Introducing Agile Methods in a Large Software Development Organisation: A Case Study

Mary Giblin
I.D. Number:0600567

Ph.D. Thesis

Supervisor: Dr. Chris Exton
External Examiner: Prof. Par J. Agerfalk
Internal Examiner: Dr Gabriela Avram

Submitted to the University of Limerick, May 2012
Introducing Agile Methods in a Large Software Development Organisation: A Case Study

Mary Giblin

ABSTRACT

The adoption of agile methods of software development has gained momentum within the software industry. NW Soft Solutions Ltd. (a pseudonym) is a large software development unit that develops large-scale network centric software solutions. NW Soft Solutions Ltd decided to adopt an agile development methodology. Martin Fowler in his article “The New Methodology”, states that in his opinion “Since agile methods are so fundamentally people-oriented, it's essential that you start with a team that wants to try and work in an agile way”. Using NW Soft Solutions as a case study, this thesis sets out to show how the developer’s attitudes towards agile methods change during the course of a transition from a more traditional waterfall methodology to agile methods. We see a shift in focus from agile practices at a superficial level towards the core values that underpin agile methods. Object-oriented metrics are used to evaluate and characterise the source code produced by teams using agile methods. The results obtained from the source code produced using agile methods are compared with the results for source code produced using a more traditional methodology. The contrast is stark. This case study shows that agile methods have guided the developers to produce code that manifests better quality and maintainability characteristics. Correlations between the degree of agility in a team and the characteristics of the code produced by that team are also explored.
ACKNOWLEDGEMENTS

I would like to thank the following people for their help in the course of this work:

Dr. Chris Exton for his excellent supervision, clear guidance and providing focus at all times.

Prof. Par J. Agerfalk and Dr Gabriela Avram, for their in-depth review, constructive comments and guidance in completing the thesis.

The management and the developers from the case company involved in the study, in particular, the change program manager and all the developers, testers and systems engineers that participated in interviews and surveys.
# Table of Contents

1 Introduction ................................................................. 1  
1.1 Preamble ...................................................................... 2  
1.2 Background ................................................................. 3  
  1.2.1 Traditional Software Developments Methods .......... 3  
  1.2.2 History of Agility ................................................. 4  
  1.2.3 Overview of Agile Methods ..................................... 7  
  1.2.4 The Agile Adoption Wave ....................................... 9  
1.3 Research Gaps .............................................................. 10  
1.4 Research Questions ....................................................... 11  
  1.4.1 Research Question 1 - Impact on the Developers ...... 12  
  1.4.2 Research Question 2 – Impact on the Code .......... 13  
1.5 The Case Study Company ............................................... 13  
  1.5.1 Background Information ...................................... 13  
  1.5.2 The Process before Agile ....................................... 15  
  1.5.3 The Need for Change ............................................. 16  
  1.5.4 Agile as the Answer - XP and Scrum ...................... 19  
  1.5.5 Adopting Agile Methods ........................................ 24  
1.6 Roadmap ....................................................................... 26  
2 Research Methodology ..................................................... 30  
  2.1 Introduction ............................................................... 31  
  2.2 Research Process Overview ........................................ 32  
  2.3 Motivation for Choice of Research Method ................. 33  
  2.4 Data Collection .......................................................... 34  
    2.4.1 Overview of Data Generation Techniques .............. 34  
    2.4.2 Selection of Data Collection Methods .................... 35  
    2.4.3 Data Collection Stage – D1 ................................. 36  
    2.4.4 Data Collection Stage – D2 .................................. 41  
    2.4.5 Data Collection Stage – D3 .................................. 42  
    2.4.6 Data Collection Stage – D4 .................................. 44  
  2.5 Data Analysis .............................................................. 44  
    2.5.1 Grounded Theory ............................................... 44  
    2.5.2 Why a Grounded Theory Approach is Suitable ...... 45  
    2.5.3 Using Metrics to Evaluate the Code ...................... 47  
    2.5.4 Reliability and Validity of the Study ...................... 53  
    2.5.5 The Researcher .................................................... 55  
3 Initial Concerns and Expectations of the Developers .......... 58  
  3.1 Research Methods and Data Collection ......................... 59  
  3.2 Analysis of the Interview Data ..................................... 63
8.3.1 Agile Adoption and Transformation Models .......... 199
8.3.2 Agile Adoption as Organisational Change .......... 200
8.4 Contributions ......................................................... 201
References ........................................................................ 203
Appendix A ........................................................................ 214
Appendix B ........................................................................ 217
Appendix C ........................................................................ 222
Figures

Fig. 1. Waterfall Model According to Royce ........................................ 3
Fig. 2. Infrastructure for Network Management ................................ 14
Fig. 3. Functional Units ...................................................................... 15
Fig. 4. Pre-agile process timeline ....................................................... 16
Fig. 5. Bug Report Trends .................................................................. 17
Fig. 6. Relative Cost of Correcting Faults ........................................... 17
Fig. 7. Systems and Developers ........................................................ 18
Fig. 8. Developers and Testers .......................................................... 19
Fig. 9. Cross-functional teams ............................................................ 25
Fig. 10. Increased Co-operation between Systems and Developers ... 26
Fig. 11. Co-operation between Developers and Testers .................... 26
Fig. 12. Timeline for the case study .................................................. 27
Fig. 13. Addressing the research questions ......................................... 29
Fig. 14. Model of research process .................................................... 32
Fig. 15. Data Collection Stages .......................................................... 36
Fig. 16. Directed Graph ..................................................................... 48
Fig. 17. Code Example showing Edges and Nodes ......................... 49
Fig. 18. Coding data with Weft QDA ............................................... 54
Fig. 19. Screenshot from Weft QDA ................................................ 61
Fig. 20. Example of a query in Weft QDA ........................................ 61
Fig. 21 Six C’s Ineffective Use of Developers Time ......................... 78
Fig. 22. Six C’s Too High Fault Slippage to Test .............................. 79
Fig. 23. Six C’s Missing Impacts ....................................................... 81
Fig. 24. Overview Pyramid – Structural Aspects .............................. 167
Fig. 25. Size and complexity characterisation .................................. 168
Fig. 26. System coupling characterisation ........................................ 168
Fig. 27. Detection of a God Class ..................................................... 169
Fig. 28. Detection of a Brain Method ............................................... 170
Fig. 29. Detection of a Brain Class ................................................... 170
Fig. 30. Detection of a Data Class ..................................................... 171
Fig. 31. Waterfall Development at the case company ....................... 194
Fig. 32. Research Approach ............................................................. 197
Tables

Table 1: Plan Drive Culture to Agile Development ........................................... 6
Table 2: Research Question 1 .............................................................................. 31
Table 3: Research Question 2 .............................................................................. 31
Table 4: Data Collection Methods .................................................................. 35
Table 5: Sample Question .................................................................................. 42
Table 6: Summary of literature on CK metrics ................................................. 50
Table 7: Codes and categories in Grounded Theory Analysis .......................... 62
Table 8: Summary of major concerns of developers ......................................... 83
Table 9: Summary of major expectations of developers .................................... 83
Table 10: Number of years in software development ........................................ 92
Table 11: Experience Level of Agile methods .................................................. 92
Table 12: Document Driven Process ................................................................. 92
Table 13: Documents are important for maintenance ...................................... 93
Table 14: We write unnecessary documents .................................................... 93
Table 15: Design Impact Spec. is not a valuable document .............................. 93
Table 16: Easier to write the code ................................................................. 93
Table 17: Unit Testing & Code Inspections Compromised ............................. 93
Table 18: Agile is common sense ..................................................................... 94
Table 19: Agile is just a buzzword ................................................................. 94
Table 20: We should continually improve our processes ................................. 94
Table 21: Improve Existing Process ............................................................... 94
Table 22: Too many new concepts in Agile ........................................................ 94
Table 23: More Pressurised Working Environment .......................................... 95
Table 24: Not suitable for home working ....................................................... 95
Table 25: Pair Programming for knowledge transfer ....................................... 95
Table 26: Pair Programming is nothing new .................................................. 95
Table 27: Personality conflicts in Pair Programming ....................................... 95
Table 28: Pair Programming Benefits ............................................................ 96
Table 29: Agile solve our process issues .......................................................... 96
Table 30: Teams need to be co-located ........................................................... 96
Table 31: View on Open Plan ........................................................................... 96
Table 32: Materialisation of initial concerns ..................................................... 112
Table 33: Materialisation of initial expectations .............................................. 113
Table 34: Did the amount of documentation reduce with Agile? .............. 151
Table 35: Opinion on co-location, open plan................................. 151
Table 36: More pressurised environment in agile? ......................... 152
Table 37: Pair Programming .......................................................... 152
Table 38: Test Driven Development ............................................. 153
Table 39: Situation with code inspections .................................... 153
Table 40: User Stories ................................................................. 154
Table 41: Stand Up Meetings ......................................................... 155
Table 42: Continuous Integration ................................................. 155
Table 43: Automated Acceptance Testing ..................................... 156
Table 44: Product Owner Role ....................................................... 156
Table 45: Retrospectives .............................................................. 157
Table 46: Adoption Strategy ........................................................ 157
Table 47: Continue with agile ....................................................... 158
Table 48: Time or Quality Improvements .................................... 159
Table 49: Biggest Benefit of Agile .............................................. 159
Table 50: Negative Aspect of Agile ............................................. 160
Table 51: Materialisation of initial concerns-Teams ....................... 161
Table 52: Materialisation of initial expectations - Teams ............... 161
Table 53: New observations ........................................................ 162
Table 54: Agile Levels defined by Skidy ...................................... 172
Table 55: SAMI populated with principles and concepts ............... 173
Table 56: SAMI table adapted for this study ................................ 174
Table 57: Nominal Values ............................................................ 175
Table 58: Collaborative teams ..................................................... 175
Table 59: Frequent face-to-face communication ............................ 175
Table 60: Continuous Integration ............................................... 176
Table 61: Daily progress tracking meetings ................................ 176
Table 62: User Stories ............................................................... 176
Table 63: User Stories ............................................................... 176
Table 64: TDD ............................................................................ 177
Table 65: Pair Programming ....................................................... 177
Table 66: Agility Assessment per Team ....................................... 177
Table 67: Size and Complexity – Direct Metrics ............................ 178
Table 68: Size and Complexity – Computed Proportions ............... 179
Table 69: System Coupling – Direct Metrics ................................. 179
Table 70: System Coupling – Computed Proportions ..................... 180
Table 71: Design Flaws Detected ................................................ 180
Table 72: Size and Complexity – Computed Proportions ............... 181
Table 73: PMD Violations ............................................................ 183
Table 76: Size and Complexity – Direct Metrics......................... 187
Table 77: Size and Complexity – Computed Proportions............... 187
Table 78: System Coupling – Direct Metrics............................ 188
Table 79: System Coupling – Computed Proportions................... 188
Table 80: Design Flaws Detected.......................................... 188
Table 81: Size and Complexity – Direct Metrics......................... 189
Table 82: Size and Complexity – Computed Proportions............... 189
Table 83: System Coupling – Direct Metrics............................ 190
Table 84: System Coupling – Computed Proportions................... 190
Table 85: Design Flaws Detected.......................................... 190
1 Introduction
1.1 Preamble

The market for software has dictated that in order to stay competitive companies must be able to react to changing customer needs. One software development paradigm that emerged in the last decade to address this challenge is agile software development. The adoption of agile methods of software development has gained momentum within the software industry. Software methodologies and processes underpin software development. A software process is the set of activities and associated artefacts, which produce a software product (Sommerville 2004). Since the earliest days of software development, Software Process Improvement has been an ongoing research topic resulting from the need of organisations to constantly yield productivity and quality improvements. Some initiatives have been more successful than others, but far too much time has been spent on finding and fixing software defects. This thesis investigates the implementation of agile practices in the software industry. The primary research method used throughout the thesis is the case study (Runeson and Host 2009; Yin 2003). A variety of secondary methods for data collection have been used such as semi-structured interviews as well as quantitative data. The case was a large software design company with in excess of 800 software engineers. Software development centres in Ireland and Sweden were involved in the study. The company produces software applications in the telecommunications and network management domains. At the case company, agile practices were introduced for the first time. The company studied was changing from a plan-driven traditional software methodology towards an agile development approach.

In relation to the introduction of agile development approaches this thesis makes two contributions. The first contribution (Contribution I) investigates the impact that migrating from a plan-driven to an agile development approach had on the attitudes of the developers in a large software development team. The second contribution (Contribution II) describes the impact that the agile approach had on characteristics (such as complexity and coupling) of the software produced. Relationships between Contribution I (the concerns and expectations of the developers) and Contribution II (the quality characteristics of the code) are explored.
1.2 Background

1.2.1 Traditional Software Development Methods

The term “traditional methods” as used in this thesis refers to some of the earlier plan-driven software development methodologies. Plan-driven software development is focused on planning everything from the start of a project. The plan-driven approach is characterised as being document centric. Perhaps the most well-known example of plan-driven software development is the waterfall process as suggested by Royce (1970). The process is shown in Figure 1 below.

![Fig. 1. Waterfall Model According to Royce (Sommerville 2004)](image)

The waterfall process is executed sequentially, following steps representing the different stages of software development. Royce’s objective was to provide some structure by assigning distinct software process phases. When a phase is completed the output from that phase is handed over to the following phase, e.g. when the software requirements are specified, the specification is handed over to the analysis phase. However, it would appear that Royce recognised the potential drawback of applying a sequential model to software development. When discussing a sequential model for software development Royce (1970) says “"I believe in this concept, but the implementation [...] is risky and invites failure."

The Rational Unified Process (RUP) is another example of a plan-driven approach (Kruchten 2004). RUP is not as rigid as the Waterfall model when it comes to the sequence in which the disciplines are executed. The engineering disciplines defined by the process (business modeling, requirements specification, analysis and design, implementation, test, and deployment) are executed throughout the overall development life-cycle consisting of
inception, elaboration, construction, and transition. In the inception phase and early elaboration phase the main activities are business modeling and requirements, while these receive less attention in the construction and transition phase. So although the disciplines overlap, they have different emphasis depending on the development phase. RUP also proposes several plans to be documented, such as measurement plans, risk management plans, plans for resolving problems, and plans for the current and upcoming iteration. The RUP process is supported by a number of (IBM) Rational products, such as Rational DOORS (IBM Rational Doors) for requirements management, ClearCase (IBM Rational Clearcase) and ClearQuest (IBM Rational ClearQuest) for change and configuration management as well as Rational Rose RT (IBM Rational Rose Technical Developer) for modeling. The Rational Unified Process is really a process framework that usually needs to be adapted for an individual organisation. To support the tailoring of the RUP process to specific organisational needs, Rational Method Composer (IBM Rational Method Composer) was developed.

The V-Model (Schuppan and Russwurm 2000) is primarily an extension of the waterfall process. The difference is that it maps verification and validation activities to each sequential development step. The delivered product is verified through operation and supported by maintenance activities. The specification of the requirements is verified through acceptance and system integration testing, and the detailed design and coding activities are verified through unit and component testing.

1.2.2 History of Agility

The main goal of Software Engineering consists of the establishment and use of sound engineering principles and methods to obtain economic software that is reliable and works as required (Bauer 1972). These engineering principles and methods form the software development process. The Standish Group studied 40,000 software projects in 10 years between 1994 and 2004 (Johnson 2006). They found that project failures had decreased to 15% of all projects by 2004. This was a 100% improvement over the 31% failure rate reported in 1994. Projects meeting the “challenged” description, which meant they were over time, over budget and/or lacking critical features and requirements accounted for 51% of the projects in 2004. But the size of the cost overruns decreased threefold between 1994 and 2004. According to Standish Chairman Jim Johnson “The primary reason is the projects have gotten a lot smaller. Doing projects with iterative processing as opposed to the waterfall
method, which called for all project requirements to be defined up front, is a major step forward.” (Johnson 2006).

After realising the significance of software development processes in producing “good” software, many efforts arose to identify the “most suitable” software development methodologies. One of the pioneers in this quest was the Software Engineering Institute (SEI). The SEI introduced the Capability Maturity Model for Software (CMM or SW CMM) to the software development community in 1986. The CMM or what is now called CMMI is a process maturity framework that helps organisations improve their software process through a set of recommended practices in a number of key process areas (Emam and Madhavji 1999).

Moving into the twenty first century, the software market presented new challenges to the software development industry. There was increased pressure for faster and more efficient product development, faster time to market, changing customers’ demands, and reduced budgets. The use of the Capability Maturity Model (CMM) and the improved Capability Maturity Model Integration (CMMI) preferred extensive upfront planning and leant itself to rigorous checklists and documentation in an attempt to make development more efficient and predictable. Hence, they were gradually leading the development process towards perfection (Boehm 2002). Many believed that the traditionalists’ approach was the best solution for the problems of the software industry; others did not agree.

Seventeen practitioners wanting to find an alternative to the detailed plan driven development approach convened in February 2001 (Highsmith 2002). The outcome of this meeting, “The Manifesto for Agile Software Development” (Beck et al. 2001), states:

*We are uncovering better ways of developing software by doing it and helping others do it.*

Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

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

The creation of this manifesto brought to bear many different agile development processes and methods. “Agile” means “the quality of being agile; readiness for motion, nimbleness” and the introduction of Agile software development attempted to offer an answer to the
enterprise community that were seeking more efficient and nimbler software development processes. The Internet software industry was rapidly growing and mobile application development was emerging.

When we look at the four values of agile software development, it is clear that some of the principles behind the agile approach are diametrically opposite to those of the plan-driven approach. The plan-driven approach prefers strictly separated phases and the communication between phases relies heavily on documentation, whereas agile advocates “Individuals and interactions over processes and tools”. Agile emphasises the implementation phase and the importance of creating a working product as early as possible with “Working software over comprehensive documentation”. In waterfall development where each phase has to be complete before the next one can begin, there is no opportunity to create working software until the “coding” phase. “Customer collaboration over contract negation” means that the content of the contract is not set in stone at the beginning of the project, but that the specification changes in consultation with the customer throughout the project. In plan-driven development where the requirements specification is a measure of whether the contract has been fulfilled, the requirements specification being established early in the process. “Responding to change over following the plan,” contradicts the plan-driven ethos where changes late in the project cycle require a lot of effort. Table 1 based on work from Hirsch (2005) summarises on a high level the contrast between agile and plan-driven development.

Table 1: Plan Drive Culture to Agile Development (Hirsh 2005)

<table>
<thead>
<tr>
<th>Discipline</th>
<th>Plan-driven</th>
<th>Agile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Model</td>
<td>Waterfall</td>
<td>Iterative &amp; Incremental</td>
</tr>
<tr>
<td>Project Planning</td>
<td>Detailed project plan from start to end. Created early in the project.</td>
<td>Coarse-grained plan for overall project, detailed plans per iteration.</td>
</tr>
<tr>
<td>Requirements Engineering</td>
<td>Dedicated specification phase. Initial requirements are signed off; rigorous change request regime afterwards. Comprehensive requirements often part of a contract.</td>
<td>Requirements evolve over the course of a project. More relaxed change request regime. Easier access to customer rather than reliance on comprehensive requirements document.</td>
</tr>
<tr>
<td>Architecture &amp; Design</td>
<td>Comprehensive architecture and design specifications before implementation begins. Architecture and design try to accommodate for future extensions.</td>
<td>Minimum upfront architecture and design work. Architecture is validated iteration by iteration.</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Implementation</td>
<td>Programming work concentrated in “Implementation” phase. Programmers are assigned to subsystems.</td>
<td>Programming work spread out over entire project. Pair programming. Collective code ownership.</td>
</tr>
<tr>
<td>Testing</td>
<td>Testing phase at the end of the project. Tests are designed and executed by test specialists.</td>
<td>Testing spread out over entire project. Functional tests are specified and executed by end users.</td>
</tr>
<tr>
<td>Quality Assurance</td>
<td>Formal QA role. QA role is responsible for formal and informal reviews, development process, configuration management, testing, code inspections</td>
<td>No explicit QA role No formal reviews Attitude: quality is the result of how we work here.</td>
</tr>
</tbody>
</table>

The publication of the Agile manifesto resulted in the emergence of an amount of literature around this time (Cockburn 2002; Highsmith 2002; Martin 2002) and some debate on the subject (Yourdon 2000; Highsmith 2001; Highsmith 2002). Some authors such as Baskerville et al. (2002) and Merisalo-Rantanen et al. (2005) argued that agile software development methods did not offer anything new with regard to software development. The proponents of agile maintained that the agile principles represented a paradigm shift in software engineering (Rajlich 2006). An interview with INFOQ, Standish Chairman Jim Johnson (Preuss 2006), cites Agile Software Development as a factor for the improvement the group found in software development projects between 1994 and 2004. “Agile has helped by chunking projects into small pieces. Even if you start to get it wrong, you know it early on. Not like the old waterfall projects where you go into a cave and find out the bad news two years later...”

### 1.2.3 Overview of Agile Methods

Agile methods are characterised by the following attributes: incremental (small software releases, with rapid development cycles), cooperation (close customer and developer interaction), straightforward (the method itself is easy to learn and to modify and it is sufficiently documented), and adaptive (the ability to make and react to last moment changes)
(Abrahamsson et al 2002). Abrahamsson et al. (2010) presents the intellectual origins of agile methods and an evolutionary map of agile software development methods.

As mentioned earlier, methodologies that have been characterised as the more traditional methodologies or plan-driven are Waterfall (Royce 1970), Spiral (Boehm 1988), RUP (IBM RUP) and the V-model (Schuppan and Russwurm 2000). A number of methodologies fall under the “Agile umbrella”. The objectives of some of the most common or well-known ones are described below.

**Adaptive Software Development** (ASD) is a software development process that grew out of rapid application development work by Jim Highsmith and Sam Bayer (Highsmith 2000). ASD uses the principle that continuous adaptation of the process to the work at hand is the normal state of affairs. It encourages incremental and iterative development, with constant prototyping. A predecessor of ASD is “RADical Software Development” (Bayer and Highsmith 1994).

**Agile Modelling** (AM), defined by Scott Ambler (2002) is the agile version of Model Driven Development (MDD). It applies the idea of agile, rapid development to modeling. It attempts to encourage developers to produce enough advanced models to support the design needs while at the same time keeping the amount of models and documentation as low as possible.

**Crystal family** of methodologies described by Alistair Cockburn includes a number of different methods from which to select the most suitable one for a project (Cockburn 2002; Cockburn 1998; Cockburn 1999). The Crystal approach also includes rules for tailoring of these methods.

**Dynamic Systems Development Method** (DSDM), (DSDM Consortium 1998; Stapleton 1997) is considered the first truly agile software development method (Abrahamsson et al, 2002). The fundamental idea behind DSDM is that instead of fixing the amount of functionality in a product, it is preferred to fix time and resources and then to adjust the amount of functionality accordingly.

**Extreme Programming** (XP) described by Kent Beck is a collection of well-known software engineering practices (Beck 1999; Beck 2000; Beck and Anders 2004). XP contains short iterations with small releases and rapid feedback, close cooperation with the customer,
constant communication and coordination, re-factoring, continuous integration and testing, collective code ownership, pair programming and test first development.

**Feature Driven Development** (FDD), described by Coad et al. (1999) and by Palmer and Felsing (2002), focuses on the design and building phases and defines frequent and tangible deliveries, along with accurate monitoring of the progress of the project as its core.

**Scrum** is a management and control process that can be superimposed on top of other engineering practices and development methodologies (Schwaber 1995; Schwaber and Beedle 2002).

**Lean software development** is a translation of Lean manufacturing and Lean IT principles and practices to the software development domain. It was initially adapted from the Toyota Production System. Mary and Tom Poppendieck have published several books (Poppendieck and Poppendieck 2003; Poppendieck and Poppendieck 2007; Poppendieck and Poppendieck 2010) on the translation and application of lean manufacturing and product development practices to a software engineering context. In the book “Lean Software Development” (Poppendieck and Poppendieck 2003), Mary and Tom Poppendieck identify seven fundamental “lean” principles and adapted them for software development. They also introduced 22 “thinking tools” that can support the customisation of the right agile practices for any environment.

**1.2.4 The Agile Adoption Wave**

At first, organisations were sceptical about adopting agile software methodologies. Information Technology (IT) managers were asking the agile community “*why should we adopt agile practices?*” The agile community provided results from many pilot projects and small-scale transitions that occurred in organisations. The results were impressive and empirical evidence showed that embracing agile practices yielded many benefits (Williams et al. 2000; Schatz and Abdelshafi 2005; Barnett and Schwaber 2004; Barnett 2006; Law and Charron 2005; Kuppuswami et al. 2003). Some of the benefits of introducing agile methods were cited as: Shorter time to market, better quality, better customer relations and better team motivation. Research by Shire Technologies (Shire Technologies Ltd. 2003) with select companies adopting agile methods found that:
93% experienced a productivity increase;

88% produced better quality software

83% found increased business satisfaction.

The initial uptake of agile methods was in small projects and organisations. Agile adoption was on the rise but when it came to large projects and large organisations, the transition became more difficult. Research by Lindvall et al. (2004) indicated that many companies, in particular large companies, approached agile methods with a high degree of scepticism due to the conflicting data regarding “in what environments and under what conditions agile methods work”.

A number of papers, case studies and experience reports produced on large scale agile adoption show that the adoption process takes time. “To move to disciplined execution of agile practices you can start straight with new practices: an overnight introduction.” However, making the transition permanent “requires a serious, long term change process with good guidance. Management has to guide and support the change” (Goos and Mellisse 2008). And “Moving from waterfall to agile cannot be done overnight or in a single step. It takes time for people to unlearn old traditional practices and move towards agile” (Sureshchandra and Shrinivasavadhani 2008).

1.3 Research Gaps

So while there appears to be a lot to gain from adapting an agile software development methodology, there also have been people who oppose this thinking. Germain & Robillard (2005) claim that an agile software development method like Extreme Programming in the hands of inexperienced and untalented developers becomes cumbersome. Agile has been described as a “fad methodology” (Papadimoulsi 2007). One of the objectives of a systematic review by Dyba and Dingsoyr (2008) was to ascertain “What is currently known about the benefits and limitations of agile software development?” Two of the conclusions from the review were that: -

- The majority of empirical studies of sufficient quality focused on eXtreme programming (XP)
- The studies often focused on agile implementations on a smaller scale (focused on teams with a size range from 4 to 23).

This indicates a general need for studying agile implementations in a large-scale environment. The study by Dyba and Dingsoyr also found that “The effect on work practices
and job satisfaction of using agile and traditional methods has not been established conclusively.” “Developers are mostly satisfied with agile methods. Companies that use XP have reported that their employees are more satisfied with their job and that they are more satisfied with the product.” Whitworth and Biddle (2007) reported that “there is a lack of basic research into the socio-psychological experience of individuals in agile software development teams”. Tessem and Maurer (2007) stated “research on what factors are essential to help with agility in larger teams are scarce.”

