,* vol. 36, no. 12, pp. 8.

Sommerville, I. 1996, "Software process models", *ACM Computing Surveys,* vol. 28, no. 1, pp. 269.

Sommerville, I. 2004, "Software Engineering 7", 7th Edition, London: Addison Wesley, p7.

SourceMonitor, [online], Available from World Wide Web: http://www.campwoodsw.com/sourcemonitor.html [accessed 22nd February, 2008].

Standish Group International, 1994 *“The CHAOS Report (1994)” [online]*. Available from World Wide Web: http://www.standishgroup.com/sample_research/chaos_1994_1.php [accessed 17th October, 2007].

Talby, D., Hazzan, O., Dubinsky, Y. & Keren, A. 2006, "Agile Software Testing in a Large-Scale Project", *IEEE Software*, vol. 23, no. 4, pp. 30.

Thomas, D. 2005, "Agile Programming: Design to Accommodate Change", *IEEE Software*, vol. 22, no. 3, pp. 14.

Turk, D., France, R. & Rumpe, B. 2005, "Assumptions Underlying Agile Software-Development Processes", *Journal of Database Management*, vol. 16, no. 4, pp. 62.

Williams, L.A. & Kessler, R.R. 2000, "All I really need to know about pair programming I learned in kindergarten", *Association for Computing Machinery.Communications of the ACM*, vol. 43, no. 5, pp. 108.

Williams, L., Kessler, R.R., Cunningham, W. & Jeffries, R. 2000, "Strengthening the case for pair programming", *IEEE Software*, vol. 17, no. 4, pp. 19.

Williams, L., Krebs, W., Layman, L., 2004, “Extreme Programming Evaluation Framework for Object Orientated Languages”, version 1.4, *[online]*. Available from World Wide Web: http://collaboration.csc.ncsu.edu/laurie/publications.html [accessed 23rd November, 2007].

Williams, L., Maximilien, E.M., Vouk, M., 2003 "Test-driven development as a defect-reduction practice," *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on* , vol., no., pp. 34-45, 17-20 Nov. 2003

Willoughby, M. 2005, "Coder Be Agile, Coder Be Quick", *Computerworld*, vol. 39, no. 30, pp. 36.

Wood, W.A. & Kleb, W.L. 2003, "Exploring XP for scientific research", *IEEE Software*, vol. 20, no. 3, pp. 30.

Yin, R.K, 2003, “Case Study Research: Design and Methods”, London: Sage.

Zhang, Y. 2004, "Test-driven modeling for model-driven development", *IEEE Software*, vol. 21, no. 5, pp. 80.

Appendix B: Proposed Interview Questions

Appendix B details the proposed interview questions for each of the interview participants, traditional developer, agile developer and customer representative. The list below details where these can be referenced:

Appendix B1: Proposed Traditional Developer Questions

The headings in bold above each question relates to the benefit which the question is investigating.

Software Engineering Problems/Requirements Being Met:

1. Was the project on time? Budget? To specification?

Reduction in Planning Time:

2. Did it take a long time to initially plan the system? (requirements documents)

Regular Contact/Rapid Feedback:

3. Did you have regular contact with the customer? Reviews? How often?

Relation to Project:

4. Did you have an overall view of the system at the start of development?
5. Did you understand what the project was doing?
6. Could you relate to the project and your part within it?

Communication within Team/Knowledge Spread throughout the Team:

7. Was there good communication throughout the project team? (developers, customers) could this have been improved? Did you learn from other developers?

Easy To Implement Changes/Project Dynamism:

8. Was it easy to implement changes in the system imposed by the customer? Was it a lengthy process? Were there a lot of changes?

Simplify Integration Process:

9. Was it easy to integrate the whole system – timely? Errors propagate?

Satisfaction:

10. Were you satisfied with the system? Confident? Motivated?
11. Was the customer satisfied?

Resolve Issues/Set Priorities:

12. How were problems solved? Priorities set? Easy?

Metric questions:

1. How many errors have occurred in the system?

2. What stage in the project life cycle do they mainly occur?
3. How long does it take to fix errors?
4. How long did it take to develop the system? Time per class?
5. How many requirements have been added/removed? Change requests?

Appendix B2: Proposed Agile Developer Questions

The headings in bold above each question relates to the benefit which the question is investigating.

Reduction in Planning Time:

1. Is planning time reduced in comparison to a traditional system (no requirements documents)

Regular Contact/Rapid Feedback:

2. Did you have regular customer reviews for feedback?

Requirements Being Met:

3. Did the system meet the requirements?

Relation to Project:

4. Did you have an overall view of the system at the start of development and what the project was doing?
5. Could you relate to the project and your part within it?