In order to assist organisations with their agile adoption efforts, agile adoption frameworks such as that developed by Sidky et al. (2007a) can assist in identifying the agile potential for projects and organisations. Because agile is fundamentally a “people orientated process”, the success or failure of the adoption of the agile methods will be reflected in the satisfaction of the developers. So a key question is “how does the adoption of agile methods affect the developers in a large team?” The motivation for the research in this thesis was to explore the opinions of developers in a large software development organisation regarding agile adoption and to gain an understanding of how their opinions changed throughout the course of the agile adoption period. Another objective was to examine the characteristics of the code produced using agile methods and to investigate if the characteristics such as complexity and coupling were different to those of code produced using traditional methods. This case study takes a twofold approach by studying the concerns, expectations and views of the developers as well as examining the code characteristics such as coupling and complexity measurements. This allowed us to build up a richer picture of the transition from the use of a plan driven approach to agile methods. By studying the views of the developers as well as analysing code characteristics, we felt that we may be able to better understand and explain the changes or perhaps lack of changes to the characteristics of the code when agile methods had been used. It allowed us to explore possible relationships between the views of the developers and possible changes in the code characteristics when moving from a traditional waterfall model to agile methods.

1.4 Research Questions

As mentioned earlier, this thesis makes two contributions. It is primarily focusing on two areas: -

1. The impact that the introduction of an agile methodology has on the developers.
2. The impact that the introduction of an agile methodology has on the code being produced.
The two contributions are fundamentally linked, because findings from the research on Item 1 can be used to explain possible findings with respect to Item 2. This two-pronged approach was chosen in order to obtain a better understanding of the findings. For example if there were no improvements in the characteristics of the code produced using agile methods –that be might be attributed to the fact that the developers were struggling with adopting the agile practices. Based on the contributions, two broad research questions were formulated. Each research question was split into a number of sub-questions, which are listed below.

1.4.1 Research Question 1 - Impact on the Developers

This part of the research concentrated on the concerns and the expectations of the developers as they progressed through their agile journey. We documented the expectations of the developers and their concerns at the very initial stage of the agile adoption. We got a picture of how they initially reacted to the introduction agile methods. Then at a later stage (after 9-12 months working with agile methods), we monitored the concerns and expectations of the developers again. So first of all, we explored what the initial concerns and expectations of the developers were with respect to agile methods. Then as the developers gained experience with the methodology, we analysed how their concerns and expectations changed over time. The research question is formulated as a main question with a number of sub-questions.

Q1 What is the effect of introducing agile methods on the concerns and expectations of the developers?

q1.1 What are the initial concerns and expectations of the developers before the adoption of agile methods?

q1.2 Did the initial concerns and expectations materialise and how did they change during the agile adoption phase?

q1.3 Did any new concerns or issues materialise during the agile adoption phase?

q1.4 Did the introduction of agile produce results or effects not initially anticipated?
1.4.2 Research Question 2 – Impact on the Code

The second part of the research looked at the software produced using an agile methodology and compared the characteristics of the code with the characteristics of code produced using traditional methods. One important aspect of this work was to decide how to evaluate or characterise the code. Again the main research question was split into a number of sub-questions.

Q2  What is the effect of introducing agile methods on the characteristics of the software produced?
q2.1  How can Object Oriented software be characterised in a good way?
q2.2  What are the characteristics of software produced using agile methods?
q2.3  How does the characteristics of software produced using agile methods compare with the characteristics of software produced using traditional methods?
q2.4  Are there possible links between the views of the developers, their engagement with the agile process and the characteristics of the code produced using agile methods?

1.5 The Case Study Company

1.5.1 Background Information

As previously stated this research was carried out as a case study in conjunction with a large software development company. The company produces mobile and fixed telecommunications network infrastructure and broadband and multimedia solutions and has in the region of 65,000 employees worldwide. The area of the operation involved in this study, NW Soft Solutions Ltd. (a pseudonym) is a Product Development unit responsible for the design, development and maintenance of Network Management solutions for the Wireless Access and Core Network. The Product Development is distributed globally. This study involved units in Ireland and Sweden. The Irish product development unit has been established for approximately 25 years and the organisation in general and the Irish and Swedish operations have a long tradition of mature and stable software development processes. They have experience with a number of process lifecycles from the traditional waterfall through to a variation of the Rational Unified Process (RUP) (Kruchten 2004) and iterative design.
NW Soft Solutions Ltd. employs 800+ software developers, testers and systems engineers who are responsible for the design, development and maintenance of Network Management solutions for the Wireless Access and Core Network. Network Management products are complex and their development is generally fraught with co-ordination issues. A description is included to attempt to illustrate the complexity and coordination issues facing the developers of Network Management Systems. Saydam and Magedanz (1996) provides a definition of Network Management. “Network management includes the deployment, integration, and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyse, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost.” Figure 2 shows the basic network management architecture. The managed devices in the network consist of managed objects whose data is gathered in a Management Information Base (MIB). An instance of the MIB is known as a MOM (Management Object Model.). Descriptions of the UTRAN and Core Network architectures can be found in (UMTSWorld).

![Fig. 2. Infrastructure for Network Management (Kurose and Ross 2005)](image)

The Network Management System is designed and built to run on Unix machines and is coded mainly in Java. It uses a 3rd party commercial database as well as a number of other 3rd party products (e.g. topology viewers). The development of the Network Management System is complex with many coordination issues contributed to by the following:

1. The products are developed in a two-stage process, i.e. a domain (platform) features and the applications themselves. An example of a platform feature would be a communications bus that would be used by many applications.

2. There are a large number of types of managed devices in the network and it must be possible to cater for multiple versions of the Management Object Model (MOM).
3. Any new features introduced in a managed device requiring configuration will impact the Network Management System, thus introducing a dependency to another project.
4. There are a number of 3GPP standards and regulatory issues that must be taken into account.
5. One Network Management system generally caters for the needs of multiple customers, with optional features turned on with a license key, therefore in a single release, requirements for a number of customers are handled.

1.5.2 The Process before Agile

It is worth noting that NW Soft Solutions have been developing software systems for more than 25 years and have had a long history in software process and methodology. Up until approximately 10 years ago, the process used was predominantly waterfall with a big emphasis on documentation. Since then, variations of the Rational Unified Process (Kruchten 2004) with UML modeling has been used, along with other home grown iterative processes. The organisation has launched many Software Improvement Initiatives, and achieved CMMI Level 3 compliance.

The company had dedicated “siloeed” Systems, Design and Test groups. Figure 3 shows the functional units that existed in the organisation.

![Fig. 3. Functional Units](image)

We now describe the process that was used before the adoption of agile practices. From Figure 4 below we can see three distinct phases, which are feasibility, design and test. The top row of numbers gives the average duration of each phase in weeks in a typical project. We can see that generally each iteration took 15 weeks from start of design to delivery and up to 26 weeks from the start of feasibility to delivery. Normally a complete project (delivery to customer) consisted of three to four iterations of the design and test phases.
Prior to the feasibility phase, the requirements were gathered, documented and an extremely high level textual description of the system impacts would be completed. This was a multi-site, multi-customer activity. The Systems and Product Managers conducted the requirement gathering activity. The x-axis in Figure 4 shows us that the activity began about 26 weeks before delivery of tested software. At the end of pre-study or start of feasibility a definitive set of requirements was selected for development. The feasibility phase further detailed the requirements and documented the system impacts again in a text-based document called an Implementation Proposal (IP). The Systems Engineers conducted the feasibility study. When the Implementation Proposals were approved, Interworking Descriptions, Managed Object Models etc. were updated and placed under version control. Therefore Requirement Specifications were frozen 26 weeks prior to product shipment and Interface Descriptions were frozen 15 weeks prior to shipment.

The Design phase consisted of coding and unit testing. A Feature Test (FT) phase, an Integration Test Phase (I) and a System Test (ST) phase followed this. The product was then ready to be delivered to a customer where it was tested in a network with other inter-working products in the form of a Network Integration Verification (NIV).

### 1.5.3 The Need for Change

To stay competitive in the market software development teams need to continually update and improve their processes and methodologies. The main objective for NW Soft Solutions Ltd was to find faults earlier in the design cycle thus reducing the overall cost of testing and debugging. The goal in adopting an agile methodology was to significantly reduce the amount of faults found in the later testing phases and after product release. Figure 5 shows that the number of total number of bugs reported for 5 different releases of the product developed using traditional methods. Faults continued to be reported after the product was shipped to customers i.e. Generally Available (GA). Too many faults were also found during development (before GA) and too late in the development cycle.
The later a fault is discovered in the product development lifecycle, the higher the cost to analyse the cause of the fault and to fix it. Figure 6 depicts how the defect removal cost increase exponentially with time.

Based on Root Cause Analysis reports that investigated defect introduction and some Opportunity for Improvement sessions the following areas were identified by the case company as issues to be addressed.

1. Unit Testing and Basic Integration Testing were sometimes compromised due to pressure to deliver.
2. Competence and intentions were sometimes lost at handover.
3. One process was used for all sizes and types of impact or feature
4. The design cycle and associated tests did not have sufficient controls, techniques or
tests to eliminate faults e.g. enough automated tests.
5. No early characteristics verification was performed.
6. The first integration point was also the start of Feature Test.
7. It was difficult to remove or switch off offending applications that “break” the
system.

Intention and knowledge seemed to be lost at handovers. It appeared that there was a
“wall” between the Systems Engineers and the Developers. Systems passed on their
requirements to the developers and there was little interaction thereafter. Developers “threw
the code over the wall” to the Testers and the testers returned bug reports. This phenomenon
is depicted in Figures 7 and 8.

Fig. 7. Systems and Developers
1.5.4 Agile as the Answer - XP and Scrum

This section introduces XP and Scrum in more detail, since they are the two agile
development methods used in the case study.

**XP**

eXtreme Programming (XP) is regarded as the most popular agile method in use today
(Fowler 2005). In 2000 Kent Beck published the first version of the book: *Extreme
states that the goal of XP is “outstanding software development in an environment of vague
or rapidly changing requirements”. Beck describes the practices as “things you actually do”.
Writing a test before you change the code is an example of a practice. Practices are evidence
of values. Values are universal and are expressed at a high level. Communication is an
example of a value. Values are not domain specific. Principles are domain specific guidelines
that bridge the gap between values and practices. An example of a principle would be
“redundancy” in critical systems.

The driving **values** of XP as specified by Beck, which underlie the principles and practices
are:

- **Communication**- In order for the team to function, good communication is vital.
Other team members may already know the solution to a problem or at least help to
solve the problem.
Simplicity- Beck asks the question “What is the simplest thing that could possibly work?” This is to encourage teams to remove unnecessary complexity and to produce working software that meets today’s requirements, instead of attempting to cater for features that may or may not be required in the future.

Feedback- The team needs to generate short feedback cycles that are as short as possible in order to get closer to reaching their goals with that software.

Courage- In order for a team to work well, the value of courage needs to prevail. Otherwise fear will dictate their effectiveness. One example cited in the book is “the courage to discard a failing solution”.

Respect- This value was added by Beck in the second edition of his book (Beck and Anders 2004). The team members need to respect each other as well as the project they are working on in order for XP to work. Equality among team members is important. “No one is intrinsically worth more than anyone else.”

Beck describes fourteen principles of XP, which are grounded in the driving values. Teams can develop practices based on the guiding principles of (1) humanity, (2) economics, (3) mutual benefit, (4) self-similarity, (5) improvement, (6) diversity, (7) reflection, (8) flow, (9) opportunity, (10) redundancy, (11) failure, (12) quality, (13) baby steps and (14) accepted responsibility.

XP practices
The practices of XP are what XP teams do on a daily basis in their work (Beck and Anders 2004; Highsmith 2002). In the 2nd edition of his book, Beck specifies a set of primary practices, which he envisages as the ones you should start with when you begin to apply XP (Beck and Anders 2004).

1. Sit together- The team sits in an open space, surrounded by artefacts displaying project status. This is not always possible for a multi-site team.
2. Whole team- This refers to the old “cross-functional team” idea. People with all the skills needed for that project to succeed must be included on the team and the sense of “team” is key for success of the project.
Informative workspace—The workspace should provide visibility on the status of the project and facilitate communication within the team about the project. The workspace should also consider other needs such as social interactions, for example, water or snacks.

Energised work—The team should only be working the number of hours they can be productive.

Pair programming—Two programmers sit at one machine. They take turns in driving the keyboard and mouse. Pairs are encouraged to rotate frequently.

Stories—(or User Stories) are units of customer-visible functionality. As soon as a story is defined, an attempt should be made to estimate the development effort. Early estimation is the key difference between Stories and other requirement practices.

Weekly Cycle—Work should be planned only for the next week. The goal is having deployable software at the end of each week.

Quarterly Cycle—Quarterly reflection about the team, the project and how it’s progress aligns with other large goals. For example, identifying bottlenecks, especially those controlled outside the team could fit into the quarterly cycle planning.

Slack—Include minor tasks in the development plan that can be dropped if necessary.

Ten-minute build—Automatically building the system and running all of the tests should be done in ten minutes.

Continuous integration—Changes should be integrated and tested every few hours.

Test-first programming—Before writing any code, developers should write a failing automated test. Test-first programming address problems such as scope creep and coupling and cohesion.

Incremental design Instead of Upfront System design, XP advocates that it is more efficient to do the design close to when it is used. Making constant improvements to the design as the system is implemented incrementally, keeps the cost of change low. Improving system design by refactoring is a technique used by XP teams.

Beck and Anders (2004) also specifies a set of corollary practices, which they say are “difficult or dangerous to implement before completing the preliminary work or the primary practices”.

Real Customer Involvement—The end customer is part of the team. They can be part of the weekly and quarterly planning.
2 Incremental Deployment- This means finding a way to migrate from the old system to the new system slowly in a stepwise fashion.

3 Team Continuity- Effective teams should be kept together. Large organisations often have more “fluid” teams with teams being formed for the duration of a project.

4 Shrinking Teams- As a team grows in capability, keep its workload constant, but gradually reduce its size.

5 Root-Cause Analysis- Every time a defect slips through, ask why the problem occurred and if possible eliminate the cause of the defect.

6 Shared Code- “Anyone on the team can improve any part of the system at any time.”

7 Code and Tests- The only documentation maintained should be the code and the tests. Other documents should be generated from the code and the tests.

8 Single Code Base- Avoid multiple code branches or streams.

9 Daily Deployment- Daily Deployment means putting new software into production every night. There are many pre-requisites to be able to do that, such as, very low bug or defect slip through, trust and cooperation with the customer, automated deployment tools.

10 Negotiated Scope Contract- Contracts should be fixed on the basis of time, budget and quality, but the scope should be under constant negotiation.

11 Pay-Per-Use- Charge for every time the system is used, instead of charging for each release of software.

Scrum

Scrum is not an acronym but a term from rugby. It is named after the scrum in rugby, which refers to “a tight formation of forwards who bind together in specific positions to get an out-of-play ball back into play (an action called a scrumdown),” (Lenz and Moeller 2003). Scrum was first observed by Takeuchi and Nonaka (1986). Scrum is described in Schwaber and Beedle’s book published in 2002 (Schwaber and Beedle 2002). Schwaber describes Scrum as “operating at the edge of chaos,” which aims to “operate adaptively within a complex environment using imprecise processes” (Schwaber 1995). Schwaber states, “The Scrum software development process uses an iterative, incremental approach. Interaction with the environment (technical, competitive, and user) is allowed, which will change the project scope, technology, functionality, cost, and schedule whenever required.” (Schwaber 1996). Here some of the key roles and concepts used in Scrum as defined by Schwaber and Beedle (2002) are explained.
**Product Owner**- Defines the features of the product, decides on release dates and content, and prioritises features according to market value. They can change features and priority every 30 days and the Product Owner accepts or rejects work results.

**Product Backlog**- This is a list of all features, functionality, enhancements and bug fixes needed in a product. It is constantly evolving and the Product Owner maintains it.

**Scrum Team**- The team is a cross-functional team that performs the development.

**Sprint**- The Scrum Team takes on as much Product Backlog as they think they can fit into an increment of product functionality within a 30-day iteration called a Sprint.

**Sprint Backlog**- The Sprint Backlog is a list of tasks to be performed during the sprint. The team maintains the Sprint Backlog.

**Daily Scrum**- The Scrum Team meets daily for a short status meeting, called the Daily Scrum. At the daily scrum, progress is reviewed and barriers to progress are identified.

**Scrum Master**- The Scrum Master helps the team implement the Scrum practices. When the team is in a Sprint, it should not be disturbed or be given direction by anyone outside it.

**Sprint Planning Meeting**- Users, customers, the product owner and the scrum team determine the next Sprint goal and functionality at a planning meeting called the sprint planning.

**Chickens and Pigs**- Team members commit to a goal and are called “pigs”. Everyone else is a “chicken”. Chickens may attend the Daily Scrums but they have to stay on the sideline and not interfere with the meeting.

**Sprint Review**; The Sprint Review takes place at the end of a sprint. The team present what they have achieved during the sprint and assess the positive and negative outcomes of the sprint.

---

**Experiences with XP and Scrum**

The first documented use of XP was the Chrysler Comprehensive Compensation (C3) project. The objective of the project was to rewrite the Daimler-Chrysler payroll package (THE C3 TEAM 1998). It is clear from the literature that since then XP has been successfully applied to numerous small (Layman et al. 2004), medium (Hodgetts and Phillips 2002), and large (Lindvall et al. 2004) software projects as well as distributed teams (Braithwaite and Joyce 2005).
XP has also been successfully applied different application domains (Grossman et al. 2004; Nielsen and McMunn 2005).

Takeuchi and Nonaka (1986) reported on successes with Scrum in high profile companies such as Fuji-Xerox, Canon, Honda, NEC and Hewlett-Packard. According to Sutherland (2001), Scrum any be applied in any development environment (Smits and Pshigoda 2007). Cases where the implementation failed are less prevalent in the literature with one such case documented by O’ Donnell and Richardson (2008). Lindvall et al. (2002) found that “Whether or not to use a certain software development methodology is not trivial and depends on many factors” and that the three most important success factors are “culture, people, and communication”.

1.5.5 Adopting Agile Methods

Initially in our case company, a small number of senior technical personnel along with project managers and others, detailed what they perceived as areas for improvement based on data and metrics from previous projects. Their analysis undoubtedly pointed towards a need for early testing, independent integration and verification of features and subsystems before they were delivered into the Feature and System Test thus eliminating the problematic “big bang” style integration. Seven core principles to be underpinned by agile practices were agreed as a basis for improving the process and presented to the developers. An aggressive adoption of agile practices (a mix of XP and Scrum) was outlined and it included

- Feature teams (systems, developers, testers)
- Improved product and project anatomy planning
- User stories
- 2-3 week iterations
- Continuous build and integration
- Measuring velocity and burndown charts
- Agile leaders and Scrum Masters
- Co-located and self-organising teams
- Automated unit and feature test
- Test driven development
- Pair programming
- Stand-up meetings.

To adopt an agile process in a software development organisation, “who needs to change?” And the answer is “everybody”. Using an Agile Approach to software development “is a
challenging task demanding a great deal of adjustment from all the stakeholders involved in the software development process” (Whitworth and Beedle 2007). When introducing an agile way of working, the organisation opted to keep the Systems, Developers, and Test as separate functional units, but hoped for a much-increased co-operation between the units.

![Cross-functional teams](image)

**Fig. 9. Cross-functional teams**

**Systems**

Systems had to introduce the concept of User Stories in the early design phase. Due to the lack of access to the “real customer”, Systems had to take on the role of “customer” or “product owner”. The plan was for Systems to receive demos of the User Stories at the end of each sprint and the intention was that they would co-operate closely with design during the development of features. As Product Owners, Systems had the task of continuously prioritising User Stories and maintaining a Product Backlog. For very large features, the plan was that Design would work with Systems on the User Stories.

System Engineers also had to deal with feature anatomy handling, where each feature would have a dependency description outlining User Stories that were dependant on other User Stories. They needed to reduce dependencies (without duplicating functionality) between User Stories and Features so that features or stories that “break” the system could be removed.

While Systems remained as separate unit, most of the Systems Engineers would be part of the “Feature Team” consisting of Systems, Developers and Testers. This objective was to have increased co-operation between Systems and Developers, thus lowering the barrier between Systems and the Developers.
Developers

Developers were to employ the following mainly XP practices and SCRUM – Pair Programming, Test Driven Development, Planning Game, User Acceptance Test per Story, Task Tracking, Daily Stand-up Meetings, Visible Charts on progress, Daily Builds.

Testers were to work with developers to prepare and execute the User Acceptance Tests. Increased cooperation between Developers and Test was envisaged as shown in Figure 11.

Testers

Testers also had to make adjustments. Testers were now be involved with design in writing and running the User Acceptance Tests. As many tests as possible were to be automated in Feature Test and Basic Integration Test. Automated Test cases were to be identified and provided as part of the build procedure.

1.6 Roadmap

The next chapter, Chapter 2, discusses the research methodology used; the principles used when designing the questionnaires, and the way in which the data was analysed. It also describes the method used to characterise the code. First of all Figure 12 gives an overview of the
timeframe for the case study. Items 3 and 4 (interviews and questionnaires) from Figure 12 were conducted in Ireland and the interviews after the adoption of agile methods (Item 6) were carried out in Ireland and Sweden.

Chapter 3 contains the qualitative data and findings from an analysis of the initial interviews with developers. This analysis took place at the start of the agile adoption process.

Chapter 4 contains the data and findings from a survey that was carried out at the beginning of the agile adoption phase.

Chapter 5 analyses data from interviews carried out 9-12 months into the adoption of agile methods and describes how the views of the developers have changed from the initial stages of adoption of agile methods. The analysis is based on one-to-one interviews with developers.

Chapter 6 also analyses data from interviews carried out 9-12 months into the adoption of agile methods and describes how the views of the developers have changed from the initial stages of adoption of agile methods. This time the analysis is based on group interviews on a per team basis.

Chapter 7 contains an analysis of the code and compares characteristics of code developed using traditional methods with code developed using agile methods. Because some interviews that were conducted after adoption of agile methods were team based, it was possible to develop an “agility index” per team. The purpose of the “agility index” was to quantify the degree of agile practices the team had implemented. This made it possible to investigate if code from teams with a higher level of agile practices presented code with more favourable characteristics.

Finally Chapter 8 contains a summary and conclusions. Figure 13 provides an overview of the breakdown of the main research questions and how they are linked to the individual chapters in which they are addressed.
Contribution 1: Research questions related to the effect of introducing agile methods on the concerns and expectations of the developers?

q1.1 What are the initial concerns and expectations of the developers before the adoption of agile methods?
q1.2 Did the initial concerns and expectations materialise and how did they change during the agile adoption phase?

Chapter 3 Initial concerns and expectations of the developers

Chapter 4 Validating the view of the developers

Contribution 1: Research questions related to the effect of introducing agile methods on the concerns and expectations of the developers?

q1.3 Did any new concerns or issues materialise during the agile adoption phase?
q1.4 Did the introduction of agile produce results or effects not initially anticipated?

Chapter 5 The change in view of the developers (one-to-one interviews)

Chapter 6 The change in view of the developers (team/group interviews)
Contribution 2: Research questions related to the effect of introducing agile methods on the characteristics of the software produced?

q2.1 How can Object Oriented software be characterised?
q2.2 What are the characteristics of software produced using agile methods?
q2.3 How does the characteristics of software produced using agile methods compare with the characteristics of software produced using traditional methods?
q2.4 Are there possible links between the views of the developers, their engagement with the agile process and the characteristics of the code produced using agile methods?

Chapter 7 Agile Methods: The impact on the code

Fig. 13. Addressing the research questions
2 Research Methodology
2.1 Introduction

As mentioned earlier in Chapter 1, one of the contributions of this thesis is to determine how the introduction of agile methods in a large software development organisation affects the developers. The second contribution is to analyse the effect of introducing agile methods on the code. The research questions are repeated here in Table 2 and Table 3. Each research question consists of a number of sub-questions.

Table 2: Research Question 1

<table>
<thead>
<tr>
<th>Q1</th>
<th>What is the effect of introducing agile methods on the concerns and expectations of the developers?</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1.1</td>
<td>What are the initial concerns and expectations of the developers before the adoption of agile methods?</td>
</tr>
<tr>
<td>q1.2</td>
<td>Did the initial concerns and expectations materialise and how did they change during the agile adoption phase?</td>
</tr>
<tr>
<td>q1.3</td>
<td>Did any new concerns or issues materialise during the agile adoption phase?</td>
</tr>
<tr>
<td>q1.4</td>
<td>Did the introduction of agile produce results or effects not initially anticipated?</td>
</tr>
</tbody>
</table>

Table 3: Research Question 2

<table>
<thead>
<tr>
<th>Q2</th>
<th>What is the effect of introducing agile methods on the characteristics of the software produced?</th>
</tr>
</thead>
<tbody>
<tr>
<td>q2.1</td>
<td>How can Object Oriented software be characterised?</td>
</tr>
<tr>
<td>q2.2</td>
<td>What are the characteristics of software produced using agile methods?</td>
</tr>
<tr>
<td>q2.3</td>
<td>How does the characteristics of software produced using agile methods compare with software produced using traditional methods?</td>
</tr>
<tr>
<td>q2.4</td>
<td>Are there possible links between the views of the developers and the characteristics of the code produced using agile methods?</td>
</tr>
</tbody>
</table>

This chapter explains the research methodologies that were used in this study and the motivation for their selection. We also illustrate the data gathering and analysis methods and discuss aspects of the study that may affect the validity and reliability of the results.
2.2 Research Process Overview

Figure 14 (Oates 2007) gives an overview of the research process as it applies in the software engineering context. Some commonly used strategies, data generation methods and data analysis methods are shown.

![Model of research process](image)

Fig. 14. Model of research process (Oates 2007)

Commonly used research methods in the software engineering context are experiments (Wohlin et al. 2000), surveys (Fowler 1995), case studies (Whitworth and Biddle 2007), action research (Somekh 2006), and simulation (Shull et al. 2008).

**Experiments:** This research method is used to test theories in a controlled environment. There is a “before” and “after” measurement. Hypotheses are formulated regarding the cause effect relationship between one or more independent variables and independent (outcome) variables. Factors that might affect the results are excluded from the study. This means that variables other than the independent variables should not have an effect on the outcome variables. For example, when testing the effect of an agile practice such as Test Driven Development on code quality, other factors such as experience levels should be controlled so they do not affect the outcome.