Communication within Team/Knowledge spread throughout Team:

6. Has communication improved throughout the project team (developers, customers)?
7. Is the knowledge of the system spread throughout the team? (Could anyone work on one part and understand it) – would that reduce training?

Easy To Implement Changes:

8. Is it easier to implement changes because of incremental development?
9. Is there less change in the project overall?

Satisfaction

10. Has agile increased your job satisfaction/confidence? Customer satisfied?
11. Does it make you more motivated/stimulated?

Simplify Integration Process:

12. Is the integration process simpler than in traditional projects? Less time? Less errors?

Resolve Issues/Set Priorities:

13. Is it easier to resolve issues/set priorities with the customer?

Metric questions:

1. How many errors have occurred in the system? Fewer than traditional development?
2. What stage in the project life cycle do errors mainly occur?
3. How long does it take to fix errors?
4. How long did it take to develop the system? Time per class?
5. How many requirements have been added/removed? Change requests?

Appendix B3: Proposed Customer Questions

The headings in bold above each question relates to the benefit which the question is investigating.

Satisfaction:

1. Are you satisfied with the developed system? Was it on time, budget?

Requirements met:

2. Does the system meet the requirements you specified? Does it do what it needs to?

Regular Contact/Opportunity for giving Feedback:

3. Did you have a chance to see the system regularly during development?

Relation to Project/Communication throughout Team:

4. Did you understand what the project was about? What system was being built and what it would do?
5. Did you feel you could relate to the project – did you know your role in the project? Could you easily communicate any changes needed?

Appendix C: Interview Transcripts

Appendix C details the interview transcripts for each of the interview participants, traditional developers, agile developers and customer representatives. The list below details where these can be referenced:

Appendix C1: Traditional Developer Interview Transcript (1)

Was the project on time? Budget? To specification?

It was yes. The system was legislative so it had to be implemented on time. We don't have a budget structure due to the organisation so at that time it wasn't budgeted.

Would you say it met the specification?

The initial release had some bits missing, so it was a basic system because of the legislative requirement. So the system did all the basics but didn't have search facilities etc these were added on late once released.

How long would you say the planning stage took?

The basic system about 2 weeks to a month.

How did you do that? Did you go and speak to the customer?

There was a representative in charge and I worked with her to find out what they do every day and how the system could help them. To give you the exact details that are required, not necessarily from an IT perspective.

Was that quite helpful?

It depends on the customer. I've had some really good ones and some really bad ones. A lot of the time they don't realise their role in the project. At that time the customer was very helpful.

Can it be difficult at times to get help?

Yes, but it depends on the customer.

How often throughout the project did you have contact with the customer?

We could have had daily contact by telephone calls. But the office was in the same building at that time so we often had informal meetings when I needed to really.

Did you have scheduled meetings?

The important things required formal meetings for example when an outside organisation was involved we had a formal scheduled minuted meeting. But within this organisation the meetings tended to be informal.

Were the meetings only when problems came up?

Yes and also when questions came up. I didn't know the application area well so had a lot of questions.

Did the customer see the system at any point before it was released?

Yes, about one week before. The operators had to be trained too so they used the system to train that. I was involved in the training because the system had to be working correctly by a certain date because it was legislative.

Was that when you got feedback on the system or did you get feedback earlier than this?

Yes this was when we got feedback on the system because it was the first time that the customer could see the system.

When you started development did you have an overall view of what the system would do or did you learn as you went along?

I had an overall view of the basics but it was only when the system went live that the people who were using the system started to ask for more. I also didn't really know a lot about what the system was doing, the actual business logic of it, so I had quite a few questions.

So at the start would you say you understood the project?

Not 100% no, but as I went on and became more familiar with the business logic of things I understood it more.

Was that because they thought of other things that could be added or because things were missing?

It was probably because things were missing like reports and things. But the main thing is when they said someone is asking for this information but we cant get it can you give them a report for it. So it wasn't until they went to do things that they realised that other parts would have been helpful.

Could you relate to the project and did you know your part within it?

Yes I did understand what the project was doing and that I was a developer in the project but I would say at the beginning I couldn't exactly relate to all the functionality because I didn't know everything that would be implemented inside out because I didn't really know the inside workings of the department that the system would serve.

Do you think there was good communication between the team?

There were three of us and the communication was pretty good, every day face to face talking. John was good at coming up with ideas.

What about the communication with the customer? Were they easy to get a hold of?

Well the customer was very apprehensive then, she had never used a computer before to do this job. You don't get that now though, everyone has used a PC. It was easy to get to the user though. Initially I spent the full day with them. The user was also quite hostile for the first few weeks and she was picking up every little fault.

Did you learn from the other developers?

Yes.

Do you think all 3 of you knew what each part of the system was doing?

I could have worked on any part but the other two had a minimal input so they probably couldn't.

So would you say the knowledge of the whole system wasn't spread throughout the team then?

No I don't think it was. I don't think we knew enough about each others code or functionality.

When the customer wanted changes were they easy to make?

They weren't difficult. The legislation has since changed again though so some changes were quite major and timely. The technical complexity was quite difficult it was the size and time consuming. Also, it was quite high profile.

Overall has the system had a lot of changes?

Yes, they are mainly legislative though. But also some major changes implemented. It was introduced 5 years ago so some major changes have taken place since.

At the end was it difficult to integrate everything into one system?

No, I had been doing that as I went along so it was just a case of putting the test stuff in the live environment so it didn't take a lot of time.

Did you find that there were a lot of errors when you integrated the system?

I have to confess yes! We had to put it in 7 sites and didn't have any remote access so had to go to the sites if problems occurred.

How many errors have occurred in the system?

They are still happening to this day, the error log has 8554 but these can be minor things like an index at zero and these were counted from day one of development.

What stage to do most errors occur?

After release but it eventually stabilised. But the longer it has been released the less errors have occurred.

Do you think that is because people have become more used to the system?

We had a small time period to develop the system so a lot of the parts had been missed out and there was a lot of post installation work carried out to get it just right so these also caused errors.

How long does it take to fix errors that occurred?

I would say a few hours rather than days, possibly less. It depends on the errors. Also, because of the proximity of the users then changes could be implemented immediately.

How many changes has there been?

Well over 100.

How long did the development take?

The basics took two months which had to be implemented by the 1st of April for legislation but after that we could sit back and look at the system and see then what we could change and expand with regards to extra functionality. So overall it was probably six months. But the basic parts we had to have in were written in two months. We had to do a lot of overtime to get it ready though.

How long would you say it took to write a class?

They weren't big but they were complex, but it would take less than a day to write them.
Other things were easy though.

Was the customer satisfied at release?

I would say 75% satisfied.

Did you have any problems trying to resolve issues with them?

No, we didn't really have issues with them because they didn't see the system until it was complete. We tended to do what we wanted.

Was it easy to set priorities with the customer?

We set the priorities ourselves because we were developing the system.

Were you satisfied with the system?

Not when it was released no, but once it matured yes.

Would you say that effected your job satisfaction?

Yes probably because I wasn't happy with what had been developed.

Were you confident in the system and its development?

Not when it was released.

Were you confident in your part of the project?

Not really no, because I didn't manage to get the full system ready in time.

Did you feel motivated by the project?

Not motivated, but pressurised.

What training did you need when you started the development?

I didn't need any training because I was familiar with the software but we had a meeting with the customer's department to discuss how they do their job and the reasoning behind some of the complex requirements.

Did you learn from the other people?

I did ask other people yes, but probably not within the team. Some people were more experienced than me so I did ask them.

Appendix C2: Traditional Developer Interview Transcript (2)

How many people were working on the project?

2 developers mainly, sometimes three.

Was the project delivered on time?

Yes.

Was it to the specification?

When first release all the functionality wasn't enabled because we had to implement the system in line with legislation. But after the system was released we changed it to add the missing parts so then it did meet the specification but the specification has changed a great deal now.

Was there a budget defined?

No, we don't have a budget structure in place because we are an in house development team.

How long did it take to plan the system at the start?

Initially, about one month (ish). The customer was involved in meetings during the planning stage so we knew what requirements they wanted.

During development did you have regular meetings with the customer?

No, only if further clarification was needed. The customer only saw the system when it went to a pilot before release. But if there were any problems we could easily phone and ask the customer.

Did you get feedback on the system throughout development?

We got feedback on requirements and changes to requirements but not the actual system because the customer didn't see the system until nearer the release date at the pilot stage.

At the start of development did you have an overall view of what every part would do?

We probably learned as we went along because of natural changes in the requirements and through gaining experience as we went on.

Did you understand what the project was doing at the start of the development?

No. At the very start of the project the customer didn't really know what the system had to do so we found it hard to get requirements from them.

Could you relate to the project and your part within it?

Yes, I knew what parts I would be responsible for but it was difficult because the customer didn't know what they wanted so we didn't know everything that would be implemented and so I didn't know absolutely everything that I would be responsible for.

Was there good communication within the project team?

Yes, we discussed the project everyday with regards to how we were progressing. But the communication with the customer could have been better, we didn't talk to the customer too much and to be honest I think with hindsight that was a mistake.