**Surveys:** A survey focuses on obtaining the same kind of data for a population by surveying a sample that is representative for that population. In order to collect the data from the sample, questionnaires and interviews are often used. Online questionnaires are useful because of their potential to reach a large sample of the population.
**Case Studies**: Focuses on one instance of the phenomena that is being investigated. The cases are objects of the real world studied in a natural setting. In software engineering this means they are real software organisations, software departments, software projects, software developers, discussion forums etc. The aim of the case study is to obtain a “rich detailed insight into the ‘life’ of that case and its complex relationships and processes” (Oates 2007). To conduct a case study, a case to be studied must be selected and a data collection strategy defined. A case study can be exploratory, descriptive or explanatory (Yin 2003). An exploratory case study is suitable when there is little knowledge about the topic in the real world available and it aims at identifying theories and propositions. A descriptive study tells a story of what occurred and how different people perceive what occurred. An explanatory case study sets out with a proposition or hypotheses to be tested in the real world. Case studies can vary in their approach to time and are classed as historical (examines what has happened in the past), short-term (examines what is occurring now) and longitudinal (examines the case over time) (Yin 2003). This thesis contains a case study that can be classified as descriptive longitudinal study where the effect of the transition between two development paradigms is studied.

**Action Research**: In action research the goal is to plan and “do something” (Yin 2003) in a real world setting. The affect of the action is observed. The researcher takes an active part in the action (e.g. by participating in a software development team affected by the action introduced).

**Ethnography**: focuses on understanding the culture and ways of seeing a particular group of people.

### 2.3 Motivation for Choice of Research Method

The primary or over-arching research method used in this thesis was the case study. As mentioned earlier, the contribution of this thesis is the investigation of the adoption of agile practices in a large software development team. An experimental approach was not applicable because it would be impossible to replicate the environment in a laboratory setting. A survey is more suited to obtaining an overall view of a topic but does not lend itself to more detailed investigation. Survey respondents are unlikely to answer detailed in-depth questions in a survey. We felt that action research was not appropriate due to the size of the
organisation, and the fact that there were multiple locations involved. It would not have been possible to take an active part in the many sub-teams that were involved in the organisation. Overall the case study approach was the most suitable. Another important factor was that a case company about to commence a transition to agile methods was available and agreed to facilitate the research as part of their own monitoring of the progress of the agile transition. Yin (2003) proposed five components of case studies:

1. A study's questions,
2. Its propositions, if any,
3. Its unit(s) of analysis,
4. The logic linking the data to the propositions, and
5. The criteria for interpreting the findings

The unit of analysis in a case study can be, for example, an individual, a community, an organisation, a team or a document. The main units must be at the same level as the study questions & typically comparable to those previously studied. An embedded case study is a case study containing more than one sub-unit of analysis (Yin 2003). An embedded study may include main & smaller units on different levels but looking for consistent patterns of evidence across units, but within a case. With regard to research question 1, the primary unit of analysis is the developer. Developers can be thought of as embedded units of analysis within the team unit. Because the primary unit of analysis was the developer, this had a bearing on the data collection and the data analysis and is discussed further in 2.4.2, 2.4.3 and 2.4.5. With respect to research question 2, the unit of analysis was Java source code.

2.4 Data Collection

2.4.1 Overview of Data Generation Techniques

A data collection method or data generation method is a way of producing empirical (field) data or evidence. Data can be quantitative or qualitative. Quantitative data is numeric data, such as, number of employees or number of bugs in a piece of code. Qualitative data is all other types of data such as words, images etc. (Oates 2007). Within case studies, a number of data collection methods can be used such as interviews, observations, and documents as shown in Figure 14.

Interviews are conversations between people. The interviews can be one-to-one or group interviews and are usually guided by a set of questions. They can vary in type from
structured where the interviewer rigidly follows a set of questions to semi-structured where a set of questions are used but it is possible to change the order of the questions or add in other questions. Finally with unstructured interviews, the interviewer introduces the topic and the interviewees develop their ideas with minimal interruption by the interviewer (Oates 2007).

Questionnaires involve a set of questions assembled in a pre-defined order where the respondents answer often via multiple choice or Likert Scale.

In a workshop a group of people works together to solve a task.

Documents could be minutes of meetings, policy documents and in software development, for example, the process artifacts are documents, such as the code.

### 2.4.2 Selection of Data Collection Methods

Within the case study a number of data collection techniques were used, namely, interviews, surveys and documents. Table 4 maps the data collection into four stages D1, D2, D3 and D4. It shows the research question that the data collection relates to, the collection methods and the chapter in which the resulting data is presented and evaluated.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Research Question</th>
<th>Collection Method</th>
<th>Evaluation Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>1</td>
<td>Interview</td>
<td>3</td>
</tr>
<tr>
<td>D2</td>
<td>1</td>
<td>Survey</td>
<td>4</td>
</tr>
<tr>
<td>D3</td>
<td>1</td>
<td>Interview</td>
<td>5,6</td>
</tr>
<tr>
<td>D4</td>
<td>2</td>
<td>Documents</td>
<td>7</td>
</tr>
</tbody>
</table>

In relation to Research Question 1, the goal was to monitor how the concerns and expectations of the developers changed over time. This required at least two stages of data collection. Data had to be gathered at the very early stage of the agile adoption process and then data was gathered 9-12 months into the adoption process. Research Question 2 involved evaluating the code produced so copies of the source code were obtained from the code repository. Figure 15 shows the data collection stages in context of an overall timeframe.
2.4.3 Data Collection Stage – D1

Method Selection
The main objective of this first stage of the research was to capture the initial reaction of developers when faced with adopting an agile methodology. This data was collected at the very beginning of the agile adoption process. The level of exposure of the developers to Agile methods at the time ranged from none to a 1-day course, a 5-day course, up to having worked for a maximum of 5 or 6 weeks in a team piloting agile methods. The goal was to determine “What were the concerns and expectations of the developers with respect to adopting agile methods? What did they expect from agile methods? What aspects of agile methods and its adoption were they concerned about?” Did they feel they moving to agile methods would result in a better work environment?”

Given the objectives, semi-structured one-to-one interviews were selected as the first mechanism to collect the data at the beginning of the agile adoption process. This was also consistent with the fact that the primary unit of analysis for the study was the developer, as discussed in 2.3. A guideline questionnaire was developed. Because the interview type was semi-structured, as the interviewer, I could depart from the guide to ask follow-up questions when interesting point were raised during the interviews. In that way semi-structured interviews provided more flexibility and allowed the interviewees to speak with more detail on the issues raised and introduce issues of their own that they thought relevant to the theme.
The interviews were all face-to-face and conducted at the site of the case company in Ireland. The interviews were conducted back-to-back in blocks of 5 to 8 interviews at a time. This approach was taken in order to minimise the scheduling effort on behalf of the case company.

The participants gave their full permission for their interviews to be voice-recorded and were aware of the purpose of the interviews. Each interview lasted between 15 and 30 minutes. The interviews were then transcribed line-by-line for analysis.

Selecting the interviewees

The case company itself nominated the interviewees based on our request that it was mainly people that had attended some agile training courses and seminars and that were soon moving to an agile way of working or developers who had just start to work with agile methods. All the interviewees were based in Ireland. As it was in the interest of the cases company to obtain a true reflection of the reality, the fact that the company selected the participants was not seen as risk to the validity of the study. They had no motivation to nominate individuals that had a particular bias towards the adoption of agile methods or otherwise. Also since individuals were selected by a number of different line managers, the selection can be seen to be random to a certain degree. And before participating in a study such as this, individuals needed to have their participation cleared with their line manager first due to issues such as company confidentially.

Formulating the questions

According to Edberg and Anderson (2006) managing software process improvement initiatives “is similar to managing any other form of profound organizational change and requires broader understanding of the effects of organizational culture and politics on that change.” According to Senge et al. (1999), if we believe that organisations are living organisms capable of learning and growth, we should stop asking why people resist change and instead attempt to understand the natural response of the enterprise, given its traditions, norms and assumptions. Research by Baddoo and Hall (2003) indicates that the primary reason people are willing to change software processes is evidence of the benefits of the change. Other reasons (Baddoo and Hall 2002) are dependent upon the role of the person within the organisation but include such factors as

- Visible senior management support, input into the type of process changes, and sufficient resources available to maintain the initiative (for software developers);
• Ability to make decisions about the changing processes, sufficient resources available to maintain the initiative, and ability to change processes as needed (for project managers); and
• Ability to meet financial/managerial targets more effectively and relative cost benefits of the initiatives (for senior managers).

A key issue with the reasons above is that most are dependent upon clear evidence of the benefits of the software improvement initiative.

We posit that there are three board reactions that an organisation can exhibit towards a planned change:
• Resist the change
• Treat the change with acceptance
• Treat the change with indifference

From the literature mentioned above we can conclude that developers might accept the change of methodology if they feel it will offer improvements over their existing methodology. They could resist the change if they are of the opinion that it is inferior to their current process. And if they may treat the change with indifference if they are of the belief that it will result in a similar advantages and disadvantage to their current process. Therefore we formulated the following statements that reflect the three reactions defined above.

1. "Agile methods will be a much better way of working"
2. "Our existing process is adequate. We should not adopt agile methods."
3. "Agile methods will not make much of a difference to the day-today work."

Ultimately we were trying to gauge if the developers felt that the agile methodology would provide more benefits than their existing methodology. One aspect to focus on was process non-compliances in the existing process. Based on personal experience if a step in a process is constantly skipped or results in a non-compliance it generally means that it is not feasible to execute that step (based on time or other constraints) or that the particular step is not adding value. We then formulated the following themes for the interviews.

1. Is the current software design processes/methodology adhered to?
2. What are the issues or problem areas in the current processes?
3. Will Agile methods be an improvement on methods currently used? Will it solve some of the problems that exist in the current process?
4. Are they aspects of Agile that will not work?
5. What do developers think of the various XP and Scrum practices, particularly those that are seen to be challenging developers to work in a different way?

6. Will the use of Agile methods make the day-to-day work more or less enjoyable?

The complete questionnaire can be found in Appendix A. Details of the questions and the reasons that they were included are given here.

Questions 1 to 3 are not related to any of the themes listed above, but are used to capture some specifics about the interviewees.

**Question 1 How long have you worked in software design?**

| <1 yr | 1-3 yrs | 3-5 yrs | > 5 yrs |

This question was included to capture data on the experience level of the interviewee. Viewpoints could possibly be dependant on experience level. However as the case company nominated the participants, there was no attempt on our part to select participants of varying experience levels.

**Question.2 How long have you been employed by this company?**

| <1 yr | 1-3 yrs | 3-5 yrs | > 5 yrs |

Because the company had a long history in software development and process improvement and document heavy processes, employees that had worked in the organisation for a longer period of time may have had different opinions to employees that were new to the company.

**Question.3 What is your role in the current project (e.g design, test, systems etc.)?**

This question was included in order to capture the role or function of the interviewee. Again this was captured to allow the possibility for correlation at a later stage.

**Question.4 Describe the process that you currently follow for your part of the project lifecycle.**

The interviewees were asked to describe the process that they followed. Some project teams might have used slightly different variations and interpretations of the organisation wide process. Sometime developers may not have been aware of a particular aspect or step in
the process. This could be as a result of non-adherence to the process or a lack of visibility of the process. When answering this question, developers also had a chance to think about the existing process before talking about any issues or problems later in Question 6. Questions 4 to 6 are related to theme 2, which is concerned with problems or issues with the traditional process. These questions are open questions.

**Question 5** *Is the process strictly adhered to?*

Developers were asked if in their opinion the process was adhered to. Sometimes a non-adherence or non-compliance would indicate that there is a problem with the process. For example if a particular activity is always skipped, this could imply that it is not feasible for some reason to carry out the activity as defined by the process.

**Question 6** *In your opinion, what are the (2 biggest) problems that you see with the existing process and how do you think they can be addressed?*

After describing the current process and the level of adherence as viewed by the developers, they were then asked to prioritise the two biggest problems and how they could be solved. The reason for asking how they could be solved was to see if they felt that Agile methods could be an answer.

**Question 7** *What is your level of knowledge of Agile methods?*

- a) informal presentations/articles
- b) formal training
- c) used Agile methods previously

Question 7 was included in order to capture details of the developer’s level of experience with regard to agile methods. Most developers would only have attended some training, but there was a possibility that they could have used agile methods in another company or organisation that would have influenced their opinions.

**Question 8** *Do you agree with the introduction of Agile practices? If not why not?*

Question 8 and Question 9 are based on theme 3, which is to see if the introduction of agile methods will solve any of the issues that developers feel exist in the current/traditional process.
**Question.9** Do you think that the introduction of Agile methods will solve any of the main issues you see in the current processes? ref Q.6

**Question.10** Agile methods introduce concepts such as pair programming, test driven development, collective ownership. What is your view on those aspects?

This question is related to theme 5. Agile methods aim to empower development teams. They also emphasise the need for communication within the development team. Some of the ways of promoting this is, for example, pair programming (Larman 2004) and Scrum meetings (Schwaber and Beedle 2002). Test Driven Development is an important core activity of agile software engineering. But these are challenging practices and mark a radical difference is the way of working in comparison with traditional methods. So this question was included to provoke developers to think about some of the parts of agile that might push them out of their current comfort zone.

**Question.11** Are there any particular aspects of Agile methods that you are apprehensive about?

Question 11 is based on theme 4. Its aim is to see if developers feel that there are aspects of agile that are not applicable or suitable for their organisation. The key to making agile work would appear to be not the blind adoption of all the techniques and practices, but rather integrating the practices most suited to the environment and the project in question.

**Question.12** Do you feel that your day-to-day work will be more enjoyable as a result of a move towards Agile methods?

This final question is a straightforward one that aims to judge the developers level of enthusiasm for the adoption of agile methods.

More details on the methodology of how the interviews were coded is contained in Section 2.5 and the results and findings based on this data is documented in Chapter 3.

### 2.4.4 Data Collection Stage – D2

**Method Selection**

Several major themes were captured from the semi-structured interviews with the developers. Based on the themes that are discussed in detail in Chapter 3, we decided to use a
questionnaire with pre-defined answer ranges to validate the themes and opinions that had emerged. It was also possible to sample a larger percentage of the population under study with a questionnaire. A questionnaire using closed questions was devised and completed by 27 developers. The questionnaire was distributed via e-mail and again the case company selected the participants. All participants were based in Ireland.

**Formulating the Questions**

The questionnaire had two questions that were related to factual data (experience levels). The rest of the questions were developer opinions where the Likert Scale was used in most cases to indicate a degree of agreement or disagreement. For most of the questions a 5 point Likert Scale was used, which ranged from “Agree Strongly”, ”Agree”, “Neither agree or disagree”, “Disagree” to “Disagree Strongly”. An odd number of choices were used. One disadvantage of an odd number is that is does not force the respondent into taking a stand on the matter. It could be convenient to just choose the middle one without contemplating the alternatives before answering. However, some respondents might not be comfortable by having to take a stance, especially since they could be relatively “new” to agile methods. As an example, one of the themes that emerged from the interviews was that “The existing process was document driven”. So to quantify the extent of this opinion, the following is an example of the related question that was included in the questionnaire.

**Table 5: Sample Question**

Q.3 “Before introducing agile, the design process we are using is a document driven process.”

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
</table>

The complete questionnaire is available in Appendix B.

**2.4.5 Data Collection Stage – D3**

**Method Selection**

As the developers gained experience and practiced the agile methodology the objective was to gather information on the following themes:

- Which of their initial concerns (from analysis of data from stages D1 and D2) were valid?
- Which of their concerns had materialised to a lesser or greater extent and why?
- Which concerns didn’t materialise at all?
- Were their expectations met to a lesser or greater extent?
- Which of their expectations turned out to be false and why?
- Did new concerns materialise during the adoption phase?
- Did agile produce results or effects not initially anticipated?

To allow new concerns, expectations and results to emerge, we decided to use semi-structured interviews again at this stage. Seven interviews were conducted on a one-to-one basis with developers. The case company also wished to gather data on a per team basis so nine group interviews were used on a per team basis. The teams interviewed were based in Ireland and Sweden. Group interviews were found to be suitable in that it was possible for the group to interact during the interview that allowed new insights to emerge. They also helped generated consensus among the team. The team view was necessary and useful when investigating if there was any link between the views of the developers and the characteristics of the code produced by those developers. The interviews were all face-to-face and conducted at the site of the case company in both Ireland and Sweden. Later in the data analysis, the individual interviews and the group interviews are analysed separately. This is to avoid any risk of the outcome being difference when developers were interviewed individually compared to group interviews. This is important since the basic unit of analysis for the study related to research question 1 is the developer.

**Formulating the questions**

An example of how the questions for the interviews were formulated is given here. When the developers were interviewed initially (D1), an expectation that emerged was that “Agile would result in less documentation”. To see if this expectation was realised, the following question was included on the guideline questionnaire for the semi-structured interviews. The complete interview guideline questions can be found in Appendix C.

*Question 8. Has the introduction of agile resulted in less documentation? Are there documents produced now that are not being used? If answer is yes, are there any adverse consequences? And are there any problems with knowledge transfer or system maintenance?*

The results and the discussions of the findings are detailed in Chapter 5.
2.4.6 Data Collection Stage – D4

The data collection here was in relation to analysing the code, so documents as in the source code was the data source. Code from the version controlled ClearCase repository was made available, so it was possible to obtain copies of code that was produced using traditional methods and copies of the code that was produced using agile methods.

One way to determine if the code quality had improved is to look at the number of bug reports it produces. But since other variables would have to be factored in, such as, the amount of testing performed, the quality of the testing, the number of customers that had received the code etc. that could impact the result, it was decided to look at the internal quality of the code itself as a measurement. The methods selected to evaluate the source code is described in more detail later in this chapter and the results and the discussions of the findings are documented in Chapter 7.

2.5 Data Analysis

After the data has been generated it must be analysed. “Quantitative data analysis uses mathematical approaches such as statistics to examine and interpret the data. Qualitative data analysis looks for themes and categories within the words people use or the images they create.” (Oates 2007). Grounded Theory is a qualitative research method that promotes the generation of theory from data, originally proposed by Glaser and Strauss (1967). Concepts from grounded theory (Glaser 1978; Glaser 1992; Glaser and Strauss 1976) were used to analyse the qualitative data from the interviews. We now discuss Grounded Theory in general. This is followed by a discussion on how principles from Grounded Theory was used to analyse the interview data in relation to research question 1 (What is the effect of introducing agile methods on the concerns and expectations of the developers?)

Finally we describe how the source code was analysed and characterised for research question 2 (What is the effect of introducing agile methods on the characteristics of the software produced?)

2.5.1 Grounded Theory

Glaser (2002) explains that grounded theory is “the generation of emergent conceptualizations into integrated patterns, which are denoted by categories and their properties.” Grounded Theory is a qualitative research method that promotes the generation of theory from data (Glaser and Strauss 1976). The theory is said to ‘emerge’ from the analysis of data obtained from interviews, observation sessions, etc., and is, therefore,
‘grounded’ in reality. Glaser further explains that grounded theory is “the generation of emergent conceptualizations into integrated patterns, which are denoted by categories and their properties” (Glaser 2002). In Grounded Theory, data collection and data analysis are interrelated processes. Once the first piece of data has been collected, data analysis can begin. With grounded theory the interview data must be coded. Coding allows for interpretation and analysis of the data. Strauss and Corbin (1990) outline three types of coding: open, axial and selective coding.

**Open coding:** the initial process of labelling units of data, based on terms and concepts found in the data.

**Axial coding:** as a list of codes begins to emerge, the researcher moves to a higher or more abstract level of analysis and looks for relationships between the codes. It may be that some codes can be incorporated under broader headings.

**Selective coding:** the researcher focuses on the core codes – those that have emerged as being vital for a theory of the phenomenon being investigated.

Analysis involves a “constant comparative method” - as new codes, categories or concepts identified in the data, previously coded material is revisited to see if it can be better coded.

As categories and concepts emerge from the data, the researcher determines what data to collect next. This is known as **theoretical sampling**. Theoretical sampling as defined by Glaser (1978) “is the process of data collection for generating theory whereby the analyst jointly collects, codes, and analyses data and decides what data to collect next and where to find them, in order to develop . . . theory as it emerges.” The researcher starts with one person or instance and generates the data, analyses it and based on what emerges from the data, they decide what to look at next. The process data generation -> data analysis -> data generation continues indefinitely. It only finishes when it becomes clear that further data no longer triggers new modifications to the data and emerging theory. That is when is reaches the point of **theoretical saturation** (Oates 2007).

### 2.5.2 Why a Grounded Theory Approach is Suitable

The goal of grounded theory is to develop theories (Robson 2002). In this research the goal was to allow themes to emerge from the data rather than test out existing theories. The theory is said to ‘emerge’ from the analysis of data obtained from interviews, observation sessions, etc., and is, therefore, ‘grounded’ in reality. The interviews generate a large amount of data and grounded theory provides valuable tools to analyse what can be an overwhelming amount
of qualitative data (Robson 2002). Oates (2007) warns “researchers claim that they are using grounded theory to justify sloppy research.” Creswell states “the qualitative study approach is considered an appropriate method when little is known about the phenomenon under investigation and the concepts are immature due to lack of theory and previous research and a need exists to explore and describe the phenomena” (Creswell 1994). Qualitative research methods are “designed to help researchers understand people and the social and cultural contexts within which they live” (Myers 1997). This part of our research was in the context of a people-aspect of software development and Duvall (1995) and Lings and Lundell (2005) have identified the grounded theory approach as appropriate for studying the socio technical aspects of software development. A number of recent studies in the area of agile software development research have employed grounded theory:
  
  o Whitworth and Biddle used the grounded theory approach to investigate “motivation and cohesion in agile teams” (2007a) and “the social nature of agile teams” (2007b),
  o Hoda et al. (2010a) performed a grounded theory study on customer collaboration in agile teams.
  o Grounded theory research was performed by Dorairaj et al. (2010) on “understanding the importance of trust in distributed agile projects”
  o The Customer Role in XP teams was investigated by Martin et al. (2009) using a grounded theory analysis.
  o Brown et al. (2008) used grounded theory to analyse processes captured in video data in “a study of collaboration between user interaction designers and developers”
  o Ferreira et al. (2007) researched “the integration of UI design into agile development” using a qualitative grounded theory study based on interviews.

In our research we used some principles from Grounded Theory to analyse the data from data collection stages D1 and D3. Open coding and axial coding were used in conjunction with the constant comparative method of analysis. We did not employ theoretical sampling and theoretical saturation techniques. First of all, we were approaching the research without an abundance of pre-established theories, which meant a qualitative approach was suitable. During the literature review we found a number of studies that documented the benefits of using agile methods (Williams et al. 2000; Schatz and Abdelshafi 2005; Barnett and Schwaber 2004; Barnett 2006; Law and Charron 2005;Kuppuswame et al. 2003). Other studies contained stories of agile adoption and documented various pitfalls (Goos and Mellisse 2008; Sureshchandra and Shrinivasavadhani 2008). And other studies concentrated on the job satisfaction and psychological experience on individuals in large teams. However
in the public domain at least we did not find a longitudinal study that focussed on the change in concerns and expectations of the developers over time. Our goal was to build a rich and broad picture of the changing concerns and expectations of the developers over time, thus capturing as many concerns and expectations as possible. Theoretical sampling and theoretical saturation in all core categories was not attempted to avoid the risk of certain themes in the being overlooked in favour of following a smaller number of specific theories to theoretical saturation. While at the same time open coding and axial coding of the emerging themes (concerns and expectations) ensured that they were grounded in the empirical data.

2.5.3 Using Metrics to Evaluate the Code

Software Metrics is a collective term used to describe the very wide range of activities concerned with measurement in software engineering. Metrics can be grouped into two broad classifications. **Software product metrics:** These metrics measure the software product (documents and code) at any stage of its development. They are often classified according with the size, complexity, quality and data dependency.

**Software process metrics:** These metrics measure the process in regards to the time that the project will take, cost, and methodology followed and how the experience of the team members can affect these values.

There are numerous software metrics that can be applied and many tools available to generate metrics. The objective of this part of the research was to find a way to use metrics to characterise and evaluate the source code so the product metrics were relevant.

The history of software metrics dates back to the 1960’s when the Lines of Code metric was used as the basis for measuring programming productivity and effort. The first book dedicated to software metrics was published in 1976 (Gilb 1976). The Lines of Code (LOC or KLOC) metric was the first metric and it was used initially as a basis for measuring programmer productivity. Later it was also used indirectly as a basis for measuring program quality as defects per KLOC.

Software product metrics measure the characteristics of the software and are often classified according with the size, complexity and data dependency. Software metrics can be used to reflect on two main aspects of a system

1. Size and complexity
2. Coupling and cohesion
Some software metrics that are often applied during the design phase are McCabe’s Cyclomatic number, Information Flow (fan in/out), Function Points and the Bang metric by Tom DeMarco.

**McCabe’s Cyclomatic number:** The Cyclomatic Complexity – \( v(G) \) was proposed by McCabe (1976). McCabe suggests viewing the program as a graph, and then finding out the number of different paths through it. The program is represented as a control flow graph such as a directed graph. A directed graph consists of nodes or vertices connected by edges. An example of a directed graph is shown in figure 2.3. In terms of a software program, a vertex would be a statement and an edge would be A-C, if control flows from A to C.

![Directed Graph](Fig. 16. Directed Graph (Fenton and Pfleeger 1998))

Vertices: A, B, C, D, E, F  
Edges: (A, C), (A, E), (B, C), (B, F), (C, D), (D, A), (E, F)

\[ v(G) = E - N + 2 \]

where \( v(G) \) is the cyclomatic complexity  
E is the number of edges  
N is the number of nodes

The cyclomatic number \( v(G) \) is defined in Figure 16. Figure 17 shows a code example with its corresponding directed graph. The graph contains 8 edges and 7 nodes, therefore its cyclomatic number is \( 8 - 7 + 2 = 3 \). Intuitively more complex the code results in a higher cyclomatic value.
**Function Points** - The function point metric developed by Albrect and Gaffney (1983) has been used successfully to measure software size and determine quality metrics. It measures the functionality of the software product based on the following parameters: user inputs, user outputs, user inquiries, number of files and the number of external interfaces.

**Information Flow (Fan in, Fan out)** - Information Flow is a metric from Kafura and Reddy (1987) to measure the complexity of a software program. This metric identifies the number of calls to a module (i.e. the flows of local information entering: fan-in) and the number of calls from a module (i.e. the flows of local information leaving: fan-out).

**The Bang Metric** - The Bang metric, first described by De Marco (Chidamber et al. 1998; Rosenberg 1998) measures the size of the project based on the functionality of the system detected during design. It relies on diagrams generated during the design (data dictionary, entity relationship, data flow and state transition) to give the functional entities to count. It is relies on non-code artifacts, so it is not the most suitable for use in projects using agile methods.

Chidamber and Kemerer published a suite of metrics (CK metric suite) specifically for Object Oriented software (Chidamber and Kemerer 1994). The CK metric suite contains the following metrics:

**Weighted Methods per Class (WMC)** – the sum of the complexity of the methods in a class. WMC is a predictor of how much time and effort is needed to develop and maintain the class. Classes with large number of methods are viewed as having limited re-use.