Was the customer quite helpful?

Yes they were fine when you needed help and clarified problem areas. We could email or call them.

Do you think you learned from the other developers?

No. the other person didn't understand the code or functionality I was working on so we couldn't discuss any problems that I had or work to solve them together.

Do you think the knowledge of the system was spread throughout the team?

To an extent. I think I was aware of the work the other developers were doing but that was not always the case for the other developers. I don't think they fully knew what functionality my code did.

Do you think it was easy to implement changes from the customer?

Yes it was quite easy to make changes during the development, and probably afterwards too. The changes from the customer were mainly small things but some changes involved quite difficult business logic so that took a bit longer than usual.

Were there a lot of changes in the project?

Yes there were a lot of changes throughout the whole project. An average amount would probably be about 20.

Was the overall system integration easy?

Yes, there were a few errors but not too many due to the testing that had been performed on it. The way we work we build the system as we go along so there was no real bringing together of the system. It was basically just making it go live so it wasn't too time consuming.

How many errors would you say have occurred within the system?

Less than one hundred. You would have to check the error log though.

What stage in the life cycle do most errors occur?

Mainly after the release of the system, it is generally the case that the more the customer uses it the more bugs are found.

How long did it take to fix errors?

Some errors are simple and don't take too long, maybe 15 minutes work. Others may be more complicated and take a couple of days. It just depends on the errors but as an average I would say a few hours.

How long did it take to develop the system?

Probably altogether about 6 months.

How long would you say it would take to write a class?

Definitely less than a day, they are mostly quite small, probably an hour.

Do you think the customer was satisfied with the end result?

I believe so. They didn't ask for too many changes after implementation so that tends to suggest they are relatively happy.

Did you find it difficult to resolve any issues with the customer?

No they were quite helpful throughout and relatively easy to get a hold of although some of the changes they wanted were too late to be implemented because they didn't tell us until the pilot stage.

Did you set priorities with the customer?

No, we set them ourselves.

Were you satisfied with the system?

At the time when released I believe I was yes, but now I wouldn't be, from experience I think why did we do things that way?

Would you say that you had a high level of job satisfaction?

I was satisfied that the system had been released but I wasn't satisfied in the way it has been developed. I think we could have worked as a team more.

Do you think you would have been more confident if you had developed the system in a different way?

Possibly, but you only gain that from experience. I wasn't confident in our system though.

Do you think you were motivated by the project?

At the start yes, but not at the end.

What type of training did you need before development?

We didn’t go on any courses for it, we just learnt as we went along by ourselves. We basically used other peoples code as a learning base and manipulated it for our purposes. We did get training from the department on what they do with regards to the business logic of the requirements but no specific training on the development method.

How many changes would you say have been requested?

Probably around 25, somewhere around that region.

Appendix C3: Agile Developer Interview Transcript (1)

How long would you say it takes to plan the system when you are using agile?

Agile planning is easier to plan because you don't plan 100% you take the project as a whole and break it up into iterations.

Do you do you planning throughout then rather than at the very start?

Yes, because the project can change, each iteration is done in depth then after you release the first iteration you then need to know a bit more so that effects the plan being changed.

So do you think it takes less time then to plan than traditionally?

Yes because the first thing is to break the project into iterations and then divide the work into the sections, that way it is quicker. Then you tend to deliver the system in shorter iterations and then you can assess what needs to be done next and put into the next iteration.

How do you decide what is going to be in an iteration? Do you talk to the customer and decide or decide as a team in the organisation?

We talk to the customer roughly to get the first iterations (4 weeks). We create a document like a case study of what will be in an iteration so the user can understand what they are going to get in an iteration. We use this to implement iterations. Through use cases the user knows what they are going to get in detail.

How often do you get feedback from the customer? Do you have regular meetings and reviews?

Yes, before each iteration is released we get the iteration reviewed, this depends on the customer being ready so we have to rely on them.

If you deploy without the customer seeing the system they come back and say oh I didn't want that and that is right, so for that reason we finish an iteration then speak to the user and say here it is.

How long is an iteration?

Ideally 4-6 weeks but it depends on the system, its quite flexible.

Do you think that's a quicker way to get feedback than traditionally?

Yes, because that way you are putting the user in the spotlight all the time and you get information out of them all the time. Its not like a traditional way when you would do analysis and design and after a year give them a system and they have forgotten what they asked for so this way the user is involved more with feedback.

So do they give you more feedback this way?

Yes, its more fresh in their mind and the feedback is more to the point because of the iterations.

Do you think the system meets the requirements?

It does meet the requirements but because of the iterative development if there are extra things that have been missed these can be added to another iteration.