**Depth of Inheritance Tree (DIT)** – The maximum length from the node to the root of the tree. A small value for DIT may indicate that inheritance is not being used.
**Number Of Children** (NOC) – Number of immediate subclasses subordinate to a class in the class hierarchy.

**Coupling Between Objects** (CBO) – This is a count of the number of classes to which it is coupled. Small values of CBO improve modularity and encapsulation, thus making it easier to test and maintain a class.

**Response For a Class** (RFC) – This is the number of methods of the class plus the number of methods called by any of those methods. If a large number of methods are invoked from a class, testing and maintenance of the class becomes more complex.

**Lack of Cohesion in Methods** (LCOM) – Measures the dissimilarity of methods in a class via instanced variables. A higher value would indicate higher complexity.

With regard to thresholds for the metrics, some literature (Kafura and Reddy 1987; Rosenberg 1998; Kan 2003) recommends that thresholds are derived locally and that 80th and 20th percentiles should be used to determine high and low values of the metrics. The Chidamber and Kemerer (CK) suite of quality metrics has been widely used and validated in the literature. Many studies show that the maintenance effort and the fault proneness of classes can be predicted using the CK metrics. Table 6 taken from an empirical study of CK metrics by Subramanyan and Krishnan (2003) summarises the key literature linking the CK metrics to maintenance effort and fault proneness.

Table 6: Summary of literature on CK metrics (Subramanyan and Krishnan 2003)

<table>
<thead>
<tr>
<th>Study</th>
<th>Dependent variable</th>
<th>CK Metrics tested</th>
<th>Size controlled For?</th>
<th>Summary of Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Li and Henry 1993</td>
<td>Maintenance Code change</td>
<td>All metrics in CK suite</td>
<td>Yes</td>
<td>Two commercial systems studied. Five of the six (except CBO) helped predict maintenance effort.</td>
</tr>
<tr>
<td>Basili et al. 1996</td>
<td>Fault proneness (presence/absence of a fault in a class)</td>
<td>All metrics in CK suite</td>
<td>No</td>
<td>Eight student projects studies. WMC, CBO, DIT, NOT and RFC correlated with defects.</td>
</tr>
<tr>
<td>Binkley and Schach 1998</td>
<td>Maintenance Code change</td>
<td>Class coupling, NOC</td>
<td>No</td>
<td>Two of four systems studies were developed with OO methods. Coupling measures were correlated with maintenance code changes due to field faults, but not NOC.</td>
</tr>
<tr>
<td>Study</td>
<td>Type</td>
<td>Metrics</td>
<td>Confound</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------</td>
<td>--------------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Briand et al. 1999</td>
<td>Fault proneness</td>
<td>CBO, RFC, LCOM</td>
<td>No</td>
<td>An industrial case study was performed and the three CK metrics were found to be associated with fault proneness</td>
</tr>
<tr>
<td>Tang et al. 1999</td>
<td>Fault proneness</td>
<td>WMC, RFC</td>
<td>No</td>
<td>Three real time systems were analysed for testing and maintenance defects. Higher WMC and RFC were found to be associated with fault proneness</td>
</tr>
<tr>
<td>Briand et al. 2000</td>
<td>Fault proneness</td>
<td>All metrics in CK suite</td>
<td>No</td>
<td>Eight student projects were studied. Classes with higher WMC, CBO, DIT and RFC were more fault prone while classes with more children (NOC) were less fault prone. LCOM was not associated with defects.</td>
</tr>
<tr>
<td>Cartwright and Shepperd 2000</td>
<td>Defect density (Testing and post-release defects per line of code)</td>
<td>DIT, NOC</td>
<td>No</td>
<td>A medium sized telecommunication system was studied. Both DIT and NOC were found to influence detect density.</td>
</tr>
<tr>
<td>El Emam et al. 2001</td>
<td>Fault proneness</td>
<td>All metrics in CK suite</td>
<td>Yes</td>
<td>A large telecommunications application was studied. Size was found to confound the effect of all metrics on the fault proneness.</td>
</tr>
</tbody>
</table>

Li and Henry (1993) investigated the relationship between several OO design metrics (including CK metrics) and the extent of code change in product maintenance effort. Binkley and Schach (1998) found that the coupling measures (including CBO and NOC from the CK suite) could be linked with maintenance changes made in classes due to field failures. Basili et al. (1996) show that several of the CK metrics were associated with fault proneness of classes. Chidamber et al. (1998) found that higher values of the coupling and the cohesion metrics in the CK suite were associated with reduced productivity and increased rework/design effort. According to Cartwright and Shepperd (2000), based on the analysis of a medium-sized telecommunication system, the inheritance measures from the CK suite (DIT, NOC) were associated with defect density of classes. Similarly, studies on CK metrics by Briand et al. (2000, 1999) and Tang et al. (1999) show that several design metrics from the CK suite were associated with fault proneness of classes. Most of the studies to this point had not taken into account the class sizes until El Emam et al. (2001) examined a large C++ telecommunication application and provided evidence for the argument that the size of the software may confound the effect of most OO design metrics on defects. They argue that,
after controlling for the size of the software, the effects of most CK metrics (except for coupling and inheritance metrics) on defect proneness are not significant.

From the literature discussed above it is clear that there is a link between some of the CK metrics (coupling and inheritance) and the fault proneness. However using metrics values in isolation doesn’t clearly characterise a system. The size of the system must be accounted for in some way. Knowing a system has 500 classes does not tell us how large a system is because the classes could be very small or very large. We do not know how complex a system is if we know that it has 5,000 lines of code. However if we know that a system has 100 classes and 1000 lines of code, then we can deduce that each class have on average 10 lines per class. If we could compare the value 10 with some baseline or threshold value then would provide us with a better indicator.

The Overview Pyramid introduced Lanza and Marinescu (2006) is a simple way to characterise a design based on some well-known metrics, such as, the Cyclomatic Number defined by McCabe (1976) and the CK metric Weighted Method Count (WMC) (Chidamber and Kemerer 1994).

The Overview Pyramid also provides thresholds for low, standard and high values. The values were derived from a number of industrial and open source projects.

Software visualisation can be divided into two areas: program visualisation and algorithm visualisation (Stasko et al. 1998). Program visualisation techniques provide views of actual program code or data structure in either static or dynamic form. Lanza and Marinescu, in the Overview Pyramid produced by the iPlasma tool (LOOSE Research Group), provide static code visualisation by visualising object oriented source code using information that can be statically extracted from the source code. The reverse engineering research community has used static code visualisation over the past decade or two (Stasko et al. 1998; Storey et al. 1997; Storey et al. 1998; Best et al.).

A detection strategy is a composed logical condition, based on metrics by which design fragments with specific properties are detected in the source code (Lanza and Marinescu 2006). In the software engineering community many authors have been concerned with identifying and formulating design principles, rules and heuristics (Meyer 1998; Liskov 1987; Martin 2002; Coad and Yourdon 2001; Riel 1996; Johnson and Foote 1998). An alternative approach used by Fowler et al. (1999) in their book on refactorings and by the “anti-pattern” community (Brooke et al. 1998) is to identify situations where the design must be improved. Fowler (1999) describes twenty code smells that are symptoms of bad design. Lanza and Marinescu also use detection strategies to detect design flaws.
Because the Overview Pyramid is grounded in proven metrics, uses detection strategies and also has a visual aspect to it, we felt it would be suitable for our task of comparing the characteristics of two software designs. More detail on the use of the metrics and the detection strategies is provided in Chapter 6.

2.5.4 Reliability and Validity of the Study

There are many factors that can threaten the validity and reliability of qualitative data. Perhaps the biggest challenge is the elimination of researcher bias. It is important that it is the “participant’s perspective on the social phenomenon of interest that should unfold as the participant views it and not as the researcher views it” (Marshall and Rossmann 1989).

The following outlines the possible identified risks to the validity of this study and discusses the measures taken to address them and therefore increase the accuracy of the data.

**Researcher bias:** In this chapter we already discussed why some of the principles from Grounded Theory such as open coding and axial coding were suitable in this study. However it is important that these grounded theory based methods are applied rigorously to eliminate researcher bias and ensure the theories are actually “grounded”. According to Glaser (2002), findings that are grounded will also necessarily be valid. Glaser (1978) proposes two criteria against which the quality of an emergent theory should be evaluated. The first is that the theory should fit the data, which means that the categories should emerge from the data - without the researcher forcing data to fit conceptual categories that can not be found in the data. The second criterion is that the results explain the core problems and processes in the relevant area, which may be useful for practitioners in that area. In our study Qualitative Data Analysis Software, Weft QDA, (Weft QDA) was used to support the grounded theory coding of the interviews. Figure 18 provides an example of the output from weft QDA. The codes arising out of each interview were constantly compared against the codes from the same interview, and those from other interviews. This ensured that the emerging theory was “grounded”. This is Grounded Theory’s “constant comparison method” (Glaser 1992; Glaser and Strauss 1976).
Small or unrepresentative sample: The choice of research participants is extremely important as the views of the participants are then taken to reflect the organisation. When researching in an industrial setting, the choice of participants will always be limited in some way by our access to them. In this study the sample size was sufficiently large to characterise the context under study. Eighteen developers interviewed initially at the start of the agile adoption process and the second stage of interviews consisted of seven one-to-one interviews and nine team interviews. The majority of participants were experienced with approximately 75% of participants with greater than five years experience. Other grounded theory researchers have used as little as seven participants in their studies (Dorairaj et al. 2010), while some studies used up to thirty participants (Hoda et al. 2010b). According to Martin and Turner (1986), “By the time three or four sets of data have been analysed, the majority of useful concepts will have been discovered”. The participants for the interviews at the beginning of the agile adoption process were based in Ireland while the later interviews consisted of 7 individual interviews based in Ireland and nine group based interviews (four in Ireland and five in Sweden. However the sample size was large in both locations to provide a good reflection of the overall organisation.

Use of both individual interviews and group interviews: Initially interviews were individual interviews and in Data Collection D3 both individual and group interviews were used. The unit of analysis in this case study with respect to research question 1 is the developer. Therefore by using group interviews where the group is made up of members of the same team, there was a risk that the data coming from the group interviews might be
different than if the developers were interviewed individually. Therefore the data from the individual interviews and the group interviews is analysed separately in chapter 5.

**Inaccurate recording of data:** The interviews were voice recorded to ensure accuracy and completeness of the data. The interviews were then transcribed.

**Using interviews only:** The initial interviews were followed with a survey. This ensured that the theories that emerged from the interviews were valid. Also the results of this study were presented and accepted at international agile development conferences.

**Choice of method used to characterise the code:** When characterising the code, the method used was based on metrics and strategies that were well validated in the literature. This has been described in 2.5.3.

### 2.5.5 The Researcher

When using qualitative analysis, it is extremely important that it is the “*participant’s perspective on the social phenomenon of interest that should unfold as the participant views it and not as the researcher views it*” (Marshall and Rossmann 1989). The professional experience and educational background of the researcher can undoubtedly influence their interpretation of the data; therefore I have included a history of my own education and work experience here and some opinions that I think are relevant.

My primary degree was a Bachelor of Engineering in Electrical and Electronic Engineering (1st class honors) from the National University of Ireland Galway. The topics covered on this course would have included basic science and engineering modules as well as the fundamentals of analog and digital electronics. My final year project was in the area of digital signal processing. Software engineering was not the key focus of the course, but a number of projects were completed using Fortran, Turbo Pascal and assembly language.

After graduation I worked in a multi-national software development organisation for fourteen years between September 1988 and July 2002. Between 1988 and 1995 I worked in various roles such as developer, tester and systems engineer. The application domain was telecommunications. At this time the development was based mainly on proprietary platforms, languages, processes, methods and tools. The primary product being developed had a very well defined modular architecture. A waterfall process model was used and the process was very well defined and documented. Unit test and function test tools were available as well as excellent tracing and debugging facilities. From 1995 onwards, the company migrated towards the use of open source platforms with Java and C or C++ as the
main development languages. At the early phase of this migration as a developer and as a project manager I experienced many problems such as lack of ability to trace and debug, lack of test tools, problems with software upgrade. Open source tools such as the xUnit testing frameworks (Fowler n.d.) were not available initially. These were facilities that we had taken for granted in the proprietary system, so I learned to appreciate the value of well-documented processes and methods and in particular the need for testing tools.

Between 1995 and 1998, I worked as a project manager. I completed some fundamental project management courses and leadership training. In 1996, the organisation set a goal to achieve Level 2 on the CMM (now known as CMMI) process maturity ranking scale. CMMI is a process maturity framework that helps organisations improve their software process through a set of recommended practices in a number of key process areas (SEI n.d). It has a scale from 1 to 5 in terms of process maturity level. Level 2 includes practices under the key process areas of Software Project Planning, Software Project Tracking, Software Quality Assurance, Requirements Management and Configuration Management. The sub-project that I was responsible at the time had 45 developers and testers and was one of the sub-projects selected for examination by the CMM auditors. The conclusion by the CMM auditors was that we were in fact operating at CMM Level 3. The main difference between CMM Level 2 and Level 3 is that at Level 3 there is evidence of continuous process improvement.

Between 1998 and 2002, my role as Systems Manager involved writing and reviewing Requirement Specifications, reviewing all early design documentation and approving System impacts, managing small feasibility studies and working with the Project Managers to specify the cost and scope of new projects. During this time I observed that a lot of re-work was caused due to the fact that the requirements were specified well in advance of the work being performed. A lot of time was spent planning projects that could not start for some time because the developers were still completing the previous project.


Since 2002, I have been employed as a lecturer in the Computer and Software Engineering Department at Athlone Institute of Technology. The main modules I teach are Web Technology, Internet and Multimedia Systems and Software Engineering. For the past two years I have delivered software engineering courses with a focus on XP and Scrum at South East China University in China. I have been a member of the Agile Software development
community for the past three years and have attended the XP2009 conference and attended and presented at the XP2010 and Chaos and Agile in Action (CAIA) 2009.
3 Initial Concerns and Expectations of the Developers
This chapter documents the analysis of the first stage of the research. The data collection for this first stage is referred to as “Data Collection Stage 1” or “D1” in Chapter 2. The main objective of this first stage of the research was to capture the initial reaction of developers when faced with adopting an agile methodology. It was vital to capture the sentiments of the developers before they became “tainted by reality.” For example if a developer had a positive experience with agile, they may very easily “forget” their initial apprehension, concerns and reservations. Conversely if a developer had less than satisfactory brush with agile, they may equally forget their initial enthusiasm and optimism. The goal was to determine “What did the developers really think about adopting agile? Were they skeptical, optimistic or something else? What did they expect from agile? What were their concerns about agile?”

Would it be better than their traditional process?

### 3.1 Research Methods and Data Collection

Given the objectives, a qualitative approach was employed. As already mentioned in section 2.4.3, the research consisted of an analysis of the transcripts of eighteen recorded semi-structured interviews constructed specially for this purpose. The level of software design experience of the interviewees ranged from less than 1 year to greater than 10 years. Their level of knowledge of Agile methods ranged from none to a 1-day course, a 5-day course, up to having worked for 5 to 6 weeks in a team piloting agile. The participants gave their full permission for their interviews to be recorded and were aware of the purpose of the interviews. Each interview lasted between 15 and 30 minutes. A guideline questionnaire using open questions was used to steer the interviews. The questionnaire is available in Appendix A. More detail on the formulation of the questions has already been discussed in 2.4.3. The questions used in the semi-structured interviews were formulated so as to obtain detailed information on the following main issues:

- Is the current software design processes/methodology adhered to?
- What are the issues or problem areas in the current processes?
- Will Agile methods be an improvement on methods currently used? Will it solve some of the problems that exist in the current process?
- Are there aspects of Agile that will not work?
- What do developers think of the various XP and Scrum practices, particularly those that are seen to be challenging developers to work in a different way?
- Will the use of Agile methods make the day-to-day work more or less enjoyable?
Afterwards the interviews were transcribed. As stated earlier in chapter, some principles drawn from Grounded Theory were used in the qualitative data analysis. A full Grounded Theory with theoretical sampling and theoretical saturation was not attempted. Open coding was used to analyse the interview transcripts in detail (Georgieva and Allan 2008; Glaser 1978). Key points from each interview transcript were collated. Then a code – a phrase that summarises the key point in 2 or 3 words was assigned to each key point (Georgieva and Allan 2008). The codes arising out of each interview were constantly compared against the codes from the same interview, and those from other interviews. This is Grounded Theory’s *constant comparison method* (Glaser 1992; Glaser and Strauss 1976), which was used to group the codes to produce a higher level of abstraction called a category. Qualitative Data Analysis Software, Weft QDA, was used to support the coding of the interviews. Weft QDA supports the importing of text documents. Codes can be defined and text that corresponds to a particular code can be marked. Figure 19 contains a screenshot of the *Documents and Categories* in Weft QDA. Weft QDA facilitates queries on particular codes or categories. An example is displayed in Figure 20. Using open coding the major themes, concerns and expectations were extracted from the interviews.

The final step of Grounded theory is generating a theory, also known as theoretical coding. Theoretical coding involves conceptualising how the categories and their properties relate to each other as a hypothesis to be integrated into a theory (Glaser 1978). This theoretical coding was only employed at the last stage of analysis (Glaser 1992), rather than being enforced as a coding paradigm from the beginning as suggested by Strauss and Corbin (1990). Glaser lists some common structures of theories known as *theoretical coding families* (Glaser 1978; Glaser 2005). Because of the nature of the research, the *Six C’s* coding family (Glaser 1978; Glaser 2005; Kan and Parry 2004), (Contexts, Conditions, Causes, Consequences, Contingencies, and Covariances) was used. The Six C’s model was applied to problems that had been identified by the developers in the existing software development process. The model was used to analyse the causes and consequences of these problems and present the theory that the developers expected that Agile practices would improve the situation as regards these problems.
Table 7 contains the codes and categories that emerged from the data. The table also shows the number of interview transcripts that contained a particular code.
<table>
<thead>
<tr>
<th>Categories</th>
<th>Codes</th>
<th># Mentions</th>
<th># Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
<td>Existing process is document heavy</td>
<td>4</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Existing process has documents that are of no value</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Agile will reduce the level of documentation</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Documents do no always reflect the code</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>No input by design to IP</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>No standard way of writing the DIS</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DIS done after implementation</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DIS of no value</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Introducing Faults</td>
<td>Missing impacts in the DIS and IP</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Lack of information from external parties cause faults</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Code inspections</td>
<td>Code inspections not always performed</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Code inspections get squashed at the end</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Unit testing</td>
<td>Unit testing not always performed</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>Unit testing left too late</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Difficult to unit test GUI code</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unit tests not properly written or are brittle</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Pair Programming</td>
<td>Good for quality</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Good for competence building and knowledge transfer</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pair programming is tiring</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Personality conflicts will be an issue</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Personally I would not like it</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pair programming is nothing new</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OK for complex coding</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Would be good</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>TDD</td>
<td>TDD requires a change of mindset</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>TDD will be difficult with legacy code</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Will not be enough time to do it</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Need more integration test than unit tests</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TDD has benefits</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Open Plan</td>
<td>Open plan area will be too distracting</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Just need the open area for pair programming</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Management</td>
<td>Managers will need to support agile</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Micromanagement will not work in agile</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>More visibility</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
3.2 Analysis of the Interview Data

A number of major themes or categories were evident in the data. The underlying theme that guided the interviews in which this data was collected was based on the fact that the literature indicated that developers might accept the change of methodology if they felt it would offer improvements over their existing methodology. So of course the design of the questionnaire has a major bearing on the results and the categories that were evident in the data. A description of the major categories and codes is provided below. Some selected quotations drawn from the interviews are included to shed particular light on the codes. The interview number and time of the quotation during the interview is used as a reference. For example [I1-9: 36] means that the quotation is from interview 1 at 9 minutes and 36 seconds into the interview.

3.2.1 Documentation

Existing Process is Document heavy

A number of the participants felt that the pre-Agile process was document heavy, especially in terms of Microsoft Word documents.

“If you look at the template for a document, you are supposed to do it that way, but the thing is it’s a word document. They are dreadful for word documents in here.” [I1-9: 36].
“I suppose the way we worked before we had – it was very document heavy. That was one thing.” [I10-0:45]

**Existing Process has Documents that are of no value**

Some of the documents in the pre-Agile process were written for the sake of following the process and apart from that the documents were of little use or value.

“It would be one of those documents I don’t actually believe provides a lot of useful information” [I1-7: 32]

“There is a lot of documentation that could be eliminated that is only done for the sake of a process”. [I3-4: 27]

“There would be a lot of documentation that I think are pointless because nobody else reads.” [I6-3: 31]

**Agile will reduce the level of documentation**

There was an expectation that developers would produce less documentation when they employed Agile methods.

“Yeah I think anything that reduces documentation is good.”[I3-6: 11]

“In Agile we will have much less emphasis on documentation.”[I5-10: 21]

**Documents do not always reflect the code**

Sometimes the code changed but the corresponding documentation (for example an architecture description or a package description) didn’t get updated accordingly. This rendered the document useless in that case, so there was a tendency to read the code rather than consult an “inaccurate” document.

“If you make a change in your implementation, you won’t go back to the DIS and reset it.” [I1-6: 22]

“You might as well look at the code and to be honest the code is more accurate than the document.” [I5-7: 37]

**No input by design to the Implementation Proposal**

The IP (Implementation Proposal) was a document written in the early design phase outlining on a high level the new feature to be implemented or developed. Developers felt that when they received this document it was “set in stone” and they had no opportunity to contribute towards the system solution and outline of effort or cost.
“By the time the IP was handed to us, you know, decisions had been made – We had no input.” [I1-3: 41]
“You just got the IP, it was handed over to you.” [I5-1: 48]

No standard way of writing the Design Impact Specification

The DIS (Design Impact Specification) was a Design Analysis document used in the pre-Agile process. The level of detail varied depending on the author.
“There was no standard way of writing the DIS. Everybody- whoever got hold of it just did what they needed to do.” [I1-5: 43]

Design Impact Specification done after implementation

The DIS, which contained the Design Analysis information, was often written after the coding for sake of process adherence.
“We are supposed to write a DIS from the IP but that nearly always happens after we have done it (the code) for the sake of the document – that’s what I feel anyway.” [I4-0: 49]
“But it was hard to write a Design Impact Spec. straight away so you would have a bit of analysis, you'd nearly have code done and then you would realise ok this is what should go into the document because you could write your Design Impact Spec and it could be a completely different implementation that goes into it.” [I6-2:24]

Design Impact Specification of no value

Developers felt that the Design Impact Specification (DIS) as an artifact was of no value, resulting from the fact that it was often written after the code was complete or if it was written before the code, there was no guarantee that the code reflected what was described in the document.
“The DIS is a bit of a hindrance you know. You could see that no one really reads it. We were kind of updating it for the sake of it really.” [I6-3: 37]
“So the documents after a few weeks of development they are not required anymore because they don't have enough of details, they can't have enough details if you are in development so long that you have so long shipment dates.” [I9-2: 48]
3.2.2 Introducing Faults

**Missing Impacts in the Design Impact Spec. or Implementation Proposal**

The solution was not fully specified in the IP or sometimes the DIS. These missing impacts would then cause additional unplanned work during design and coding which lead to delays and overruns.

“I suppose the level of detail in the IP - we always had extra stuff or stuff missing in the IP that you’d only see afterwards even though it went through a number of processes or reviews and stuff. Invariably there is always stuff missing from it.”[I11-7: 13]

“We don't seem to get enough detail in from our IPs, so it leads to a lot of faults.”[I14-2: 45]

**Lack of information from external parties**

Sometimes network nodes from other units were not available to test towards or the interface towards them changed during development.

“Well we are designing to nothing - the nodes a lot of the time are not existing you know I mean “ [I17-4: 35]

3.2.3 Code Inspections

**Code Inspections not always performed**

In the pre-Agile process code inspections were not always performed even though there was an ambition to carry out code reviews.

“The idea was that at the end someone would look at your code, but that didn’t always happen, due to the rush to get it out the door.” [I6-2: 45]

“We do - I suppose we have an ambition level to do desk checking and code reviews. I wouldn't say they are always done but we do make an effort - they are planned and I have actually done some myself” [I11-2:01]

**Code Inspections get squashed at the end**

Because the code inspection was one of the last activities before code was delivered to test, it could get omitted due to time constraints. Even if the code inspection was performed there was a reluctance to change code that “worked” in order to improve the characteristics of the code.
“One of the bad things about our old way is someone writes code and you do a code review or a code walkthrough and again what would happen is you would only do a code walkthrough very close to a delivery” [15-13: 07]
“There were code reviews like, at the end I think the idea was that there would be a code review so someone would look at your code but that didn't always happen due to you know the rush to get out the door” [I6-2: 45]

3.2.4 Unit Testing

Unit Testing not always performed
Unit Testing was not always performed on the code before it was integrated or delivered to Feature Test.
“And I skipped basic test because we didn't do any basic test. There was no basic test done, there was no basic test suite created. There was no unit test - we didn't have any time to do it.” [I10-2: 40]
“We haven't really being doing that much. Basic testing or unit testing- Basically we would code it and then put it on a server and test it and see if it works.” [I16-2: 41]

Unit Testing left too late
Because unit testing was the last activity before delivering the code to Feature Test, the developers often ran out of time and the activity was skipped.
“Towards the end, you always find that an overrun in stuff and code you know or what you have to do so, that side of things, the documentation, the basic test, inspections, at the very end and often they would be missed.” [I11-7: 34]
“One thing we are very bad at is unit tests. Like we do write them but sometimes they can be the last thing left behind and we should do more of that” [I18-2: 48].

Difficult to unit test the GUI code
Some of the applications provide GUI (Graphical User Interface) code, which was difficult and time consuming to unit test.
“Any of the GUI test are typically very difficult to write, to get good coverage in the GUI side.” [I11-10: 45]
Unit Tests not properly written or are brittle

Unit tests were written in order to increase the code coverage, but were not really testing the code in a logical way.

“Too often unit tests are written more for code coverage rather than actually verifying anything.” [I1-14: 48]

3.2.5 Pair Programming

Good for Quality

There was an expectation that pair programming would be beneficial in terms of a quality improvement.

“I like the Pair Programming. In the old process, code walkthrough was done very close to delivery and it was hard to get time to do it and you do it very quickly and you might find you disagree with the way something is done, but it is too late to re-do if it works.” [I5-12: 53]

“I do think you write better code because you will stop little faults.” [I6-10: 46]

Good for competence building and knowledge transfer

Pair Programming was expected to assist with the competence building of less experienced developers and seen as a good way to transfer knowledge.

“I see a benefit from a competence, from a training type of scenario, but again it depends on the pairs you put together.” [I1: 25:50]

“Pair programming, we have done a small bit of that and there are benefits in it, especially for competence development.” [I3: 6:58]

“Pair programming is great if you want to bring a person that’s new into a project.” [I11-17: 10]

Pair programming is tiring

Developers that had started to try out pair programming observed that it was more intense and tiring than “solo” programming.

“Pair programming is very tiring- very intense- very hard to see that everyone would spend 6 hours a day on it.” [I6-8: 55]

“It’s very intensive as well. I wouldn’t recommend it every day of the week” [I10-7: 32]
Personality conflicts will be an issue

A concern of the developers was the personality conflicts would be a problem as regards pair programming and that the success or otherwise of pair programming would very much depend on the pairing.

“The pair programming would depend on the people you are working with because there are certain people that are right all the time and no matter what you say they are still right and if you are trying to work with somebody like that in pair programming you might as well not be pair programming.” [I3-8: 00]

“I don’t know with two experienced headstrong - a lot might not - you’d loose time because if the two are fighting over how it should be done as opposed to helping one another they might not be as productive.” [I14-8: 10]

Personally I would not like Pair Programming

Some developers felt that would not like to pair program under any circumstance.

“No I absolutely don’t like this. I absolutely disagree with that.” [I12-5: 48]

“From the course on agile, it was stated every two hours you are meant to switch partners, ...the way it is being implemented you have the same guy with you for a couple of days.... even some one I love, I don’t like sitting beside them the whole day every day.” [I17-9: 44]

Pair Programming is nothing new.

There was an opinion that people always worked in pairs to solve difficult problems, so pair programming was just putting a name on something that was always done.

“I think it’s really good, it’s nothing new, even when the Agile wasn’t there people used to come together or team leads used to help you out or you helped them out. There is nothing new in that.” [I8-7: 44]

“I’m doing a bit of that at the moment anyway-because I’m new to the project, I’d be getting help from people that would be there a few years.” [I16-4: 39]

Pair Programming OK for complex coding.

For a number of developers, pair programming was viewed as an activity that would be beneficial when there was complex coding involved only. In other circumstances it would not add value.

“Definitely if you have something very complex it might be worthwhile.” [I1-26: 10]
“I think pair programming is probably great for the initial design of it where you are coming up with the problems and the issues and you go away and test, but then a lot of it after that is refactoring and refactoring and I don't know do you need two people together to refactor.” [I11-16: 21]

“See it all depends on if you are going to be doing it beside somebody who is going to be doing very simple updates - that would probably be boring” [I18-4: 52]

**Pair Programming will be good.**

A number of developers expected that pair programming would have its benefits.

*I like the Pair Programming. In the old process, code walkthrough was done very close to delivery and it was hard to get time to do it and you do it very quickly and you might find you disagree with the way something is done, but it is too late to re-do if it works..... We find we are more likely to re-factor...you feel you have backup....we have done that and we haven’t introduced faults.* [I5-12: 53]

“I think pair programming could be good- I haven’t done it really but I in the project we were working in, we tried it unofficially to try it out and one of the guys I was programming with was more experienced than me and I learnt a lot from him.” [I7-10: 35]

**3.2.6 Test Driven Development**

**TDD requires a change of mindset**

Developers were concerned that Test Driven Development which involves writing the test case before the code requires a change of mindset when they were familiar with writing the code first and testing afterwards.

“We are struggling with the concept of TDD and getting it working so we are still not doing a good enough job. ... The concept of writing the test before you do any work still quite hasn’t worked yet.” [I1-20: 23]

“So it’s handy in one way but it would take a while to get used to it or to get into the mindset of writing the tests first. Everyone goes straight into the code – it’s just natural” [I6-12: 58]

“I think it’s very hard to do. It’s a different mindset...I started it and I thought it was good, then it went downhill and now its kind of gone out the window.” [I7-9: 19]
**TDD will be difficult with the legacy code**

Because many features being developed were product enhancements, the developers were concerned about how TDD would work, as they did not have test suites available for the legacy code.

“It is not possible to do Test Driven Development because in one of the projects I was extracting existing code, creating the framework, so we didn’t do any Test Driven Development in that. This is applicable just for new features.” [I9-10: 52]

“We won’t have any test cases for the legacy stuff....if you are changing two lines in your old code, you’re supposed to have a full test suite for that....the legacy code will be a problem. Test Driven Development for new code is probably not that bad”. [I11-13: 08]

**Not enough time for TDD**

One concern was the Test Driven approach is time consuming and that it might not work as a consequence.

“I would like to see the Test Driven Approach applied, but it is not there because of the deadlines...We are writing a lot of tests so we need to tweak a little bit more to write the test harness first and then the code, which requires time.” [I8-9: 10]

**Need more integration test than unit tests needed**

Another concern voiced was that integrating and testing a number of classes together was of more benefit that unit testing individual classes.

“If you need to create a server component which provides some functionality then testing the actual components just helps you with testing parts, a few classes. If these components are for example you know, it is not very easy, you would need to do integration test more than unit test and this is type of integration test, they are more complex.” [I9-12: 01]

**TDD has benefits**

Developers had a positive view of TDD with many of them seeing potential benefits in TDD.

“We have automated testing...it is test driven...there are benefits it in it.” [I3-6: 29]

“I quite like TDD, so I have been using that.” [I4-1: 35]
3.2.7 Open Plan Environment

Concern was expressed about the possibility of moving to an open plan work area. While some developers felt that there would be too many distractions and that it would be difficult to concentrate in an open plan area, others felt that a dedicated open area used only for pair programming would be sufficient.

Open Plan would be too distracting

“In general people are just talking at lunch time about the whole agile, the desk layout, the desks have to be laid out in certain areas because is basically open plan for the purpose of pair programming but people are doubtful over it and wondering how it will workout with noise levels, its grand when you are pair programming, but when you want to go to your desk and concentrate” [17-11: 30]

“I don’t see the point….chances are in open plan, lads will still take phone calls and have meetings and another having the stand-up meeting, so if some guy is trying to do something and you have three more over there shouting and roaring about something else” [114-10:22]

Just need open area for pair programming

“I sit in my corner…. but then it’s good for a team to shout across to some one, but maybe if you just had that as a dedicated section where you go for certain periods of the day.” [115-5:33]

3.2.8 Management Support

A concern emerged that managers would need to support the adoption of Agile Methods for it to succeed and that micromanagement would not work in Agile. There was an expectation that Agile methods would provide more visibility for managers and team leaders into the progress of developers.

Managers will need to support agile

“People have said it depends on the support you get from managers as well. If your manager doesn’t like it, it is not going to work” [14-3: 40]
Micromanagement will not work in Agile

“The added advantage we have is that our manager recognises his role and leaves us alone. I can see that the previous process was coming from where there was micromanagement going on. That could never happen in agile. It just won't work.” [110-4: 40]

More Visibility

“Its more visible, the guys they need to, at the meeting they need to say what they are doing, what they are trying to accomplish today and the team leader is there and he keeps track of what the guys should do and you know it is very easy to find out if someone is producing something or not.”[I9-9:06]

3.2.9 Adoption Process

The organisation being observed was a large development organisation and developers naturally had concerns about the agile adoption process itself including how agile could be scaled across the organisation.

Success will depend on the actual implementation

The concern here was related to how the organisation would implement agile methods and what practices would be adopted. The fear was that the organisation would just pick some of the practices.

“They (the organisation) weren't fully getting onboard with it. They were only taking the bits they wanted from it, so if they do that I am not sure if some of it can work.” [14-6: 07]

Difficulty in scaling up agile

It was not obvious to the developers how agile could be scaled up across the organisation and as a result they were concerned about it and whether it was actually possible.

“In reality I can't actually see the company getting there they way things are right now.” [11-25:06]

“Certain parts here where they wouldn't suit. Some of the products here design might be in wippro and systems are here so there is no way that systems and design and test are all going to sit together in the one little team and develop a story.[I14-5:23]

“I am not really 100% sure of how or are we "bastardising" what agile is meant to be to.” [I17-7:56]
A big change to move to agile in one project

The concern expressed here although it was only mentioned once in this context was that a lot of new practices in terms of XP and SCRUM were proposed to be introduced in one project, which would be a lot for developers to deal with, as well as implement the new features.

“I think it's a new process. Its something they are pushing big time for R7 and it is sort of going for R7 and going from where we are in R6, doing IP and the way we are working now - to go and change all that in R7 is a big change for - in one project lifecycle I suppose”[I11-8: 40]

3.9.10 Should be trying out new processes

A positive theme that emerged from the interviews was that the developers agreed that the organisation should be trying out new process such as agile and were open to new ways of working.

“I wouldn't be overly skeptical about it but it sounds good on paper, but you have to try something and it is probably worth doing.” [I14-12: 16]

“It seems like a good idea, it seems like there is going to be a quicker turnaround from what I have seen in the course anyhow. You introduce things in very small parts and quicker turnaround.” [I16-4: 18]

“I think we are open to it here you know from what I can tell. I mean some of the people that I used to work with are now actually working on agile at the moment and they are very enthused about it.” [I17-7: 26]

3.2.11 More pressure to deliver

A concern that emerged in three interviews was that there would be more pressure to deliver because of the frequent deliveries in Agile.

“Over the month if your cycle is over the month you could say ok you'd find the time to make up but now in two weeks time you don't have the time to find so I actually think it makes it - it going to make it. So I think there is more pressure from that point of view.”[I1-27: 12]

“With the very short iterations, probably one week two weeks so if you miss something today you can compensate tomorrow you can work the extra hours, but it is only 2 days you have to do this so sometimes I can see people are not comfortable to come to the meeting.”[I2-19: 45]
3.2.12 Difficult to work remotely

A small opinion or concern was with regard to home working or working remotely. Because the proposed use of a whiteboard and post-its was a low technology mechanism it could not be visible from a remote location.

“I don’t particularly like it because if I have to work from home for instance I can’t see the board so I rather have a software solution.” [11-28: 53]

“Some people start at 7 in the morning, some people start at 9.30. I prefer to work - that is when my brain is working better - in the evening time.” [I6-10: 30]

3.2.13 Standup Meetings

While the general view was that the stand-up meetings would be good, the following comments indicate that the meetings held to date were getting side tracked with technical discussions. Some teams had started to use Standup meetings.

To me they haven’t run all that well. They get a little bit side tracked on technical discussions where it should be just what am I doing today. [11-28: 25]

Sometimes the stand-up meeting becomes a discussion of some of the problems that we have. [15-11: 30]

The stand-up meetings everyday, they are good because if you have any issues they are brought up at a meeting everyday or any questions because often if you have a question, in the old days you just ask one person and now you can throw the question out there and you can get three or four different answers. [I6-7: 08]

3.2.14 Will not be isolated from “other work”

The concern here was that developers would not be isolated from work that was not directly related to the user story they would be working on. They would have to answer e-mails, phone calls, and attend reviews thus slowing down progress on their agile work. Their concern was that it might not be obvious to management where the time was being spent.

You get pulled left right and centre in here. [I1-27: 22]

Some guy is fixing something and it is breaking there and one bug is coming and it is because of this work and it is because of that work and we have new development while fixing the old [I2-10:05]

Biggest problem that we have is getting pulled away from your agile work. So let’s say we try to do six hours a day, we’re not getting close to six hours. You have meetings to go to, you
get e-mails about something or a fault comes in or you have a document to review or there’s new work coming in, can you analyse it and estimate the hours. [I5-16: 25]

3.2.15 Others

Other expectations that emerged, but only featured in one interview were that the shorter iterations and CI (Continuous Integration) would be an improvement.

“I do like the two week iterations, it keeps you much more focused.”[I1-18: 05]

Also it was hoped that there would be better prioritisation of requirements in projects.

“So what I am hoping is people will come and say right here is my top 3 requirements today.”[I1-23: 30]

A concern was that in the existing process some people like to have control and this could be a problem in an Agile Way of Working

“But if they have, like the previous projects I have been on they have special, certain people who were kind of like to keep control. I don't see that happening in the agile way, so they have to be re-educated.”[I10-11: 13]

And some other views that were evident once in the interviews were that

- The lead-time was too long in the existing process,
- That heavy TR (Trouble Report) fixes came late in projects
- In the existing process code was always released even if you were not happy with the quality.

3.3 Generating Theories

The final step described in Grounded Theory is concerned with generating a theory or theoretical coding (Glaser 1978) and although full Grounded Theory with theoretical saturation was not attempted in this study, theoretical coding was used. Theoretical coding means that relationships between the categories are classified and integrated into a theory (Glaser 1978). Glaser recommends that theoretical coding be only employed at the later stages of analysis (Glaser 1992), while Strauss and Corbin (1990) promotes its use from the beginning. We followed Glaser’s recommendation in relation to when theoretical coding was carried out. Glaser lists a number of theoretical coding families. One such family that was applicable to our research is the Six C’s (Context, Condition, Causes, Consequences, Contingencies, Covariance) coding family (Glaser 1978; Glaser 2005; Kan and Parry 2004). The examples Glaser uses for the theoretical codes are formal concepts from sociology which
make basic claims about the ordering of the (social) world like the terms causes, contexts, consequences and conditions. The model begins with some particular category that has emerged from the data, for example “inefficiencies in the traditional process”. The Six C’s theoretical model describes (1) Contexts (which is the context of the study – e.g. development teams in a large organisation), (2) Conditions: factors that are prerequisites for the category to manifest, (3) Causes: reasons for the category, (4) Consequences: results or outcomes, (5) Contingencies: moderating factors between the causes and the consequences and (6) Covariances: correlations between different categories. Certain events are defined as causes and others as consequences or effects. The codes are then integrated to a causal model. The Six C’s model was applied to problems that had been identified by the developers in the existing software development process. Using the Six C’s coding model (Context, Condition, Causes, Consequences, Contingencies, Covariance), the following three theories are presented.

- Ineffective Use of Developers Time
- Too many Basic Errors found during Feature Test
- Missing Impacts discovered during Feature Test

The three items listed were identified as problems in the traditional development process being used in the case company. The theory is that the problems can be solved with the introduction of agile methods. This is illustrated below for each theory using the 6 C’s model. In the overall aim of the study, these theories are considered significant since they highlight where the developers feel agile methods can offer benefits. This is important since the research links acceptance of change with perceived benefits of the change (Baddoo and Hall 2002; Baddoo and Hall 2003).

### 3.3.1 Context

For all of the three theories, the context was that 18 developers were interviewed. The level of software design experience of the interviewees ranged from less than 1 year to greater than 10 years. Their level of knowledge of Agile methods ranged from none to a 1 day course, a 5-day course, up to having worked for 5 to 6 of weeks in a team piloting agile.

The Condition, Causes, Consequences, Covariance and Contingencies are will now be discussed for each theory.
3.3.2 Ineffective Use of Developers Time

**Condition**

Developers were spending time writing documentation that has little or no benefit, which is an ineffective use of their time.

**Causes**

As shown in Table 7, we can see that four interview participants felt that the existing process “was very document heavy”. Six participants said, “There is a lot of documentation that could be eliminated - that is only done for the sake of a process”. Two participants mentioned that “the code is more accurate than the document”, which means that the value of the document is questionable since it may not reflect the code. A document called the Design Analysis Specification (DIS) was used in the existing process to document the Design before starting coding. Four participants stated "that nearly always happens after we have done it (the code) for the sake of the document’’ and three participants said, “The DIS is a bit of a hindrance. You could see that no one really reads it. We were kind of updating it for the sake of it really.”

**Contingencies – Agile Methods**

The interview participants felt that Agile Methods would reduce the level of documentation. For example there would be more face-to-face communication and
whiteboards or wiki pages used as a means of documentation. “We won't have to do a DIS and waste a day or whatever it is doing the document and then everyone sits around for 2 hours reviewing a document.” and “In terms of documentation we document it in the code and maybe we spend more effort doing a proper design”. 3 participants mentioned specifically that they anticipated agile would reduce the amount of documentation.

Covariance

Covariance occurs when one category changes with the changes in another category (Glaser 1978; Kan and Parry 2004). It would only be possible to comment on covariance after Agile Methods were practised for a time. And since only one organisation was being observed, covariance would be unlikely to be experienced.

3.3.3 Too Many Basic Errors Found During Feature Test

Too many errors were found during Feature Test that should have been discovered during Unit testing or Code Inspection phases. It is much more costly to correct the errors during Feature Test that if they were found during the Coding/Unit Test phase.

Causes

As shown in Table 7, we can see that ten interview participants commented that code inspections were not always performed even though they were planned. “We are supposed to
list our code reviews but in reality they are not really happening” [I4]. Another problem mentioned by five participants was that the code review was scheduled very close to code delivery, so changes would only be suggested if there were faults in the code. There would not be time to take on board inspection comments that were related to the structure and readability of the code. “One of the bad things about our old way is someone writes code and you do a code review or a code walkthrough and again what would happen is you would only do a code walkthrough very close to a delivery. “Or what you might find is Jesus I totally disagree with what you have done there - you should have done it this way but I guess you can't do that now because we don't have time to redo it and you say 'oh right go ahead so it is not the way I would do it' “ [I5]

With respect to the unit testing, because along with the code reviews, the unit testing was the last activity to be performed before code delivery, it was often skipped due to time pressure.

“Code reviews minimal. You might do one or two at the start when you had time and then as more and more stuff was pushed in top of you and deadlines were moved they would be thrown out. They were always the first to go. Along with the unit tests would be second to go and then we would go straight into coding, do a happy test before you release and that's it then.” [I10]

**Contingencies – Pair Programming and TDD**

Developers felt that if the practice of pair programming was adopted, then this would mean that two developers were working on a piece of code and would replace the role of the code inspection. Also structural changes to the code could be taken on since the process was an ongoing one from the beginning of coding. “So at least when you are doing pair programming the guy sitting beside you, you both might may the same mistake but at least you have some hope of disagreeing - I don't think you should do it that way or whatever.” [I5]

With Test Driven Development, the test case is written the code, so in theory all code should be unit tested. “TDD I think will put the emphasis back on getting test harnesses up and in place early and that's good” [I1]

**Covariance**

Covariance occurs when one category changes with the changes in another category (Glaser 1978; Kan and Parry 2004). It would only be possible to comment on covariance when Agile
Methods were being practised for a time. And since only one organisation was being observed, covariance would be unlikely to be experienced.

### 3.3.4 Missing Impacts discovered during Feature Test

**Context**
Software developers in large organisation

**Condition**
Missing impacts discovered in Feature Test

**Causes**
- Hard handover of IP
- No input to IP from design
- No DIS standard
- Lack of info. From external parties

**Consequences**
- Loss of productivity
- Negative impact on developer morale

**Covariance**
Different approaches to early design analysis

**Contingencies: Agile Methods**
- Cross functional teams
- Face-to-face communication

Fig. 23. Six C’s Missing Impacts

**Condition**
Too many errors are found during Feature Test as a result of missed impacts. These errors typically involve a lot of re-work to correct them.

**Causes**
As shown in Table 7, we can see that five interview participants spoke about impacts being missed in the IP (Implementation Proposal). The Implementation Proposal documented the early design analysis. The Systems Group performed this activity. The IP was then handed over to design to complete the Design Analysis and the coding. A lack interaction or communication between Systems and Design groups meant that the developers were not always clear on what was intended in the IP.

“It is up to the designer themselves to along with the IP author to come along and say ok what is meant with this. A lot of it is, most of the time it is probably clear, but some of the time there is - it is open to interpretation.” [I11] Sometimes items that were missed only became apparent during the Design Analysis or coding. “I suppose the level of detail in the IP - we always had extra stuff or stuff missing in the IP that you'd only see afterwards even
though it went through a number of processes or reviews and stuff. Invariably there is always stuff missing from it “[111]. Developers felt that there was not enough detail in the IP. “We don't seem to get enough detail in from our IPs, so it leads to a lot of faults and I don't think we spend enough time doing our actual design or our design impact spec. We all seem to miss - we are great for the straight forward happy case and then we miss the side - we have improved a bit - but we had been missing all the side - the scenarios we hadn't thought about – buts, maybe scenarios and we end up then with a lot of unusual cases where things will fail and fail badly.”[14]

“There is always something missed. And I think the IPs probably aren't detailed enough and its only afterwards, even though it goes through the review process - its only afterwards that you start seeing that there’s new rules that are required that aren't mentioned in the IP which are more hours, that aren't considered.” [111] Developers felt that they did not have enough input into the IPs and that Systems were not available to follow up on queries or issues that arose during the design. “By the time the IP was handed to us, you know, decisions or whatever had been made. We had no input.” [11] “And once the systems gave us the IP they weren't there.” [18]

**Contingencies**

Using Agile Methods, it was proposed to have a greater co-operation between the Systems and the Design. A Systems Engineer would explain the feature as it was described in the IP and would then be available during the design phase to clarify issues and questions. This would work better than trying to document everything in an IP and do a hard handover because it would not be possible to describe everything in a document. The “IP, implementation proposal which is saying how, how we should solve the problem, how we should implement the solution, but these documents they can never contain all the details because it is not possible actually to cover, to prepare a document which describes the actual solution, it is just not possible.” [19]

“There only as you start working on the stuff, the fallout from the work that you are doing, that you see that there is more work required. So that is one of the main drawbacks now with the way that we are working in the IP based System.” [111]

**Covariance**

Covariance occurs when one category changes with the changes in another category (Glaser 1978; Kan and Parry 2004). It would only be possible to comment on covariance when Agile
Methods were being practised for a time. And since only one organisation was being observed, covariance would be unlikely to be experienced.

### 3.4 Summary and Related Work

This paragraph summarises the main concerns and expectations of the developers that emerged in this case study and relates them to other research and studies that have been performed.

**Table 8: Summary of major concerns of developers**

<table>
<thead>
<tr>
<th>Concerns</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Personality conflicts will be an issue with pair programming.</td>
<td></td>
</tr>
<tr>
<td>TDD requires a change of mindset that will take time.</td>
<td></td>
</tr>
<tr>
<td>TDD will be difficult for legacy code and for GUIs.</td>
<td></td>
</tr>
<tr>
<td>Too much distraction in an open plan area</td>
<td></td>
</tr>
<tr>
<td>Pair programming will be intense and tiring</td>
<td></td>
</tr>
<tr>
<td>Time management – getting “pulled off” your agile task and interruptions from phone, e-mails and other meetings.</td>
<td></td>
</tr>
<tr>
<td>More pressure to deliver due to the short iteration.</td>
<td></td>
</tr>
<tr>
<td>Will there be sufficient support from management?</td>
<td></td>
</tr>
</tbody>
</table>

**Table 9: Summary of major expectations of developers**

<table>
<thead>
<tr>
<th>Expectations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile will reduce the level of documentation as a result of the face-to-face communication.</td>
<td></td>
</tr>
<tr>
<td>Pair programming will help the situation where code inspections are being skipped.</td>
<td></td>
</tr>
<tr>
<td>Pair programming will be good especially for complex work and also for competence building and knowledge transfer.</td>
<td></td>
</tr>
<tr>
<td>TDD has lots of benefits in terms of testing and quality of the code</td>
<td></td>
</tr>
</tbody>
</table>
Table 8 and Table 9 summarise the main concerns and expectations of the developers with respect to adopting Agile methods. We now discuss each of these concerns and expectations and their relationship to other studies.

### 3.4.1 Documentation

Developers expected that the introduction of agile would result in less “non-code” documentation. Their pre-agile or traditional process was document heavy and less documentation would be welcomed. They felt that with agile, more face-to-face communication and less “hard handovers” would result in the need for less documents.

This was a logical expectation on behalf of the developers. The documentation in agile processes is reduced to informal documentation, face-to-face communication, and on-site customer visits. Using informal documentation reduces the cost associated with maintaining it (Coram and Bohner 2005). In contrast the pre-agile process used in the organisation based on Royce’s waterfall model was document heavy:

“How much documentation? My own view is ‘quite a lot;’ certainly more than most programmers, analysts, or program designers are willing to do if left to their own devices. The first rule of managing software development is ruthless enforcement of documentation requirements. Management of software is simply impossible without a very high degree of documentation” (Royce 1987).

The waterfall model has been in use since 1970. Studies have shown that the waterfall model results in high effort and costs for writing and approving documents for each development phase. It requires the approval of many documents, changes are costly to implement, iterations take a lot of effort and rework, and problems are usually pushed to later phases (Sommerville 2004; McBreen 2002). In a study conducted by Petersen et al. (2009) into bottlenecks and problems with the waterfall model in a large company, they found that in the Requirement phase and in the Verification phase unnecessary documentation was produced. He also says that, “the concept of quality gates requires producing a lot of documentation and checklists which have to be fulfilled before passing on the requirements to the next phase.”

### 3.4.2 Unit Testing and Code Inspections

Developers felt that in the old process, time ran out for unit testing and code inspections and they expected with agile pair programming and test driven development should improve the situation.
With the waterfall method, a problem is that finished tasks are left for later phases to solve (Sommerville 2004). Peterson et al. (2009) state that “the number of faults and quality issues is influenced negatively when using waterfall development. The main cause for this is late testing after everything has been implemented”.

3.4.3 Pair Programming

Developers expected that there would be a benefit in pair programming mainly from a knowledge transfer point of view, but they were very concerned regarding personality conflicts and that pair programming will be tiring.

Begel and Nagappan (2007) studied a team that was using Agile Methods and they experienced some concern “that some developers are not cut out for Agile”. Pair Programming was slow to be adopted (Hodgetts 2004) because it was “too intrusive on each developer’s ability to concentrate on the tasks at hand”. The results from a field-study conducted in 15 software companies in Greece using eXtreme Programming by Sfetsos et. al., (2004) show that one of the success factors was pair-programming. However, difficulties in distribution of programmers and bad relations among staff had a negative effect on pair programming.

3.4.4 Test Driven Development

A number of developers had completed a training course on test driven development and they expected that it would improve quality, but they were concerned about the lack of test harnesses for the legacy code, the difficulty in using TDD for Graphical User Interface code and they were aware the change in mindset needed for TDD.

The fact that not only TDD, but Agile itself requires a change of mindset, is highlighted by Moore and Spens (2008) stating that “Agile is a mindset, not a skill set” and in Begel and Nagappan (2007), one reason given for not practicing Agile was too much legacy code.

Hodgetts (2004) makes the case for creating “an appropriate environment for the adoption of agile practices” and found that several iterations were required before Test Driven Development began to show results.

3.4.5 Frequent Deliveries

Developers were concerned that frequent deliveries associated with agile methods would result in a more pressurised environment. Frequent delivery of working code is one of the key
principles in agile. This does not seem to have been cited in the literature as an issue for developers. This may be due to the fact that most studies are conducted after the developers were using agile methods and were able to see the benefits of the frequent deliveries.

3.4.6 Open Plan and Co-location

The subject of having an open plan team area provoked a very negative response. Developers were concerned that it would be too distracting to work in such an area.

In the “State of Agile Survey 2010” conducted by VersionOne, 43% of the respondents indicated that they use open work areas. The literature advocates the use of open work areas, which is one of the primary practices in XP. In his article “TeaRoom”, Martin Fowler (2010) acknowledges that some people are concerned about noise levels and would prefer private offices. His article gives some ideas about how to provide a good open plan work area.