When you first started to develop the system did you have an overall view of the system?

Yes definitely, you need to know that before you plan and break it into iterations.

So you didn't learn throughout the project parts that you didn't know were going to be included?

No, you have to know, you have to know the whole system because otherwise you don't know where you end up. You do your best to get as much information as you can at the start.

So would you say you understood the whole project at the start of development?

Yes definitely, that's really important.

Could you relate to the project? Did you know your part within it?

Yes.

Do you think the communication in the team is improved with agile?

Yes because that makes the developers work harder to work towards deadlines every month and it forces them to communicate because they don't have a lot of time. Its not like they can wait until something happens so it keeps them on their toes and forces them to communicate so they can get everything done and ask all the questions that they need to. It makes the development team productive.

Is the communication better with the customer?

Yes it is better, but it really depends on the customer.

How do you communicate with the customer?

Phone them, email them, bring them in to show them things. It is a lot easier nowadays though to send them a URL through an email and they can view it there and get back to you with feedback. When something is near completion I send an email with a prototype version and test data base so the user can give feedback on it. Its not a prototype it's a developed version of it before it gets deployed it is the actual thing. The front end is the same. Then they say yeah that's it or we make small changes. We do it with an aim that the iteration is *complete 90-95% so we don't expect any major changes, we allow for a little bit of tweaking and for small changes because that happens.*

Did the customer come back with a lot of changes?

Not really no, one thing about iterative development is that you develop vertically, you split the system into slices so you know whats in an iteration so they dont go beyond the boundary, we talk about whats in your mind just now. It lets the user control what they want.

Do you think by using iterations when the customer asks for changes it makes it easier to do because your only working on a small bit of the system?

Yes, it is a lot easier because you don't need to change a lot to make the change. But the most important part is that you have to communicate to the user because the user doesn't know what agile development is. You need to explain to them that they are getting whatever they want within that context. So that way you can always accommodate change.

Overall in the project do you think there is less change as there is with traditional?

Oh yes because the customer is more aware of what is going to be developed in each iteration.

Do you think that changes are much smaller to implement?

Yes they are smaller and not as complicated.

Do you think the knowledge of the system is spread throughout the team? Could you work on another's work etc?

Yes we do that if we have problems yes.

Do you think that reduces the need for training?

Oh definitely, we help each other sometimes if there is a complex part or problem and we give each others view, like swapping work. That always resolves in the person knowing more about the system than in the traditional way when you give them a big chunk of the system and you don't know what it does. It definitely does reduce training.

Do you think agile has made your job easier?

Well, it has its downsides too. The downside is that it is time consuming to write tests, but in the future these would help if you were redeveloping the system because you already have the tests. So its useful in that way. But it is time consuming.

Do you think it has increased your job satisfaction?

It does keep you busy yes, you do communicate more, it keeps you busy and you see the results of your work more quickly because you do development more often than the traditional way so that way it is satisfying to see things up and running a lot sooner than you would expect. Its like when the system was split into iterations, when the user came in and I *explained what they were going to get first, they knew what they were going to get first and he was under the impression that he would get it in a few months but it has actually already been deployed and the customer was surprised, so that is encouraging.*

Do you think the customer was surprised because he was used to seeing development traditionally?

Yeah, he thought we would be spending months and months and it would be a couple of years down the line so he was surprised to know I had it up and running already.

Do you think that this increased the customers satisfaction?

Oh definitely, because he was seeing the system working and knew he was being kept up to date I think he was really happy with the system.

So do you think that has increased your confidence?

Oh yeah, because if you know the customer is happy then you know you are doing it right.

So for the whole team it makes you more motivated then?

Yes its more motivating because you see progress but the downside is not just the testing but also the user. In the organisation the user is very slow to respond. So if you plan something for 4 weeks and you cannot get a hold of the user for 2 months its makes a mockery of your plans. We had split the system into 6 iterations and had only deployed the first 4 because we couldn't find the customer anywhere. Can you imagine how annoying that was? So that causes a delay in this company. Another thing that is good is that because of the changing working environment it can cope with it because you develop as things are changing and with a traditional way you would have a system written for 2 years for a spec that was written 2 years ago and in that period of time things have changed. That happened in an older system for example. Because it was such a massive system it took 3-4 years to develop and parts of it are not used because working practices have now changed and that is massive disadvantage.

But agile accommodates that?

Oh yes, agile is in time – just in time development – just give them what they want now. But if something happens two years down the line we have written all the tests so even then it would be quicker to change.

So even long term agile is more accommodating?

Oh yeah.

So because of the small iterations do you think when the whole system is put together it is a lot simpler?