3.4.7 Management support, Time Management & Adoption

Developers were concerned about the support that the agile adoption would get from the management and they were also concerned regarding the management of their own time in so far as they might be 100% allocated their agile work. They had concerns about how agile would scale in a large organisation. Developers that had participated in stand-up meetings found the meetings were taking too much time.

The difficulty of time management surfaces in work from Moore and Spens (2008) and interesting that it is with big projects. “In large-scale projects, team members must participate in cross-team activities and allocate time to project-wide activities” (Moore and Spens 2008). The investigation into Agile practices at Microsoft by Begel and Nagappan (2007) shows that respondents to a survey “complained about the inefficiency of these meetings [stand-ups], especially when they were poorly run. In the “State of Agile Survey 2010” conducted by VersionOne, “Management opposed to change” was cited as one of the main concerns regarding Agile Adoption. (VersionOne 2010)

Begel and Nagappan (2007) also found concerns in Microsoft regarding management buy-in. “Many program managers were worried that upper-level management would ask for progress reports and productivity metrics that would be hard to gather in an Agile work environment. Management ignorance of Agile methodologies was also a worry.”
3.5 Conclusion

Baker and Thomas (2007) point out that the adoption of agile will not happen overnight and advocate an incremental approach and the benefit of staying in touch with an agile leader is raised. It is clear from the interviews with developers that a degree of mentoring is vital is the early stages of agile adoption. Weyrauch (2006) emphasises the importance of communication in any transition and puts in place a communication framework to solve initial problems. Sureshchandra and Shrinivasavadhani (2008) advocate the use of Agile mentors and acknowledge that test driven development and pair programming are difficult to implement. Most of the concerns and expectations raised by the developers were seen in previous research. Two issues, the concern over frequent deliveries and the concern over the open plan area do not surface in existing material that I am aware of. The issue with the open plan area is likely to have been instigated by the fact that the company setup a “pilot” open plan area that was not well designed. The issue with the frequent deliveries may not have been seen previously because most studies into agile are performed well into the adoption period.

Having concluded this first step of the research, my own key takeaway points are as follows:

- Large organisations are used to following well-defined software processes. Generally the process is organisation wide. This would be the case in particular for a CMMI compliant organisation. When adopting agile methods organisations need to define an agile process tailored to their needs. This has been shown to be the case by Lindvall et al. (2004) in a study of projects at ABB, DaimlerChrysler, Motorola and Nokia. I also believe that a tailoring of the practices for the individual teams within the organisation is necessary. One major discerning factor is whether or not the team is working with legacy code or not. If a team has a large amount of legacy code and there is no test suite or an incomplete test suite existing, then their approach must be quite different to the team that is beginning a “greenfield” application. Changing from a process-centric to a people-centric approach is challenging, especially in a company where a higher level of CMMI has been achieved. In this study the organisation was setting “agile ambition” levels for the teams. That is what practices should be carried out, pair programming? Test Driven Development? Ideally the teams themselves should have the task of setting this “ambition level”. This could be an indication that there was reluctance on behalf of the organisation to “empower” the teams.
• Given that especially in large organisations in fact most of the projects will involve legacy code, there must be a clear strategy communicated to the developers on how they are expected to deal with legacy code. A number of books have been published in this area such as Feathers (2004) and Martin (2009) that deal with re-factoring of legacy code. However it will only be possible for developers to conduct a very limited amount of re-factoring within the timeframe of a project. Unless the organisation is prepared to investigate time and resources into a re-factoring activity and preparation of test harnesses it will not happen. This means that one of the vital underpinnings of agile, that is, “test early and often” will be compromised. This can have a detrimental effect on quality. The situation is similar for testing of Graphical User Interfaces. It is not realistic to expect 100% automated testing of GUI code due to the difficulties with test tools.

• A lot of communication between the agile adoption leaders and developers is needed at the beginning of the adoption process. In this study the developers had a lot of concerns – some of those could have been easily allayed by communication. In this study, it is clear that the developers did not know what would be expected of them with respect to practices such as pair programming. With regard to pair programming, the subject of “personality conflicts” surfaced many times. This gives the impression that developers felt they would be “forced” to pair program with somebody they didn’t want to pair program with.

• With regard to open plan areas and co-location, the organisation in this case study created a pilot open area. Having seen the open area, in my own opinion it was not an enticing work environment. Keeping in mind budgetary constraints, I think at the initial stage of agile adoption more resources and effort should have gone towards the creation of a more conducive work area. If this wasn’t possible then the pilot area should not have been set up. This pilot area provoked a very negative reaction from the developers.

• Having an agile coach or mentor available to the team is important especially in the initial stages. Even a relatively straightforward practice like a stand up meeting needs some guidance to keep them on track.

• Practices such as pair programming and Test Driven Development are perhaps the most challenging. I think if developers were aware that they could introduce these practices in a gradual way, some of their concerns would be put to rest.
The most interesting outcome in my opinion is that the developers talked predominately about the agile practices such as test-driven development, pair programming, stand up meetings etc. While the practices themselves are a means to an end, the core of agile is values (communication, simplicity, feedback, courage and respect).
4 Validating the View of the Developers
The objective of the research documented in the previous chapter was to capture the main themes from the interviews with developers in relation to their concerns and expectations when moving from a traditional Waterfall based methodology to an Agile one. This part of the research uses a survey to authenticate the themes that emerged in Chapter 3. Twenty-seven developers completed a survey. The questions in the survey were devised based on the major themes from the analysis documented already in Chapter 3. The objective of this research was to validate the results in terms of concerns and expectations in the previous chapter. This survey was carried out four weeks after the completion of the semi-structured interviews.

4.1 Research Methods and Data Collection

To endorse the concerns and expectations of the developers that emerged from the semi-structured interviews, a questionnaire using closed questions was devised and 27 developers completed it. Eighteen developers took part in the semi-structured interviews and thirteen out of the eighteen also completed the questionnaire. The questionnaire was distributed via e-mail. The case company nominated the participants based on our request that

- There should be about 30 participants
- At least 50% of the participants in the semi-structured interviews should be surveyed
- The participants should be aware that they would soon adopt agile methods

The fact that the case company nominated the individuals was not seen as a risk to the validity of the outcome. As with the semi-structured interviews, the case company would not have any reason to portray a particular bias. Participants needed clearance of their line management to complete the questionnaire due to confidentiality policy. According to Oates (2007), a sample size of thirty is a good rule of thumb for small-scale research projects. Statistical analysis, such as, calculating average and mean is not reliable for sample of less than 30. If findings are to be generalised to the whole population, the sample must be of an adequate size to ensure accuracy. Approximately 200 systems engineers, developers and testers worked in the area that were initially adopting agile methods, so to apply the findings to the whole population with a 95% confidence level, 130 participants would be required (Creative Research Systems n. d.). The objective of this part of the research was to substantiate the findings from the semi-structured interviews, rather than apply findings to the
whole population. The data is therefore treated as qualitative data with an emphasis on interpretation rather than statistics.

The questionnaire had two questions that were related to factual data (experience levels). The rest of the questions were opinions where the Likert Scale was used in most cases to indicate a degree of agreement or disagreement. The complete questionnaire is available in Appendix B.

4.2 Questionnaire Data

The data below was compiled from questionnaires completed by 27 developers with varying levels of experience in software design and varying levels of exposure to Agile Methods. The data in the tables consists of the actual numbers of participants that answered in a particular way. The figure in brackets is the actual number of the subset of participants (total of 13) that took part in the semi-structured interviews that answered in a particular way.

Table 10: Number of years in software development

<table>
<thead>
<tr>
<th>Years</th>
<th>&lt; 1 year</th>
<th>1-3 years</th>
<th>3-5 years</th>
<th>&gt; 5 years</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 (0)</td>
<td>2 (2)</td>
<td>4 (4)</td>
<td>20 (7)</td>
</tr>
</tbody>
</table>

Length of time working in software development

Table 11: Experience Level of Agile methods

<table>
<thead>
<tr>
<th>What I have read myself</th>
<th>Attended Presentations</th>
<th>Attended Training</th>
<th>Starting with pilot projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 (2)</td>
<td>2 (3)</td>
<td>5 (3)</td>
<td>13 (5)</td>
</tr>
</tbody>
</table>

Experience of Agile Methods

Table 12: Document Driven Process

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 (4)</td>
<td>14 (8)</td>
<td>2 (0)</td>
<td>2 (1)</td>
<td>0 (0)</td>
</tr>
</tbody>
</table>

"Pre-Agile we are using a document driven process."
Table 13: Documents are important for maintenance

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 (2)</td>
<td>15 (6)</td>
<td>3 (3)</td>
<td>2 (2)</td>
<td>0 (0)</td>
</tr>
</tbody>
</table>

“Documentation is important from the point of view of maintaining the system and transfer of knowledge.”

Table 14: We write unnecessary documents

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 (5)</td>
<td>7 (3)</td>
<td>4 (3)</td>
<td>5 (2)</td>
<td>1 (0)</td>
</tr>
</tbody>
</table>

“In the non-agile process we spend a lot of time doing documents just for the sake of the documents. Nobody uses them afterwards.”

Table 15: Design Impact Spec. is not a valuable document

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 (3)</td>
<td>6 (5)</td>
<td>6 (3)</td>
<td>9 (2)</td>
<td>1 (0)</td>
</tr>
</tbody>
</table>

“The DIS (Design Impact Specification) is not a valuable document”

Table 16: Easier to write the code

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 (4)</td>
<td>8 (5)</td>
<td>5 (2)</td>
<td>8 (2)</td>
<td>1 (0)</td>
</tr>
</tbody>
</table>

“It would be easier to write the code itself than to prepare a document such as the Design Impact Specification that fully describes all the technical details of the System impact.”

Table 17: Unit Testing & Code Inspections Compromised

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 (5)</td>
<td>10 (6)</td>
<td>2 (2)</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

“In the non-agile world, unit testing and code inspections are the last activity planned before the code is released to Feature Test and as a result these activities (code review and unit testing) are often compromised or skipped.”
Table 18: Agile is common sense

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 (2)</td>
<td>12 (6)</td>
<td>8 (4)</td>
<td>2 (1)</td>
<td>2 (0)</td>
</tr>
</tbody>
</table>

“A lot of the agile practices (Test Driven Development, Pair Programming, Stand-up meetings, Co-located and Cross functional teams, User Stories etc.) are just common sense. Calling it Agile is just putting a label on it.”

Table 19: Agile is just a buzzword

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (0)</td>
<td>5 (4)</td>
<td>13 (6)</td>
<td>7 (2)</td>
<td>2 (1)</td>
</tr>
</tbody>
</table>

“Agile is just a buzzword, promoted by training and consultancy companies in order to sell training courses and consultancy.”

Table 20: We should continually improve our processes

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>18 (8)</td>
<td>9 (5)</td>
<td>0 (0)</td>
<td>0 (0)</td>
<td>0 (0)</td>
</tr>
</tbody>
</table>

“As a software design organisation, we need to try out new processes and continually improve our way of working.”

Table 21: Improve Existing Process

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1)</td>
<td>5 (2)</td>
<td>5 (3)</td>
<td>9 (5)</td>
<td>7 (2)</td>
</tr>
</tbody>
</table>

“We should concentrate on following the existing process correctly and properly checking the existing quality control points instead of changing to Agile.”

Table 22: Too many new concepts in Agile

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>5 (4)</td>
<td>8 (3)</td>
<td>12 (5)</td>
<td>2 (1)</td>
</tr>
</tbody>
</table>

“There are too many new concepts in Agile to introduce together in a single project”
Table 23: More Pressurised Working Environment

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>11 (6)</td>
<td>10 (3)</td>
<td>6 (4)</td>
<td>0 (0)</td>
</tr>
</tbody>
</table>

"The introduction of Agile Methods will result in a more pressurised working environment."

Table 24: Not suitable for home working

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (1)</td>
<td>9 (5)</td>
<td>13 (6)</td>
<td>2 (1)</td>
<td>1 (0)</td>
</tr>
</tbody>
</table>

"It will be more difficult to work remotely (home working) in an agile environment."

Table 25: Pair Programming for knowledge transfer

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 (4)</td>
<td>15 (9)</td>
<td>4 (0)</td>
<td>3 (0)</td>
<td>0 (0)</td>
</tr>
</tbody>
</table>

"With regard to pair programming, it will be beneficial for knowledge transfer and competence building.

Table 26: Pair Programming is nothing new

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (1)</td>
<td>9 (5)</td>
<td>10 (3)</td>
<td>6 (4)</td>
<td>0 (0)</td>
</tr>
</tbody>
</table>

"Pair programming is nothing new, we always worked in pairs or groups to solve problems or assist less experienced designers"

Table 27: Personality conflicts in Pair Programming

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 (3)</td>
<td>13 (6)</td>
<td>4 (2)</td>
<td>6 (2)</td>
<td>1 (0)</td>
</tr>
</tbody>
</table>

"With regard to pair programming, conflict of personalities will be an issue"
Table 28: Pair Programming Benefits

<table>
<thead>
<tr>
<th>I think pair programming is beneficial</th>
<th>I think pair programming has benefits but it depends on how the pairing is done</th>
<th>I don’t see any benefits in pair programming</th>
<th>There could be benefits but personally I would not like that way of working</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 (3)</td>
<td>15 (8)</td>
<td>1 (1)</td>
<td>6 (1)</td>
</tr>
</tbody>
</table>

“With regard to Pair Programming.

Table 29: Agile solve our process issues

<table>
<thead>
<tr>
<th>solve a lot of the problems in our existing process</th>
<th>solve some of the problems in our existing process</th>
<th>not make much of a difference</th>
<th>introduce new issues and problems we have not seen yet</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (0)</td>
<td>24 (12)</td>
<td>0 (0)</td>
<td>4 (1)</td>
</tr>
</tbody>
</table>

“Agile will.”

Table 30: Teams need to be co-located

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neither Agree or Disagree</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 (4)</td>
<td>11 (5)</td>
<td>4 (2)</td>
<td>4 (2)</td>
<td>1 (0)</td>
</tr>
</tbody>
</table>

“It is important that teams are co-located in close proximity to improve communications”

Table 31: View on Open Plan

<table>
<thead>
<tr>
<th>something I would not mind</th>
<th>good for pair programming and working in groups, but too much of a distraction at times</th>
<th>A work environment I wouldn’t like at all</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (0)</td>
<td>13 (8)</td>
<td>10 (5)</td>
</tr>
</tbody>
</table>

“For me a very open plan environment similar to that used in the Agile pilot would be:”

4.3 Analysis of Questionnaire Data

The data from the questionnaire broadly concurs with the qualitative analysis as described in Chapter 3. We can also see that the figures (in brackets) related to the subset participants that previously took part in the semi-structured interviews are aligned with the overall survey participant group. The overall group contained 27 participants and the subset contained 13 participants so for the ratio’s to concur, the figure in brackets should be roughly half of the figured relative to the total survey figure.
4.3.1 Background and Experience of the Developers

Overall the developers that completed the questionnaires were quite experienced. 20 had greater than five years experience with only 3 having less than three years, as shown in Table 10. Table 11 shows that almost half the developers surveyed had started to trial Agile Methods in pilot projects and a further 5 had received some form of training in Agile Methods. As with the interview participants, the case company nominated the survey participants. They tended to select experienced participants, probably because the more experienced developers would have stronger opinions and more to contribute. Because three quarters of the developers were in the > 5 years category, no analysis based on experienced levels was attempted.

4.3.2 Documentation

Table 12 shows that the developers felt that their traditional process was indeed document driven with 23 out of 27 either Agreeing or Strongly Agreeing that the process is document driven. However there is an acknowledgement from the developers that documentation is important for maintenance activities with 22 Agreeng or Strongly Agreeing that “Documentation is important from the point of view of maintaining the System.”, as shown in Table 13. This figure is higher than what you would expect based on the interviews. In the interviews there was an opinion that the documentation might be out of line with the code and that it was probably better to read the code than a document that might not be aligned with the code. “You might as well look at the code and to be honest the code is more accurate than the document.” [I5]

The view that “a lot of documents are done for the sake of the process” is reflected in Table 14.

In the theme based analysis, there was strong evidence that the DIS (Design Impact Specification) was not a good way to document the Design Analysis and that a lot of time was spent unnecessarily updating the DIS to reflect the implementation. However, as shown in Table 15, only 11 participants Agreed or Strongly Agreed with the statement “The Design Impact Specification is not a valuable document.” and 10 Disagreeing or Strongly Disagreeing. Based on the interviews, it could be expected that the Agree/Strongly Agree figure would be higher. The reason may be that the developers feel that documenting the Design Analysis is necessary, but then a lot of time is spent updating the DIS after reviews
and “keeping it in line with the code.” This is supported by the evidence in Table 16 where we see that the opinion on whether it is easier to go straight to code is somewhat mixed. This is one case where the subset of participants involved in the interviews have a different slant to the overall with a far higher number of them agreeing or strongly agreeing that it is easier to go straight to code instead of writing the DIS.

Cohn and Ford (2003) described their experiences of introducing Scrum in seven organisations. They say that they “invariably found programmers who enjoy producing non-code artifacts far more than they are willing to admit.” It is interesting here that the result of the questionnaires regarding documentation is a little different to what emerged in the interviews. It could be suggested that developers place more value on the non-code documentation that they cared to admit in the face-to-face interviews, but this doesn’t seem to be the case if we only consider the subset of respondents that also took part in the face-to-face interviews. My own opinion is that it is more to do with the phraseology of the questions in the questionnaire. I think that developers do feel that good accurate documentation is valuable for maintenance purposes, but in the questionnaire they were not thinking that the documents might be out of date and not totally in line with the code. I also believe that developers see a value in the Design Impact Specification, but the questionnaire does not capture anything about the amount of time that is spent updating the Design Impact Specification after reviews. Developers would not say that they thought the Design Analysis as an activity was not valuable, but some of the formalities associated with the reviewing of the document were unnecessary. In hindsight the questionnaire should have had questions such as “Documentation is not always aligned with the code.” “Documentation will not be used if there is a risk that it is inaccurate or not aligned with what has been implemented in the code.” and “Too much time is spent updating the DIS after reviews and after the code has finished.”

In summary the developers feel that documentation is an important for knowledge transfer and system maintenance, but that too much time is spent on unnecessary documentation.

4.3.3 Unit Testing and Code Inspections

The results shown in Table 17 is strongly aligned with the theme based analysis with 89% of developers Agreeing or Strongly Agreeing that unit testing and code reviews are often compromised. This is a straightforward finding.
4.3.4 Moving to Agile Methods

Again the data here concurs with the thematic analysis. Table 18 shows that 15 out of the 27 of developers Agree or Strongly Agree that many of the agile practices are common sense that just so happens to fall under the Agile umbrella. They do not believe that Agile is just a “buzzword” as shown from the data in Table 19 and from Table 19 there is unanimous agreement (all Agree or Strongly agree) that new process and methods should be tried out. Only 6 participants Agree or Strongly Agree that the organisation should concentrate on improving its existing processes instead of adopting Agile, as shown in Table 21. This shows that the majority of the developers surveyed (21 out of 27) believe that adopting Agile Methods is the correct thing to do. From Table 22, we see that they felt it was possible to take onboard the Agile practices within the timeframe of one project. And from Table 29 we can see that in general developers felt that agile will solve some problems but are realistic in that they do not view it as a “silver bullet” that will solve all problems.

4.3.5 Pair Programming

While the developers see benefits to pair programming, the issue of how the pairing is done comes across strongly as a concern. 20 respondents Agree or Strongly that pair programming is beneficial for knowledge build up and competence transfer, Table 25. 11 Agree or Strongly Agree that pair programming has always happened to a certain extent, Table 26. But only 7 out of 27 developers that completed the questionnaire feel that conflict of personalities will not be an issue (Disagree or Strongly Disagree) when it comes to pair programming, Table 27. The majority feel that while pair programming has benefits, it very much depends on how the pairing is done, Table 28.

4.3.6 Physical Working Environment

Co-location of teams is viewed as important, but having an area that is too open is not popular. 18 respondents Agree of Strongly Agree that team members should be located in close proximity to each other, Table 30, but 13 believe that the open area should be available for pair programming only and 10 feel that they would not like that environment at all, Table 31.
4.3.7 More pressurised work environment

More pressurised work atmosphere due to amount of deadlines emerged as a theme in the interviews. From the questionnaire data, we see that 11 out of 27 developers Agree that the environment will become more pressurised, Table 23, while only 6 Disagree.

4.4 Discussion and Related Work

Research by Lindvall et al. (2004) indicates that many companies – in particular large companies – are approaching agile methods with a high degree of scepticism due to the conflicting data regarding “in what environments and under what conditions agile methods work”. They showed that it is possible for large organisations to adopt agile methods but they point out the challenge for large organisations is “efficiently integrating the agile project into its environment. To fully benefit from agile practices, organizations must better define the interfaces between the agile team and its environment, thus avoiding the double work caused by the conflict between agile practices and traditional ones.”

Cohn and Ford (2003) state that “Most developers respond to the proposed introduction of an agile process with the appropriate combination of skepticism, enthusiasm, and cautious optimism. Other developers, however, either resist the change or overzealously jump into the project without enough forethought. Either reaction can cause problems.”

The research outlined in Chapter 3 and the data from the questionnaires presented here can be used to answer the question “What do the developers really think about adopting agile? Are they skeptical, optimistic or something else? What do they expect from agile? What are their concerns about agile?”

While Cohn and Ford (2003) state “how an agile process is introduced into an organization will significantly impact the ultimate success of the process change.” Some developers will “be excited to try it” and for others that oppose change “this newness is an obstacle” (Cohn and Ford 2003). After the qualitative analysis and quantitative analysis, we would have say that we partly agree with Cohn and Ford (2003). The developers displayed a level of “enthusiasm and cautious optimism”, but there is little evidence of “skepticism”. The developers were however realistic so with respect to this particular case study, it could be said, “the developers are approaching agile with the appropriate combination or realism, enthusiasm and cautious optimism”.
4.4.1 Enthusiasm and Optimism

The developers acknowledged that the traditional process was not perfect and can be improved. Primarily they felt that they were spending too much time on documents, particularly documents that do not add value. Unit testing and code reviews and inspection activities were compromised due to time constraints. In the traditional waterfall based process, unit testing and code reviews were undertaken after coding. Since it was then the last activity before delivery of the code for testing, the activity was skipped or shortened due to time pressure. 17 of the 27 developers *Agree* or *Strongly Agree* that unnecessary documentation was being produced and 24 of the developers surveyed *Agree* or *Strongly Agree* that unit testing and code reviews was compromised in the existing process.

All of the developers *Strongly Agree* or *Agree* that they should continually improve their ways of working and 23 out of 27 feel that Agile Methods will address at least some if not all of the deficiencies in their existing process. Only 6 of developers would prefer to keep and improve the existing methodology instead of adopting Agile Methods and only 5 of the respondents feel that there are too many new concepts in Agile Methods to introduce them all at once.

In summary, developers want to improve their existing work methodology and feel that moving to Agile Methods is the correct thing to do. Linking back to the literature on change theory and software process improvement initiatives, research by Baddoo and Hall (2003) indicated that the primary reason people are willing to change software processes is evidence of the benefits of the change. This was found to be very much the situation in this case study. The developers felt that Agile methods would offer them benefits going forward and overall they were willing to change to that way of working.

4.4.2 Realism

While developers are enthusiastic about reducing unnecessary documentation, we can see that they exhibit realism by acknowledging that documentation is necessary for maintaining a system.

As discussed in Chapter 3, developers see value in Test Driven Development, but they are aware that there will be difficulties with Test Driven Development for legacy code that doesn’t have an existing test suite and that it will not be easy to switch over to Test Driven Development because it requires a mindset change.

When it comes to Pair Programming, 20 *Agree* or *Strongly Agree* that it will be beneficial especially for knowledge transfer and competence building. The realism is displayed when
only 7 Disagree or Strongly Disagree that personality conflict will be an issue when it comes to pair programming.

The developers were aware that in a large organisation, an interruption to their “agile work” is likely. This has been discussed in Chapter 3. We have seen that developers thought that more frequent deliveries in an agile way of working could result in a more pressurised work environment. Only 6 out of 27 would Disagree or Strongly Disagree that agile will add more pressure.

When it comes to the physical work environment, 18 agreed or strongly agreed that teams need to be co-located and 13 felt that open plan would be good for pair programming but were concerned that it would be too much of a distraction at times.

4.4.3 Skepticism

We have to disagree with Cohn and Ford (2003) with respect to skepticism, because very little evidence of skepticism had been found in the research so far. Our semi-structured interviews and our quantitative questions have been formulated to determine the developer’s views on moving to an Agile Methodology. However there has been very little evidence that the developers were skeptical about Agile. This is supported by the fact that only 5 out of 27 Strongly Agree or Agree that agile is just a buzzword promoted to sell courses and consultancy. 2 respondents believed that it would solve of lot of the problems in the traditional process and 21 believed that it would solve some of the problems in their existing process. So 23 out of 27 of the developers felt that agile methods would offer an improvement over the traditional methodology. Looking back at the literature again Reichers et al. (1997) concluded that cynicism about changes has detrimental consequences for such employee outcomes as commitment, satisfaction, and motivation. While cynicism and skepticism is not exactly the same thing, cynicism could be described to be a subset of skepticism of which there is little evidence.

4.5 Summary

Therefore we can conclude this part of the analysis by saying with respect to this case study that “the developers are approaching agile adoption with an appropriate combination of realism, cautious optimism and enthusiasm.”
5. *Agile in Action: The Developers View*
The objective of the research documented in Chapter 3 and Chapter 4 was to build up a picture of all of the initial concerns and expectations of the developers with respect to moving from a traditional plan based methodology to agile methods. Now as the developers gained experience and practiced the new methodology we analysed the following:

- Which of their initial concerns were valid?
- Which of their concerns materialised to a lesser or greater extent and why?
- Which concerns didn’t materialise at all?
- Were their expectations met to a lesser or greater extent?
- Which expectations turned out to be false and why?
- Did new concerns materialise during the adoption phase?
- Did agile produce results or effects that were not anticipated initially?

### 5.1 Research Methods and Data Collection

This phase of the data collection (which is referred to as Data Collection Stage 3 in Chapter 2) took place ten months after the initial data collection (which has been detailed and analysed in Chapter 3 and Chapter 4). This time we had seven one-to-one audio-recorded interviews with developers. We also conducted group interviews with nine feature teams, the results of which are documented in the next chapter. The one-to-one interviews lasted for 20 to 40 minutes and the team interviews lasted for between 1 and 1.5 hours each. All the developers involved in the one-to-one interviews were based in Ireland and six of them had previously taken part in the Data Stage Collection D1 i.e. the first set of semi-structured interviews. To ensure comparability of data, the one-to-one interviews are analysed separately in this chapter. The data from the team-based interviews is contained in the next chapter. Again the interviews were semi-structured and steered by a guideline questionnaire. The seven developers were drawn from four different teams. This time the questionnaire used questions to focus on the main themes from the initial interviews while also including some open questions allowing for new phenomena significant to the teams and participants to emerge. Some questions were included to try and determine the “level of agility” or to what degree the agile practices were being adopted. The guideline questionnaire that was used is available in Appendix C. Again the interviews were transcribed and Weft QDA tool was used in the coding of the interviews.
5.2 Main issues raised in the first stage interviews

Issues that were raised by developers during their initial interviews either as concerns and expectations are re-visited and there is a discussion of whether the concern or expectation materialised or if the opinions of the developers had changed in some way. Selected quotations are drawn from the interviews to give a better perspective. For each quotation, the interview number and the time in the interview at which the comment was made. So [I29-10:15] interview 29 and the comment was made 10 minutes and 15 seconds into the interview. Out of the seven developers interviewed, one of them (Interview No. I30) was in a team and it emerged that the project and its deliveries had not gone well for the particular team. The exact cause for the failure to meet the project goals and deadlines was difficult to pinpoint even for the team themselves, but to some extent they seemed to be overwhelmed by the introduction of new technology, a high level of functionality and migration to a new methodology all in one step. This is important to point out for the analysis.

5.2.1 Documentation

Initially developers believed that the introduction of agile methods would result in less “non-code” documentation. Their existing process was document heavy and less documentation would be welcomed. They felt that with agile methods, more face-to-face communication and less “hard handovers” would result in the need for less documents.

The evidence was that there was a slight reduction in the total amount of documentation but not by the amount they had initially anticipated.

“We still have 80% of the documentation that we had its just that it is managed better so it doesn’t feel like a big burden of documentation.” [I28-00: 54]

“I guess there is about the same in terms of administrative but in terms of design documentation there is a bit less..” [129-02:00]

“Its cut back a bit but not to the extent that I though it would.”[I32-00:32]

In the pre-Agile process, developers strongly indicated in the interviews that they didn’t see a value in some of the Design Analysis documents because the solution they would implement in code would invariably differ from what they specified during Design Analysis. However in the Agile process their view changed and they found the Design Analysis much more useful. Probably this was because it was done closer in time to the coding and focussed on smaller work items and also was documented in a less formal way.
A Design Impact statement and we would have used to start off our implementation and the expectation was that we wouldn’t necessarily produce that document anymore but we discovered that we weren’t then necessarily doing enough analysis [I30-01:14]

The conclusion was that documents became “more meaningful” when following an agile methodology.

“I find value in the DIS and the reason it is valuable now is …” [I28-01:38]

In the traditional process, “quality doors” were used as a “go/no go” indicator at the end of phases. Most of these “quality doors” and the checklist documentation associated with them were retained in the implementation of the agile process, which meant some extra administrative documents.

“I have looked at the process and there is still a very meticulous sign off on a lot of things which I don’t think is not really necessary.” [I33:01:39]

5.2.2 Physical Work Environment

In the initial interviews the subject of having an open plan team area provoked a very negative response. An open area set up initially as part of an agile pilot project was not found to be favourable with the developers. Many of the developers initially felt that an open plan environment would be distracting. Out of the seven developers interviewed, only one of them (Interview No. I30) was in a team working in an open plan area. Having seen the open plan area my own opinion was that it was not aesthetically pleasing or well laid out. The developer that worker in that area had some issues with the level of noise and distractions.

“The layout was really bad. A lot of people said they found it incredibly distracting getting your work done because there was just conversations going on all the time. Whereas if you had just gone down there to do your pair programming and you went away and you had your own space somewhere else it probably worked out a lot better” [I30-24:56]

But developers that didn’t have an open area were starting to see a need for co-location at least and spontaneous communication..

“but we are all going to sit together and that is I kind of really believe in that too because if you are sitting at something and something comes into you head, you might drop over as opposed to saying I must bring that up at the next meeting “[I31-23:01]

So I do believe that sitting together is going to make a big benefit and not just design and feature test and if you had systems all the better but that's not going to happen in our cases anyway. Hopefully we will get design and test together. [I31-23:26]
5.2.3 More Pressurised Environment

There was an initial reaction that the frequent deliveries associated with agile methods would result in a more pressurised environment.

This concern did not really materialise. Because Continuous Integration eliminated the big bang style delivery and integration that had existed previously, it actually alleviated pressure. “The continuous integration is a kind of tool that helps us monitor and kind of see things that doesn't put pressure on.” [I33-08:55]

However the continuous integration was not up to speed in all teams.

“So the lack of having an integration server, an integration system up and running, unless somebody was manually running PMD we weren't going to see that visibility.” [I30-30:34]

5.2.4 Pair Programming

With regard to Pair Programming, in the first stage interviews, developers expressed the view that there was a benefit in pair programming mainly from a knowledge transfer point of view, but there was a much concern regarding personality conflicts. Only one of the four teams from which the seven interview participants were drawn had actually pair programmed. For some teams, it hadn’t been in their ambition level, but found that it was happening to a certain degree regardless. Some had tried it and found that it was taking too much time to pair program everything, so they tended to pair for the more complex pieces of functionality.

“We didn't do so much of the pair programming” [I29:30:36]

“I think they have certain ambition levels in the process don't they and when we started we were kind of saying it is not really part of the ambition we have now. In reality we do actually sit down especially with somebody junior in the team members for quite a while.” [I33:02:23]

Some had tried it and found that it was taking too much time to pair program everything, so they tended to pair for the more complex pieces of functionality.

“But sometimes it is valuable when you are working on a new design or something like that where strictly coding by pair programming is quite useful whereby you are in effect reviewing what you are doing all the time” [I34-03:46]

“But we found that it was expensive to do it and that the benefit in the quality was probably higher but in terms of the cost to get something out was too high if you like. We wouldn't get it done. So pair programming I'd say 50-50, maybe even 40%.” [I31-09:18]
Personality conflicts were not something that arose or prevented pair programming

“People that don't like it from the start. That person had left since. Apart from that person I don't know anybody that has refused point blank to pair program” [I28:09:48]

“You could have a conflict between the two soothe person I'm working with at the moment is a bit senior to me but we have had no problems” [I32:13:07]

5.2.5 Test Driven Development

During the first interviews, a number of developers had completed a training course on test driven development and they felt that it would improve quality, but they were concerned about the lack of test harnesses for the legacy code and they were aware the change in mindset needed for TDD.

At the second interview stage the thinking in this area hadn’t really changed. The lack of test harnesses for the legacy code was an issue and developers were having some difficulty with changing their way of working to write the test case first.

“We try and do test driven development but quite often it is hard to get into the mode of thinking. I write a test before I write code - how can I write it when I don't even know what is it going to be? “[I31:10:08]

“But it is not always the case because of the way the legacy code is a fault is there its difficult to test and it's a tradeoff sometimes.” [I34:08:25]

5.2.6 Improvement in Code Inspections and Unit Test

Developers felt that in the old traditional process that time ran out for unit testing and code inspections and that by using agile methods pair programming and test driven development should improve the situation.

There was variation between the feature teams with respect to the amount of test driven development performed, which has been outlined above in 5.2.5. There was also a difference in the extent that pair programming was implemented in the various teams. Since pair programming and TDD wasn’t always carried out this expectation was compromised but there was an increased focus on quality and testing.

“Our basic test work has vastly improved in the last year and it is being treated as part of the functionality that has to be delivered so the mindset there has changed completely that we have to plan and have to deliver both your code and your testing code “[I28-02:52]
“We try to comply with it as much as we can. We have a bit of an issue with GUI code. Its our application, classically really we have a GUI layer and although there is a little business logic in there if we are making a change now we do try to write tests for that.” [I33-03:49]

“At the moment we mix and match. As I said with the junior programmers, I literally said today ok we did that together, we can consider that reviewed. Other parts where people have just done on their own we still do the code review”. [I33-03:12]

“I think we in our costing for this project, we costed it quite well that we have given ourselves enough time for code reviews and stuff like.” [I32-04:42]

5.3 New Emerging Themes

The second round of interviews contained some open questions that allowed developers to give their views on

- All of the XP practices not already covered
- The best thing about working in an agile way
- Negative issues related to agile
- Preferences to continue to work in the agile way.

5.3.1 Stand Up Meetings

Stand Up Meetings seemed to be the practice that was most widely adopted. While there were initially difficulties keeping the meetings on track and time limited, by now they were working well.

“We had an issue where we had 11 or 12 people in a standup and it would get side tracked. Now we have split it down into two separate teams and two stand ups and because of that the standups are more focused so” [I28-10:49]

5.3.2 Co-ordination between Systems, Design and Test

Teams did not have any access to the “real customer”, so a Systems Engineer generally took on the role of Product Owner. Their role was to maintain a prioritised backlog, participate in Sprint Planning and participate in a product demo at the end of each sprint. Systems were meant to add User Stories to the IP. A feature tester would sit with or as part of the team to carry the testing. The interface between the developers and the testers appeared to be working well but the interwork with Systems was taking longer to get established. The Product Owner role was slow to be adopted and demos were infrequent.
“The common goal is the better product. Before it would be my work is to design your work is to test, now it is our joint effort is to build a better product. With Systems that is not as good.” [I28-12:53]

“No we wouldn't really have demos, even though we often planned we must do this, we must do this.” [I31-13:15]

5.3.3 The Agile Adoption Strategy.

The first part of this questioning was to determine if the team felt that the organisation was correct in adopting an Agile methodology. The second part is related to the adoption strategy and ambition levels.

The organisation rolled out what was an aggressive agile adoption strategy. Many XP and Scrum practices were introduced together in a single project. Developers did seem to feel a bit overwhelmed and that it took time to get off the ground with the practices.

“It probably was too much to start but I think you don't just change everything at once. You need to go through it, change see what works, what doesn't work and some people like pair programming some people don't.” [I28-05:37]

However there was also a view expressed that an aggressive adoption strategy results in a build up momentum which carries the changes forward.

“The good thing about the big bang approach there is a lot of focus on it and there is a lot of drive behind.” [I31-6:48]

5.3.4 Continuing with Agile.

Generally the feeling was to continue to work with the agile practices, but an acknowledgement that it was “no silver bullet”. The methodology was becoming well adopted but it was a slow process.

“There is a lot of principles that are very good.” [I29-39:33]

“It seems to be fairly well adopted, its been adopted in here and its kind of all systems go using agile at the moment you know.” [I32-17:03]

“Most people are quite positive.”[I33-13:12]

5.3.5 Time or Quality Improvements.

Developers were asked to give an intuitive opinion as to whether there were making either time or quality improvements when using agile methods. This was a “gut feeling” without
having done any analysis on number of bug reports or man-hours used. They felt that code structure was improving and this would lead to a quality improvement but that would take some time to manifest itself. No improvements with regard to time was experienced some activities such as pair programming were skipped due to lack of time. As previously discussed, the opinion was that these activities were compromised to due to time constraints and it would take time before the benefits of agile were seen.

“Definitely a huge quality improvement, I can't give you measurement of figures but we'll find issues within our team as opposed to other groups finding them. That's the biggest improvement we'll have. Now people say that's just bringing the test phase earlier into the project and it is but it is a team ownership” [I28-11:15]

“Modularisation both at component and at unit level that lead to easier to impact and better code down the line.” [I29-22:00]

“We weren't really working properly I would have thought. So things weren't running smoothly though I would have thought that in six months ..“ [I31-12:15]

5.3.6 Positive Aspects of Agile.

With regards to agile being an improved way of working, overall from the developers point of view they jury was out to an extent. They felt it would take a further 6 months before measurable improvements would be evident. But they did mention some of the collaborative aspects of agile such as the cooperation between developers and testers. Fast feedback was also mentioned and improved focus on testing.

“Call it improvements, better communication, better teamwork, better integration with test, they are probably the best things we have achieved with the "Agile" word describing it”. [I28-04:20]

“I did think that feature test participation and rather than a you and me approach and having them involved was useful”[I29-19:01]

“I guess the other thing is there is a lot more focus on maybe the testing side.”[I31-3:33]

“This fast feedback and not just unit test cases, integration test cases, you know and then that's gives us confidence so that the overnight build is not going to show up anything major or worse still something in three weeks time” [I31-21:15]

5.3.7 Negative Aspects of Agile.

Developers acknowledged that they were not “fully agile” yet and they did not express negative points regarding agile methods themselves, but mentioned things that should be
improved to achieve a higher level of agility, such as the product architecture, more test driven development or more test automation. The boundary with the non-agile part of the organisation surfaced. Even the developer involved in a problematic project didn’t attribute the failure of the project to the introduction of agile methods while at the same time acknowledging that agile should not be thought of as a “miracle cure”.

“You can't make one small piece of the thing agile and expect that that is going to be a miracle cure for anything……I would hate to think that people would blame agile as the reason why common explorer was had big problems.” [I30-28:07]

### 5.4 Comparing with initial concerns and expectations

Table 32: Materialisation of initial concerns

<table>
<thead>
<tr>
<th>Before Adoption</th>
<th>10 months into the adoption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personality conflicts will be an issue with pair programming</td>
<td>Pair Programming was not implemented widely at his stage. This was more to do with ambition levels and time constraints rather than people not wanting to do it.</td>
</tr>
<tr>
<td>TDD requires a change of mindset that will take time</td>
<td>Level of TDD varied and some were struggling with the change of mindset</td>
</tr>
<tr>
<td>TDD will be difficult for legacy code and for GUls</td>
<td>Increasing the amount of TDD with GUI and legacy code was proving problematic</td>
</tr>
<tr>
<td>Too much distraction in an open plan area</td>
<td>Developers acknowledge the need for co-location but views tainted by badly laid out area.</td>
</tr>
<tr>
<td>Pair programming will be intense and tiring</td>
<td>This was found to be the case in one instance where pair programming was used.</td>
</tr>
<tr>
<td>Time management – getting “pulled off” your agile task and interruptions from phone, e-mails and other meetings.</td>
<td>Did not arise as an issue. A positive aspect from this was that developers felt there was more visibility on their work..</td>
</tr>
<tr>
<td>More pressure to deliver due to the short iteration</td>
<td>Where (near) Continuous Integration was in place it eased pressure.</td>
</tr>
<tr>
<td>Will there be sufficient support from management</td>
<td>Not arising as an issue. The boundary to the non-agile part of the organisation and the fact that agile was employed in only a part of the lifecycle was mentioned.</td>
</tr>
</tbody>
</table>
Table 33: Materialisation of initial expectations

<table>
<thead>
<tr>
<th>Before Adoption</th>
<th>10 months into adoption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile will reduce the level of documentation as a result of the face-to-face communication</td>
<td>No major change in the level of documentation, but the documents meant more</td>
</tr>
<tr>
<td>Pair programming will help the situation where code inspections are being skipped</td>
<td>This was found to be true in situations where it was used</td>
</tr>
<tr>
<td>Pair programming will be good especially for complex work and also for competence building and knowledge transfer</td>
<td>This was found to be true. Pair Programming not widely used.</td>
</tr>
<tr>
<td>TDD has lots of benefits in terms of testing and quality of the code</td>
<td>This was found to be true for the teams that had implemented a lot of TDD and there was increased focus on testing.</td>
</tr>
</tbody>
</table>

5.5. Discussion

Table 32 and 33 contains a summary of how the main concerns and expectations of the developers changed over time. Chapter 6 contains the results of the group interviews with the teams. At the end of chapter 6 we compare the findings from the one-to-one interviews with the group interviews and discuss the overall results, including new and emerging themes in more detail there. First of all, we provide a summary and discussion of the main points from the one-to-one interviews.

A number of papers, case studies and experience reports produced on large-scale agile adoption show that the adoption process takes time. “To move to disciplined execution of Agile practices you can start straight with new practices: an overnight introduction.” However, making the transition permanent “requires a serious, long term change process with good guidance. Management has to guide and support the change” (Goos and Mellisse 2008) and “Moving from waterfall to agile cannot be done overnight or in a single step. It takes time for people to unlearn old traditional practices and move towards agile” (Sureshchandra and Shrinivasavadhani 2008). This was shown to be the case for the organisation under study here. Progress was made on practices such as Test Driven Development, Pair Programming, and Continuous Integration but the transition did not happen over night. With pressure to deliver, it was not possible to have an upfront investment of time in those practices. The over-arching view of the developers was that they had made progress, it was a better way of working, there were benefits in using agile methods, but they had not yet encompassed the practices to the extent needed for them to reap those benefits.
The practices were being adopted slowly and that would continue. The co-operation between developers and testers was working well, but the co-operation between Systems was slower to become established and not working so well. There was more focus on quality and testing and a “gut feeling” that code structure was better. However it would take some time before any quality or time improvements would manifest themselves in real terms.
6. *Agile in Action: The Team View*
The objective of the research documented in Chapter 3 and Chapter 4 was to build up a picture of all of the initial concerns and expectations of the developers with respect to moving from a traditional plan based methodology to agile methods. Chapter 5 documented the concerns and expectations of the developers a number of months into the adoption process based on one-to-one interviews with developers. This chapter documents the concerns and expectations of the developers a number of months into the adoption process based on group interviews that were held per team. Again as the developers gained experience and practiced the new methodology we analysed the following:

- Which of their initial concerns were valid?
- Which of their concerns materialised to a lesser or greater extent and why?
- Which concerns didn’t materialise at all?
- Were their expectations met to a lesser or greater extent?
- Which expectations turned out to be false and why?
- Did new concerns materialise during the adoption phase?
- Did agile produce results or effects that were not anticipated initially?

### 6.1 Research Methods and Data Collection

This phase of the data collection (which is referred to as Data Collection Stage 3 in Chapter 2) took place ten months after the initial data collection (which has been detailed and analysed in Chapter 3 and Chapter 4). We conducted group interviews with nine feature teams, where a systems engineer, between one and three developers and a tester represented the team. Four of the teams were based in Ireland and three of the teams had representatives that had previously taken part in the initial interviews (D1) and the one-to-one interviews. Five of the teams were located in Sweden and they would not have been involved at all in the initial interviews or survey. This time the questionnaire used questions to focus on the main themes from the initial interviews while also including some open questions allowing for new phenomena significant to the teams and participants to emerge. Some questions were included to try and determine the “level of agility” or to what degree the agile practices were being adopted. The guideline questionnaire that was used is available in Appendix C. Again the interviews were transcribed and Weft QDA tool was used in the coding of the interviews.
6.2 Team Based Analysis

The analysis here is documented per team. Along with some background information on the team, the main issues from the first interviews (documentation, physical work environment, more pressurised environment, pair programming, TDD, code inspection/unit test improvements) are re-visited along with some new and emerging themes (User stories, Standup meetings, Continuous Integration, Automated Acceptance Testing, Product Owner Role and Demo’s, Retrospectives, Adoption Strategy, Continuing with agile, Time or Quality Improvements, biggest advantage/disadvantage of agile). Selected quotations are drawn from the interviews to give a better perspective. For each quotation, the participant is specified, e.g. P1 means participant no. 1, and the interview number and the time in the interview at which the comment was made. So [P1][I19-57:15] means participant no. 1, interview 19 and the comment was made 57 minutes and 15 seconds into the interview.

6.2.1 Team 1

Team 1 was based in Ireland. One of the participants had been interviewed in the individual interviews. The team consisted of eight developers and three of them would have been classed as senior. The team had been established about one year and there was a mix of people that had 3+ years experience with the application and 2 with less than 1 year. It was largely a legacy application of medium complexity with updates required. Overall out of all the teams based in Ireland, this was probably the one that progressed furthest with agile methods.

In terms of **documentation**, no real change experienced in the amount of documents but now the documents meant more.

“We have as much as before.” [P1] [I19-57: 15] “You still go through your regular delivery and you have to have all these checklists done and all this you know reports – what’s happening with Agile is that the documents mean more or else they are just dropped.”[P1] [I19-57: 43]

The team didn’t have an **open plan area** with Systems, Design and Test co-located. They now felt that this had been a hindrance in terms of informal communication and acknowledge the importance of the informal communication.

“With the Agile way of working it is all informal so it happens in teams that a couple of designers get together to have a chat and because a tester isn't next to them they loose out on that communication.” [P2] [I19-1: 40]
This team's experience was that the environment was now **less pressurised** as a result of Agile. They were more certain that their delivery would work because it had been completed and tested in shorter increments. They felt that they were more in control of the deliveries.

“I would say less pressurised because we have more control, we have control of the design team to what we can deliver - before we would have a big bang where we were told here is your delivery date and you work up to that date and you drop something in the VOB [Version Object Base] whereas now the plan is every two weeks we have a delivery ready so when the official delivery date comes in it's not a big issue, we just give them the last package we had ready.” [P1] [I19-4: 02]

The team felt that on average they were doing **pair programming** about 40% of the time. They only used pair programming for analysing complex problems more than actually coding.

“Its casual pair programming, as I said before I am more concerned with pair analysis than pair programming.” [P1][I19-31: 46]

“Some classes can be split out so its better have people pair program than working individually but it doesn't always be the case.” [P1][I19-32:16]

“I would say 40%. We try and get the senior designers pair program with designers for their benefit rather than the senior designers.” [P1][I19-32: 30]

With regard to **TDD**, the team was trying to write test cases for all new and impacted code, but the tests were not written before the code and not even before the code was delivered to test.

“We try and get a package ready for test and allow them to start and then we will re-visit the code and write the tests then.”[P1][I19-33: 30]

They were struggling with the mindset change needed to write the test first.

“The mindset is that we actually write Junit tests - that's the first big mindset change for us and the second mindset change would be to write the tests first - that's a bigger hurdle to get over.”[P1][I19-34: 05]

The team planned **code inspections** for code that had not been pair programmed.

“In the team I'm working in if its pair programmed there is no code reviews. If its not pair programmed then there is desk checks and code reviews, somebody else will review the other persons work.” [P1][I19-34: 28]

**Code reviews and unit testing** were still getting left too late, but the situation had slightly improved.
“Slightly improved but we are still - our code reviews and desk checks will fall to the end which is kind of pointless because at some stages it is too late when you deliver the package and you don’t want to be changing code for a desk check or code review - maybe it improving it but.” [P1][I19-36: 19]

Apart from one or two IPs written internally, the team did not get User Stories in the IPs. The team (Systems, Design and Test) met to specify the User Stories from the IP.

“Generally we get the IP and the starting point is we take a look at it and we see what stories we have and I’d say that’s the way it will continue.” [P1] [I19-26:02]

“No, we are not getting an IP with a wonderful set of stories” [P1] [I19-26:40]

The team thought that it would be better if they defined the User Stories at 1/3 IP Stage but of course that will only ever be possible for IPs written internally.

“The sooner the better, so it would make a huge difference at 1/3 stage because any design issues could be pulled out then instead of having this IP ready and we need to deliver this in two or three weeks time” [P1] [I19-28: 28]

Daily Stand up meetings were held. Initially there was one meeting with between 12 and 14 people and the meetings were not running smoothly. It had since been split into 2 sub-teams, one for the main development work and one for consolidation work, so the meetings were more focussed and working better.

“So then decision was taken to split the one team into two separate teams and each team has their standup - they are shorter, more focused and I think they are better, much better.” [P2][I19-30: 55]

The Continuous Integration solution used by the team had two loops. When code was checked in, it got built and unit tests were run. This took about 20 minutes. If successful then at night all acceptance tests were ran and at that time the nightly running of all acceptance tests was taking 20 hours to complete.

Their application had a Command Line Interface (CLI) and a Graphical User Interface (GUI). Test cases for the CLI are automated, but the majority of the GUI test cases were not.

“Most of what we implement would have a GUI side and an import side its nearly 50-50 so they same functionality would be in the GUI and import so its usually the same classes pretty much, so what we plan from a test point of view is that when a user story is delivered that day we would have automated test cases to run that night on that user story - to test that user story”.[P2][I19-50: 20]
The Systems representative acted as Product Owner and the Product Manager maintained a product backlog, but the developers were not so aware of the backlog. Systems and Product Management prioritised the backlog.

“Yeah what we did before we had a wiki page and we kind of re-ordered things that were just literally a text list of improvements and things that are in the backlog and we can put them up and down and TRs as they come in depending on the priority they can go to the top of the list.” [P3][I19: 1:06:50]

The Product Owner sat in on the Sprint Planning. An informal demo would take place involving a tester and the Product Owner (Systems person).

“Actually we found a problem in the demo which was interesting but it wasn't a code problem, it was something else but it proved to be a valuable use of time.” [P3][I19: 1:12:10]

Operational Product Management was not taking part in the demo. The developers expressed a reluctance to have additional personnel getting involved at this stage since their input had been through the Product Owner up to this point.

“I personally wouldn't like somebody coming in after a 2 week sprint at the end to demo something and say oh I don't like that.....You can't bring new..., you have to have them in at the start in the team, you can't just bring resources in that the end” [P1][I19:1:13:24]

Retrospectives were held at the end of each sprint and as a result improvements are taken on board.

“We hold our retrospectives and we look at what we did well and what we can change and we change if we see something that doesn't work we change it and we say ok we need to try it this way and if it doesn't work we'll kick it out and if it works then we'll improve on it, so we are continuously improving. I think even in the last 6 months we have got more efficient.” [P1][I19-1: 29:40]

The team felt that it was correct to adopt Agile methods.

“My own opinion is yes - our agile way of working today is better than the old way of working.” [P2][I19-0: 12:34]

They felt that their transition to Agile Way of Working was gradual enough because the developers started using some XP practices before the agile feature teams were put in place.

“The designers took a lot of the practices in and then test came in, in the last 8-9 months where we had a pure feature team - that has only happened from Summer of last year.” [P2][I19-0: 18:07]

The team wanted to continue working with the agile methodology.
“The team likes working that way it makes much more sense, with Agile practices it's a collection of practices really. I don't see anyone voting to go back to the way things were.” [P3][I19-1: 33:16]

The team felt that quality had improved and that the amount of time taken was similar to the traditional process.

“I would definitely think it improved the product quality and not even to do with TRs, it even ... the number of bugs we are writing are reducing as you go along and things that we used to get before in the old feature test, wow, silly implementation and usability bugs or TRs, pointless things and these are the kind of issues that are ironed out at the very start.” [P2][I19-1: 17:45].

“The total time I think may be similar, but quality wise its definitely better.” [P3][I19-1: 18:39].

This team saw that in their experience communication and quick feedback were the biggest benefits of adopting agile methods.

“The major benefit I see is just communication, the simple thing talking about the product every day for 1/4 of an hour in the morning.”[P3][I19-1: 21:49]

“Early feedback on product quality is, the earlier you get the feedback, the cheaper it is to fix it.” [P2][I19-1: 24:15]

The “agile boundary” was mentioned as a negative aspect of agile. This is mainly because the organisation is large and because in this agile adoption step, product management, business management and many areas such as customer support were still working based on traditional methods.