Yes.

Do you think it takes less time?

Oh yeah, that's why we do testing so when we are developing iterations we run all the tests again before integration so there are less errors too.

Do you think it is easier to resolve issues with the customer?

Yes most of the time. Yes because you can change you can always offer an alternative to the user too. But the users sometime only see things one way and that can result in a system failure. If you have an experienced developer who can resolve the system just gets exhausted, its full of bugs and doesn't do what its meant to and doesn't work. You must communicate with the user and offer alternatives to make the system work.

Was it easier to set priorities with the customer?

Oh yes definitely because we discussed what should be in each iteration when planning it before an iteration started so we could see what was most important to the customer then because it may have changed from when the project started.

How many errors occur with agile compared to traditional?

Its definitely less, it is less yes. Because you do lots of testing and write in small chunks of work.

What stage in the life cycle do the errors mainly occur?

Well, it happens in development as well as when the system in finished. When the system is released it tends to be the business logic that is wrong from time to time so most coding errors are found during testing.

How long would you say it takes to fix errors?

It depends on the errors but because the errors are smaller it is generally less, usually in hours rather than days. But with traditional it could take days or a week. The problem with agile is that when you fix an error there is always the prospect that when you run the tests more will occur.

How many people worked on the agile system?

There were only 2-3 people during development. We use that as a guideline.

How long did the system take to develop?

It took about 4 months, with iterations of 4-6 weeks.

So that's quite a bit shorter then than with traditional?

Oh yeah it would have been because with traditional you spend time to do more ground work to start with.

Has there been less changes for the agile project than the traditional?

Yes, yes.

Significantly less?

Well yes it is because you actually deal with the changes while your developing the iterations so it doesn't go through a formal process to make changes.

Have you had many changes?

No, oh yes, yes, we have had 2.

How many would you say a traditional project would have?

Oh many, many. It depends on the system but my last project had 45. So that's a big difference.

How much training did you need before you started agile?

Well we didn't need any because we started it ourselves but we have been on a training course two weeks ago and then I have changed some of my practices. But I don't think it is necessary if you work under a developer that has that experience then you don't need to because you could learn from them and it could be training on the job. But if you don't know anything about it then you should get some training.

How long have you been developing?

18 years. 9 of them here.

Additional comments:

Users get flabbergasted when they are given a piece of functionality then when they get used to it and it they understand it the want more changes.
Users still work the same way though, the way they know best. Even if they get a new system they don't change their working methods.

Appendix C4: Agile Developer Interview Transcript (2)

How many people were in the development team?

2, at times three.

How long would you say it took to plan the system?

It was quite short in comparison to a traditional project. We still made a form of a requirements documents but it focused on use cases. We spoke to the customer to generate the use cases, agile is probably more customer focused than traditional projects because you can go back to the customer for feedback.

Did you have regular meetings with the customer throughout the development?

Yes, we didn't really have set meeting times every week though we just met as and when required. A lot of the time we just phoned or emailed the customer when we needed to, like when confirming requirements.

Did the customer get to see the system before it was released?

Yes, they saw the system at the end of each iteration before it was signed off, probably every 4 weeks. Throughout the development that was quite a lot that they saw them so they could review the system throughout.

Was that when you got your feedback from the customer?

Yes, the customer gave us a list of points after each review that could be changed or enhanced, but these were usually only cosmetic things.

Do you think the system meets the requirements of the user?

Yes because we could incorporate changes they wanted throughout its development so it was adapting while being developed.

Do you think the customer was satisfied with the system?

Yes they were. There were little changes to be implemented because they had been incorporated into the development.

Did you have an overall view of the system at the beginning of the development?

Yes, I knew all the different parts of the system from the use cases we generated and I knew which of these I would be responsible for so yes I did have a view of the system at the beginning that I could work towards.

So would you say you understood the project from the beginning?

Definitely, that's quite important in an agile project because we have to implement the most difficult aspect of the requirements first so we had to understand everything before we could get started.

And could you relate to the project and your part within it?

Yes, I think so. Because we discussed everything at the beginning I was aware of what I would be doing in terms of being a developer, but also had a rough outline of what types of functionality I was expected to implement.

Do you think that communication has improved?

Yes, I could have easily went to the other developer for help if I was having any problems. Even the communication with the customer was very good because at the very beginning we explained to the customer that this was the approach that we were taking so they knew that they were important to the development of the system and that they had to get involved with the project.

So do you think the knowledge of the system was spread throughout the team then?

Yes, we both knew what each other was working on and could have worked on each others code no problem.

Do you think it was easier to implement changes in the system?

Yes it was, because the system was developed in iterations so any changes were only implemented into one part of the system so it didn't take as long and it wasn't as complicated. It also meant that the tests had already been written so these could easily have been run to make sure the change didn't cause any more bugs, so I was confident that the system still worked after implementing a change.

Do you think that there is less change in an agile project?

I'm not too sure, they are probably about the same. But I do think that agile is able to accommodate the changes better. The changes are really dependant on your understanding of what the requirements are and also the users expectations about what they are receiving. Although, the customer is more aware of the system and what each iteration will develop so they are given more opportunity to specify requirements so I can see how it might reduce the number of changes but I've never really thought about it.

Do you think the changes were smaller or easier to make than in traditional projects?

Well the changes were both to the functionality and minor changes like the layout of the screen, sometimes its difficult for the users to picture the system. They know what they want but they cant envisage that in a system because they are not necessarily IT literate so once they see the developed system they quite often come up with other ideas or they see that its not quite working how they envisaged it would. Some of the changes were to accommodate their process better.

Do you think that agile has made your job easier?

Not any easier no. In some ways it might have made it more difficult because it is more difficult to manage. Sometimes when changes are implemented it changes the plan of an iteration and that can be difficult to manage.

Would you say agile has increased your job satisfaction?

I don't think so, not between traditional development and the agile approach. Although I suppose in a way perhaps because you can see development and progress and it makes the customer happier. Yeah I can see how that would make some people more satisfied.

Do you think agile has increased your confidence?

Yes, it might have. Probably because I was getting feedback from the customer and I knew I was making a system that they wanted, yeah I would say I was more confident.

Do you think it makes you more motivated?

Yes, I think because you can see the system developing it encourages you to keep moving on with the project and put more work in to get it finished.

Was the integration process simpler?

Yes, because I had all the iterations and the tests could be run on the integrated system so less errors occurred.

Were problems with the customer easy to resolve?

Yes they were very easy to deal with. Probably because from their point of view they were seeing development early and were quite happy, quite pleased to see progress being made. That's one of the big advantages of agile, instead of the customer giving the requirements and two years down the line the customer getting the system and its not what they want.

Could you set priorities with them easily?

Yes, the customer was very helpful throughout the development. I think it was beneficial to let them know the process we were taking in advance so that they knew what would be expected of them.

Do you think agile makes less errors occur in a project?

I suppose it does yes because of all the tests that are written and because you can run these at any time.

What stage in the lifecycle would you say that errors mainly occur?

They can occur at any time really. But probably most are found at the testing stage, before its released. There is no way you can eliminate all bugs though because you can never cover *every aspect. In my experience you still throw up some bugs, but there is definitely less than in a traditional project.*

Do you think the errors were easy to fix?

Errors were mainly in the presentation side not really the functionality so they didn't take long to fix, they were quite minor so probably an hour or less.

How long did it take to develop the system?

I think about 4 Months

Were there any major changes throughout the development?

No there wasn't any major changes but there were 2 change requests which were minor.

Did you need any training before developing the system?

The system was my training! I basically learned from the other developer and by myself by developing the system.

Appendix C5: Traditional Customer Representative Interview Transcript

Are you satisfied with the developed system?

Overall yes but probably not when the system was first released.

Was that because parts of the system were missing?

Yes, I was told the whole system would be delivered at once but it wasn't. It didn't really stop us doing our job because it only missed out reports and minor things but it was an upheaval for us again to get it updated a few months later.

So does the system meet the requirements you specified?

The system did meet the requirements once it was fully implemented but not in its first release because it had those things missing, but things have changed over time and I would say it probably doesn't meet the requirements now.

Does it do what it needs to?

Yes it does everything we need it to but there are ways that it could be done simpler now which would make things faster and easier for us.

Did you have a chance to see the system regularly during development?

Yes, we got to see the system before it was released when it was piloted, but I suppose I wouldn't say regularly.

Did you understand what the project was about? What system was being built and what it would do?

Well my parts yes, but there are parts of the system that I don't use and I don't know how they work, a different department use them so I didn't understand them but the bits I work on yes.

So would you say you had an overall view of the system at the beginning?

Yes, but again only my parts so not the whole system but I came to learn about other parts as we went through the project.

Did you feel you could relate to the project – did you know your role in the project? Could you easily communicate any changes needed?

I'm not exactly sure of the role but any changes could be communicated and we could use the documents that we had to relate to the project as such.

How did you communicate?

I could phone or email the developers if I needed to but to be honest I didn’t really ever have a need to phone that much. The developers were mainly the ones who contacted me.

Appendix C6: Agile Customer Representative Interview Transcript

Are you satisfied with the developed system?

Yes, definitely. I have been a customer representative before for a traditional project and I can definitely see an improvement, especially in the time scale. We were given parts of the system after the first month which is not at all what I'm used to.

Does the system meet the requirements you specified? Does it do what it needs to?

Yes I think it does. There were a few minor things that we wanted changed but these were added in during the development so we hardly noticed.

Did you have a chance to see the system regularly during development?

Yes. We were given a part of it every month or so and were asked to comment on it at a meeting with the developers.

Did you understand what the project was about? What system was being built and what it would do?

Yes I suppose I did. As time went on it became clearer though because we were seeing parts of the system but I was aware of all parts the system would include and had a rough idea of when these would be implemented in terms of at the beginning or end of the development depending on how big they were.
I did understand how the project was being developed though because before the system was even discussed I had a meeting with the IT department and they told me that the development approach would be different and what would be expected of me.

Would you say you had an overall view of the system in the wider scope at the beginning of the project?

Yes I suppose so because although the system was split into iterations we still had to discuss what we would leave and put into other iterations so although we didn't know in depth or in detail what other iterations would involve we had a rough outline of what had to be completed in order to finish the project.

Did you feel you could relate to the project – did you know your role in the project?

Yes I could relate to it because it was me who was telling the developers what it should do. The development approach was explained to me at the beginning of the project too so I knew my role would be very communicative.

So do you think you could communicate any changes easily?

Yes, again the only changes we wanted were minor and probably cosmetic. But these were easily incorporated into the system throughout the project so we really didn't notice them.
But if I had any concerns or problems I could always pick up the phone or email the developers and it would be taken on board.

Was that the only way you communicated with the team?

No, most of the communication took place at our review meetings. Phone calls and emails were mainly only to confirm things.

Do you think the communication was improved in this project?

Oh absolutely. Before I never got to see the system but this way I got to see the developers and the system every month. The communication has definitely increased.

Appendix D: Project Risk Questionnaire

Appendix D details the questionnaire that was used to measure the risk of each respective project. The results of the questionnaire are displayed next to the corresponding statements.

Project Risk Questionnaire

Please rate the following statements on a scale of 1 to 5:

1 – Strongly Disagree
2 – Disagree
3 – Neither Agree nor Disagree
4 – Agree
5 – Strongly Agree

Project Complexity

How would you rate the complexity of the application at the beginning of the development?

	Traditional developer	Traditional developer	Agile developer	Agile developer
System would have a large number of links to future systems.	2	4	3	3
System would have a large number of links to existing systems.	2	4	2	3
The system will involve technical complexity.	4	3	4	4

User Experience

How would you rate the experience of the users?

	Traditional developer	Traditional developer	Agile developer	Agile developer
Users are not familiar with data processing as a working tool.	4	5	2	4
Users are not aware of the importance of their roles in successfully completing the project.	5	2	2	4
Users are not very familiar with this type of application.	4	4	3	5
Users have little experience with the activities to be supported by the application.	2	2	1	3
Users provide an excessive requirements specification.	1	2	2	3

User Support

How would you rate the support the user has towards the project?

	Traditional developer	Traditional developer	Agile developer	Agile developer
Users are not actively participating in the requirements definition.	2	2	1	2
Users have negative attitudes regarding the use of computers in their work.	2	3	1	2
Users slowly respond to development team requests.	2	2	1	4
Users are not ready to accept the changes that the system will entail.	2	2	2	4
Users are not available to answer the questions.	2	2	1	2
Users are not an integral part of the development team.	4	3	2	5
Users are not enthusiastic about the project.	2	3	1	2
Users have a negative opinion about the system meeting their needs.	3	2	1	2

Project Team Expertise

How would you rate the expertise of the development team involved in the project?

	Traditional developer	Traditional developer	Agile developer	Agile developer
Lack of in-depth knowledge of the functioning of the user department.	4	2	4	3
Lack of overall knowledge of organisational operations.	2	3	2	2
Lack of overall administrative experience and skill.	2	3	2	2
Lack of expertise in the specific application area of the system.	4	2	4	3
Lack of familiarity with this type of application.	3	2	2	2

Traditional Project Results

	Traditional developer	Traditional developer	Traditional Total
Total	58	57	115

Agile Project Results

	Agile developer	Agile developer	Agile Total
Total	43	64	107

Appendix E: Raw Data

The raw results of the metrics can be accessed within the CD which accompanies this report. The CD has the following files:

- Results.xls (109KB)– this file contains the raw data from the metric calculations in Microsoft Excel format.
- finalHonourReport.doc (621KB) – this file contains a copy of this report in Microsoft Word format.