“I think we are an Agile group within a non-Agile organisation. We are trying to do our Agile bit, we have non Agile external interfaces that are non-Agile, not as flexible as us.” [P2][I19-1: 27:33]

6.2.2 Team 2

Team 2 were based in Ireland. They were developing a new application but using a similar application as a design base and extending it. So they had some legacy code but were also developing a lot of new code and it was a complex application. They were a new team consisting of 10 developers, 3 of which were senior level and 2 of master level. Some of the team would have worked together in previous projects. This project in which they had just adopted agile methods had not gone well. While most of the difficulties for the team were related to introduction of new development technology, the adoption of a new methodology
in parallel added to the stress. However it is unlikely that the change of process significantly added to the difficulties of the team.

Team 2 omitted some documentation such as the Design Impact Specification, but unfortunately they also felt that as a result they did not put sufficient focus on the Design Analysis activity and that the work suffered as a result.

“You then tended to jump in and start writing code straight away. Again the analysis wasn’t really done - it was very haphazard.” [P1] [I20-34: 35]

“I think that because there was no artifact to be produced like in the old way there was the DIS- it had to be done, so there was always a formal phase that you did your analysis, where as with this there wasn’t, you could go and talk to somebody, if it wasn’t documented no one was going to look for it - it didn’t have to happen and.” [P2][I20-34: 49]

The team had a (badly designed) open plan area for Design but Systems, Design and Test were not co-located. Design and Test were on the same floor within the building, but Systems were not. They didn’t feel that more co-location would have been beneficial and they would have preferred not to be placed in an open plan area.

“I think if you asked everybody in the CE team they would prefer if we weren’t put into that sort of an open area because there is too much noise, too much distraction. To us if we did want to pair program or not even pair program if we just wanted to sit at the same location if we had an area like that that we could use if wanted rather than being put into it up front”. [P1] [I20-1: 21]

“Design and Test are on the same floor, systems are not. I think its close enough to be honest.” [P2] [I20-2: 20]

This team felt that the short iterations provided focus, but they had a big problem with the Sprint planning and were not meeting their Sprint goals, so they lost confidence in their ability to plan and there was a cumulative build up of pressure as a result of continually missing Sprint goals.

“When you get into the situation where you are not meeting your goals for each sprint and you are falling further and further behind - I think that’s where the pressure really comes, though it is kind of accumulative over a period of time where the whole team starts to feel we are not delivering, we are not delivering and we are getting further behind” [P3] [I20-3: 45]

The team didn’t really do pair programming as such. They worked in pairs to an extent but in the same way, as they would have done prior to Agile.

“There were times people would sit together, but it was no different from anything we would have done before.” [P1][I20-23: 20]
The reason they felt that they didn’t do pair programming was the time constraint.

“People are assigned to a story and they have to get it done and maybe in the long run it might be better to pair program but people think there is no way we’ll get all these stories done if we both work on one story.” [P2][I20-24: 18]

The team didn’t do any test driven development, even though they had a lot of new code as opposed to just modifications to legacy code. They felt the main reason was due to time pressure and they just didn’t have time to figure out how to really do Test Driven Development.

“That would have been the feeling all right. We don't really have time to spend analysing how are we best going to do this now because it was always a case of this has to be delivered by such and such a date. That's the top priority.” [P1][I20-26:47]

**Unit Testing** was lacking. They didn’t pair program or do any Test-Driven-Development.

The team felt that a lot of faults slipped through to test that could have been found in unit test phase.

“There was a lot of issues we would get a package and you’d load it on and straight away a fault and the problem was a blocking issue or something.” [P2][I20-32: 11]

The IP proposed an initial set of **User Stories** but more details had to be added or sometimes new User stories specified.

“The way it tended to work was - in the backlog I list all the user stories, if you like the name of the user stories and then as we were preparing for a sprint I would in a lot of cases have details already in the IP, but I would update them or re-write them or in some cases where the story was being added as a brand new one that wasn’t in the IP to begin with I would add the details separately.” [P3] [I20-17: 29]

Very often the team did not manage to actually complete the User Stories in accordance with their definition of “Done”.

“From a design point of view the definition of done was rarely complete. We would simply run out of time.” [P1] [I20-18: 40]

However they did not feel the problem was to do with the specification of the User Stories but a more general lack of experience with the Product and the Technologies.

“Well I think it all came back to just lack of experience with the product, the new technologies, lack of experience with that and not simply not having the time to try and get the story done” [P1][I20-19: 11]

**Stand up meetings** were no longer held on a consistent basis. The view from this team was that there were still some of the traditional ways of tracking Agile going on in parallel.
“I think there was benefit in the standup meetings but we had too many ways of tracking progress. ” [P1][I20-21: 50]

The Stand up meetings did get sidetracked and took up too much time.

“...but they did get side tracked quite a lot actually.”[P3][I20-22: 36]

**Continuous Integration** started late in the project, but the build process was complex and due to lack of resources to simplify the build, the practice of Continuous Integration did not continue.

“There was a resource problem there as well because at the beginning of the project the build system that we started off with was, even as it was it was very difficult to get it going so I think there was two months where the build system was actually broken, completely broken and to get to a point where the existing system could be gotten to work again, there was a lot of effort put in and that only got them back to where they started.” [P3][I20-29: 45]

**Acceptance testing** was not automated because of difficulties in automating GUI testing.

“We did try a tool called squish for automating the GUI end of it but it was just, it was too difficult to automate that upfront because you almost needed the GUI code before you could automate it with that particular tool and then the changes, the GUI changed quite a bit and even when we did try to automate some of it from regression test point of view, but even it became very difficult to maintain it because small changes just broke it.” [P2][I20-27: 24]

The team had a Systems person who acted as **Product Owner**. Initially they did some **demos**, but towards the end of the project, the demos did not take place at all.

“We did at the start but towards the end of the project we didn’t.” [P2] [I20: 44:30]

The team found that too many people were involved in the meetings. It worked better when the Product Owner and a tester were involved.

“It kind of rambled a bit, so I think it was better when we got rid of that almost and even when it was only the two of us, I was demoing the stories and going through the issues and what the problems were so we were reviewing the work that was done in the sprint and getting that feedback. I think that was much better rather than having this big huge demo with the whole team.” [P2] [I20: 0:45:47]

**Retrospectives** were held at the end of each sprint but there was no discussion on the value of these meetings. The implication was that there were over riding issues such as the delivery of non-tested stories that undermined the usefulness of other practices.

“We would simply run out of time and what we would end up doing then was to deliver stories that even at a design level weren't really tested, they weren't done essentially but we would still officially deliver those. We were going then to retrospectives, we'd have our demo
This team felt that it was OK to try out **Agile methods**, but that it hadn’t really worked out for them. They had problems with the technology, with estimating and while these issues would also have manifested themselves in the traditional process, they now had the additional task of changing to a new process.

“In my opinion I would say no. I didn't agree with it. It was good to try it but I don't think it worked particularly well.”  [P2][I20-0: 10:18]

“We were taking on a new process when we were already being challenged with a fairly complex product, with new technologies that the team wasn’t used to and the team that was changing constantly, its composition was changing constantly.”  [P3][I20-0: 11:38]

When the team felt they were under pressure they reverted to the traditional way of working. They also felt that there was a lack of design analysis and a lack of support in terms of coaching.

“The typical process we would have an analysis phase, design would but when we got into the agile process a lot of the analysis almost got skipped. It led to a lot of mistakes being made.”  [P1][I20-0: 13:15]

“There was no coaching either which there should have been. There should have been expert coaching.”  [P3][I20-0: 14:21]

This team didn’t feel they succeeded with the **implementation of Agile methods** so while they saw some benefits in it, they had at the time of the interview reverted to the tradition process. They would try it again but on a smaller “safer” project.

“I'd need to be convinced why we should do it again. I'd need to be convinced in terms of someone to explain this is why we failed with agile in CE. These are the mistakes you made, this is how you can fix them and this is the commitment to actually fixing them because I think to fix some of the issues is going to be quite difficult. Difficult in terms of the organization, they will have to commit time and people and money to it. I don't know whether that's going to be a priority. As it stands I wouldn't want to do it again. I certainly wouldn't want to repeat the headaches of that project again.”  [P3][I20-1: 00:28]

“I think in hindsight it was probably the wrong product to do a pilot of agile development on.”  [P2][I20-1:01:53]

“I wouldn’t be inclined to give up on the agile process. I have worked in an agile wow before , ..., in a different company and it worked well for us.”  [P1][I20-1: 04:18]
This team didn’t feel they could stand over the **quality** of the code they delivered especially since they did not have automated tests and they also thought there were **time** related overheads attached to the introduction of the new process.

“Overall I think what was difficult for us to guarantee quality was that we didn't have any automation really, that was one problem.” [P2][I20-0: 50:15]

“There were certainly overheads associated with adopting agile that we wouldn't have had, so the obvious question is could we have better spent that time. I don't know.” [P3][I20-0: 51:19]

They saw quick feedback and faster turnaround of faults as **advantages of agile methods**.

“I think the lead time between finding a fault and getting a fix was much shorter. We turned around faults much quicker.” [P2][I20-0: 56:33]

The team felt that the lack of a design analysis phase was a **negative aspect** of agile methods.

“The whole analysis phase didn't exist.” [P1][I20-0: 57:37]

### 6.2.3 Team 3

Team 3 were based in Ireland. One of the participants also took part in the one-to-one interviews. The team developed two new applications and some team members were also dealing with modifications to legacy applications that had complex dependencies to other Product Areas outside the organisation.

Team 3 did not experience any real change in the level of **documentation** produced. They did not do the formal Design Impact Specification, but documented their Design Analysis using a white board. The team itself did not have any problems with this. However the System Test Group, which was outside the “Agile Boundary”, did look for an alternative to the Design Impact Specification, as they needed the information about the design to “tweak” their test cases.

“We didn't have a DIS or any- no - model stored which I- for the product that was in it - it wasn't that you couldn't transfer some knowledge in a whiteboard session and the other person could grasp it and so on. We didn't have to do those ones and the only way they would be missed or the only way it was expressed to me they were missed, the system test guys who basically weren't in on the daily meetings, when it came to our first delivery to them, they wanted somewhere for us to outline the differences between what is written verbatim in the IP and we saw a problem with it – if it was illogical or it just couldn't be done or we changed
the design in some way - they felt they would have been better if that kind of stuff or a lot of that kind of stuff would have been captured in the DIS and they could have written their test cases from that. [P1][I2-57: 25]

Systems, Design and Test were not co-located, but were fairly close and their work area was not open plan. They had suggested that they sit closer together but because of departmental budget constraints that did not happen. The team did acknowledge that it was a disadvantage not to have the spontaneous communication that would result from co-location.

“I think it would certainly help if we were sitting co-located because it is more fluid, you can quickly access people” [P1] [I21-6: 32]

The team felt that initially perhaps they should have completed some Spikes instead of launching into Sprints. So the initial period was a bit stressful while trying to find their feet with a new process, new interfaces and features. However they expressed the opinion that working with shorter deadlines and iterations was positive.

“Yeah I think it helps to have shorter deadlines, say two weeks, setting a date draws a line in the sand for getting certain things done by a certain date and when that date is reached you have a good measure as to how things are going depending on what’s been completed, what stories have been completed or not completed so I think in that way it is good.” [P2] [I21-10: 52]

This team didn’t do any pair programming mainly because it was not in their ambition level, even though they were free to do it is they wished. There was no real limiting factor except that it was not requested.

“I didn't hear of anyone doing it and nobody was looking for it I guess.” [P2][I21-39: 45]

The team did some Test Driven Development, but they felt that they needed a lot of stubbing and mock code and that a type of component based testing was more useful. They unit tested to a point but ran into problems trying to stub some service classes that were used.

“Where it was relatively straight forward we did use the TDD to kind of basically design the test first, the class itself and make it easily initiated and instantiated and tested the interfaces on their own with stubbing or mocks but for the reasons I mentioned earlier on CORBA, Naming Services, basically because we were inventing from scratch to figure a way to test it, various members of the team I would say deliver tests that weren't so much unit tests as component tests.”[P1][I21-44: 40]

This team didn’t pair program much and their Unit Testing was compromised because they found it difficult to write Unit Tests without other software components being available, but they felt that Agile provided more focus on Unit Testing and code coverage improved.
“So it meant that the unit testing wasn't a last gasp effort, but there is no doubt that in so far as we compromised and have brittle unit test cases and have component tests instead of unit tests, the design principles have suffered but that we did get code coverage.” [P1][I21-47: 30]

In this case, the IPs were written before the Agile adoption started, so there were no User Stories specified in the IPs.

“The IP was initially approved by TC-CNOS or TC-OSS and they hadn't used the Agile template.” [P1][I21-26: 18]

The team had difficulty in specifying User Stories due to component dependencies and coupling.

“The previous one I was in was a layered architecture and it was possible to provide a unit of functionality without all layers in place where as this one NTCDF that was not the case, you had to have a lot of components in place before functionality was... so the whole user story thing was hard to apply to I found and there was some difficulty.” [P2][I21-30: 88]

The team consistently had the stand up meetings and found them beneficial. Work was needed to keep them on track.

“There was feedback that maybe they ran a bit too long sometimes.”[P1][I21-36:30]

“But we always had them, consistently had them and its good for - I'm thinking of NTCDF and NETCONF - keeping track of what is going on and being more aware of what is happening.” [P2][I21-36: 40]

The team had daily builds with unit tests ran as part of the build.

“Yeah we did we had daily builds, unit tests run and these unit tests and we would know the result of the daily build and the running of the junit tests.” [P2][I21-53: 46]

Some of the test cases were run towards a running server, as they were more component-based tests.

“When a build was made it was transferred and certain test cases ran towards the running server part on a real box which is getting to the component part - it really is - but that was because when the person was trying to design their unit tests for the area they were developing they couldn't at the time come up with a way to test their CORBA client without having a CORBA server running somewhere.” [P1][I21-55:08]

Some Acceptance testing was automated but there was questions over the re-work required because of the way the User Stories were split

The team had a Systems person who acted as Product Owner. They had a backlog and prioritisation was performed as part of the sprint planning, but they felt they were limited in the effectiveness of this.
“We had the systems person was the product owner and he prioritized.” [P2] [I21: 1:06:39]

“But they just juggled priority from within a list. From what I gather.” [P2] [I21: 1:06:43]

Only one demo was carried out for the duration of the project.

Retrospectives were held at the end of each sprint.

“Yes we do retrospectives at the end of each. We list what has gone well. What didn't go well and what to do next time.” [P2][I21-1:08:54]

While this team saw that Agile methods was a good way of working they expressed the opinion that the adoption was too fast and that the process was evolving as they were going along and this has not improved.

“In my opinion it was quite an ask and go from 0 to 60 with it and feel comfortable with it as much as you can at the end. I wouldn't be comfortable in somebody saying you are a fully competent practitioner now because I think there were too many areas where I would score myself pass or less and I wouldn't think I'd be alone.” [P1][I21-0: 20:22]

The team wanted to continue with agile but there were areas that needed to be worked on to improve it, for example, in the area of unit testing.

“I was in two different projects and yes on the whole.” [P2][I21-1: 27:07]

“I would say we should review the way - the aspiration level and concentrate on area that have to be gotten right.” [P1][I21-1: 27:16]

This team had the impression that code quality had improved and while the new process added some time initially after adoption that would be made up later.

“I think overall yes it has (improved quality).” [P2][I21-1:12:01]

“Initially maybe (takes more time) but if we did everything properly there would probably be a ramp down.” [P2][I21-1:12:34]

Faster feedback was the biggest benefit of agile methods.

“Able to prioritise better and the more interaction between test and design and getting quicker feedback in design and being able to give test something as soon as it is ready” [P2][I21-1: 17:12]

Difficulty specifying user stories at the correct size partly because of the product architecture was mentioned as a negative aspect of agile.

“The whole user story thing you know how the this is particular to the NTCDF project they architecture being horizontal clashing with the whole user story model.” [P2][I21-1: 18:25]
6.2.4 Team 4

Team 4 were based in Ireland and were a small team working with legacy code. All of the team had been working with the code for more than 1 year. The team consisted of 7 developers (1 master level and 1 senior level). The application was a legacy application and it could be classed of medium complexity with high dependencies to other areas.

The Design Impact Specification was done in a less formal way using a wiki page and they felt that the document in this format was valuable. That was the only change with respect to documentation.

“The only one that has changed is the DIS which we basically do the same it is a little bit more flexible now. We do it on our wiki page and we strive to have any kind of change documented in the wiki page so we probably do even more documentation but it is good documentation, we like.” [P1] [I22-21: 20]

In this team Design and Test were reasonably close (a few desks away) with Systems a bit further away. They felt that the physical separation with Systems was not helpful.

“There is a separation there which doesn't help.” [P1] [I22-1: 11]

The team felt that it was possibly a bit more pressurised but not necessarily in a negative way as the focus was on progress.

“I would say more pressurised yeah but I wouldn't say that is a negative thing either, the emphasis is on making sure you are making progress.” [P1] [I22-1: 47]

This team didn’t do any pair programming again because it was not in their ambition level as defined by the organisation. They were no issues with people not wanting to do it.

“As I said it is not in our ambition at the moment so nobody has been told to do it.” [P1][I22-14:01] [Q. People aren't saying that they don't want to do it?] “No.” [P1][I22-14: 15]

The team unit tested about 35% of their code. They had problems to unit test the GUI code.

“We still do a lot of retrospective unit tests and do it the wrong way round basically still” [P1][I22-14: 57] They were not really doing Test Driven Development yet, but were committed to getting into that way of working.

“Everybody might have got one course, but we now think about Dojo's and getting really involved and really trying it so really trying.”[P1][I22-15: 15]

When a Story was assigned, one or two designers were assigned to code review it. The team were trying to no longer let code go forward to test until it has been unit tested and reviewed, so the situation with unit testing and code reviews was improving.

“We still try to improve that and I think that we are closer now that we don't actually let it go forward until the reviews.” [P1][I22-16: 40]
Sometimes the **User Stories** were specified in the IPs, if it was an internal IP. Other IPs were written external to the team and would not have User Stories.

“There are different ways sometimes they are in the IPs, sometimes they weren’t.” [P2][I22-11: 40]

The team consistently had **stand up meetings** and again keeping them on track was the issue. Project managers were having some “project meetings” that were seen as duplication. “We have them daily yeah. The problem is trying to keep them shorter. Project managers is doing project meetings but we'll get that sorted.”[P1][I22-12: 40]

The team had a **build** of their own code running every 15 minutes with automated tests. However it was only their own code and dependant applications were not included in the build. “A build is done every 15 minutes - again it is only our own code on the previous release of the other code - that's the way we are running it at the moment.” [P1][I22-19: 35]

**Testing was not automated** yet, but the team had planned to do so.

“Usually we are like trying to get that the automation is going - to run in parallel but it is just not that far yet.” [P1][I22-18: 18]

A Systems Engineer worked with the team but they did not see themselves as having the **Product Owner** role. This was mainly due to a lack of allocated time.

“So we don't - I remember asking this from Patrick as well do we have a Product Owner and I don't think we have that role in our team at the moment.” [P1][I22-25: 52]

The team had some **demos** but because in their most recent design, there was no end user functionality changing they felt that it was not applicable. There was nothing “to see”

“We have another little issue there. We have two IPs now where there is basically no end user functionality changing so therefore the whole... That's another thing that sometimes just doesn't apply to use.” [P1][I22-27:01]

The team were not maintaining a product backlog.

“The only thing we have is our backlog from our sprint that you have people working on, but that's not what you are referring.”[P1][I22-27: 40]

**Retrospectives** were held before the sprint planning for the next sprint.

“Yeah. We kind of combine them at the moment - we would always have a retro and sprint planning straight after each other. We go through all the finished user stories in the retrospective and people comment for every user story how it went, not so well etc.” [P1][I22-0: 29:08]

The team said that the **adoption of Agile methods** was the correct thing to do and they did not feel that the adoption strategy was too aggressive.
“I am very positive and like I said I did a poll within the team everybody was positive, there was one neutral but that person was more saying we are not really agile yet and I would agree. We do need to kind of - we have to adapt the process to our particular circumstances.” [P1][I22-0: 4:54]

“I mean the way we did it we did some of the XP practices before we did the agile process, like re-factoring, well not proper pair programming, but used more of that kind of stuff before.” [P1][I22-0: 8:55]

They wanted to continue working in an agile way but were concerned, as resources are moving away from the team.

“Nobody wants to go back - most people are saying they would like to go forward.” [P1][I22-0: 45:02]

“There is a lot of people moving and changing, feature test is moved here and even then the features that are going in there is not a steady stream of features coming into PMS.” [P2][I22-0: 48:02]

They thought that their code was structured in a better way and that it would be more maintainable. More time was consumed as a result of re-factoring work and legacy code.

“I am happy that our code is much easier to maintain. I wouldn't guarantee there are less TRs. I just know that if there is a TR it will be a lot easier to fix but I have the impression there is less TRs” [P1][I22-0: 38:18]

“I would say slower. It is slower because we do much more re-factoring and there is an awful of legacy code which kind of but we are happy that the amount of re-factored code that we produce is much higher.” [P1][I22-0: 41:30]

This team also said that faster feedback was what they thought was the biggest advantage of agile methods.

“The day to day knowing where you are, how you are progressing, the whole test drive design, having the unit tests there.” [P1][I22-0: 45:34]

This team felt that they had not achieved their goals with regard to TDD, as it was not fully implemented yet and this was a negative aspect of agile from their point of view.

“Maybe the most thing people are complaining about is that we don't have a fixed content of a sprint basically and that we are not doing full TDD either.” [P1][I22-0: 44:46]

6.2.5 Team 5

This team was based in Sweden. The team six developers two of which were senior level. The main team members were established in the application area with most of the developers
working with the product for more than 6 months. They were updating a legacy application that could be considered of medium complexity.

This team did not experience any real change in the level of documentation produced. They did no formal Design Analysis documents prior to Agile. They used a wiki to document some “design rules” and hints for coding. They also have a good level of Javadocs for describing the code.

“But we try to document more like- I wouldn't say best practices but some information depending on different types of classes we have, different parts of the product, how we should implement them, hints and tips for mistakes to avoid, we did that on our wiki page” [P2][I23-45: 40]

The team had an open plan area, but the testers were not in the same area. Usually during Acceptance Testing the testers came to sit in the same area. Systems were close by, but again not in the same area. The team were looking forward to moving to an area where they would all sit together, but that was cancelled due to organisational re-structuring issues. “So it would be better if we were completely in the same area.” [P1] [I23-2: 02]

They didn’t think it was any more pressurised, apart from the fact that testing required a bit more planning due to the shorter iterations.

“I wouldn't say (more pressurised) so but that is probably my personal opinion.” [P1] [I23-3:06]

“No for me its been a bit more planning - which tester is going to do what - with this short iterations and then we have the more traditional project plan with long feature tests periods.” [P2] [I23-3: 20]

This team practiced pair programming up to 90% of the time, according to their own estimation.

“Yeah we use pair programming a lot especially the last two months where we had lots of new people, it becomes the common way of working since we had to teach quite much and everybody is still working pair programming.” [P1][I23-22: 25]

“I would 90% at least.” [P1][I23-22: 54]

“To start we used to do that (experienced with in-experienced) since we had new programmers, the first programming job they ever had and the product is quite complicated so then we had to have that but now we stopped with that kind of rule and they team up as they want. Sometimes they team up to less experienced and that is also good because they can be the driver. If you always work with a more experienced you always follow.” [P1][I23-23: 20]
“Someone that said “I don't like it” and he is one of the senior guys but on the other hand he is really good on the product so he paired a lot during teaching and now I think he is working quite a bit pair programming.” [P2][I23-24: 44]

This team did a lot of Test Driven Development. Their estimate was 95% for new code and that they have 60% to 70% overall code coverage. They have spent time on testing frameworks.

“We had people spending a couple of weeks to build testing frameworks to go around these problems but we do mostly TDD and we have some old legacy code.” [P1][I23-28: 45]

They felt it was an advantage in this case to have new developers in the team because they adopted the Test Driven approach more easily.

“I would say the pro was that we had lots of new designers so we could say this is how we work, so actually I think the new and inexperienced are the best people on TDD because they never did anything else.” [P1][I23-30: 32]

As the team implemented more pair programming, they no longer carried out formal code reviews. They felt that pair programming was more effective than code reviews.

“If you pair program it - you check each other and you can't really ignore these things and then design is much better usually when you pair program and the naming of methods, variables.” [P1][I23-16: 40]

Some IPs had now been written that specify User Stories, but design were still working off IPs that do not contain User Stories.

“At the time of speaking we have not got an IP with User Stories in” [P1][I23-16: 00]

They didn’t have too much difficulty specifying the size of the User Stories, expect in one instance.

“The problem was we had a new node and we had really bad information about the user interface. That was also developed at the time so when they wrote the IP, the interface wasn't available and they really had no good communication on how it should work” [P1][I23-17:00]

The IP had been written too far in advance, so the information was too old to be reliable.

“If we done it real agile, they would write the IP at the moment when we were working on it and say hey in two week we have to do this. And we would say ok that seems reasonable but now we got something from the bottom drawer and we should do this”. [P1][I23-17: 54]

The team had been improving the way the defined the acceptance criteria for the User Stories.

“At User Stories there is always a bit of problem to specify the acceptance criteria. We have been working a bit on that because that is something that is easy to skip” [P1][I23-19: 20]
The team had the **Stand Up meetings** and made an effort to keep them focussed especially when the group got bigger.

“We keep them quite good to the point. We are having a quite large team now so we even restricted them a bit further. In the beginning when we were much smaller we could speak a bit more about design and stuff, but now a lot of new people join and most of the feature testers and product owners and product managers comes and sometimes you can drift into other discussions that are not supposed to be there or they don't want to listen.” [P1][I23-20: 58]

**Continuous Integration** was working quite well and additional work was being done to install the package.

“On every check in we have automated build systems that builds the system and runs all unit tests. That's about 3000 unit tests and that is quite fast - about 15 minutes and then we are starting to implement a nightly build, building a package, installing and running some acceptance tests, but we only came to the point where it builds the package and installs it and then we stopped because of the migration to BuildForge.” [P2][I23-38: 49]

The team had started to introduce some **automated testing** and had some time allocated for working on automation of test cases.

“We have this work package to develop regression and stamping tests and also to find a way to how to automate the AC test to some level. We have only come that far so we have, I don't have the figures in my head, but I think about 20 automated stamping tests testing legacy code using Squish.” [P2][I23-37: 36]

The team saw the **Product Owner** role as a role that was split between the Systems manager and the IP authors. The Systems manager performed the prioritisation of items and the IP authors answered more detailed technical questions.

“We have one - I think she is the System manager, but she does the prioritisation and participates in all Sprint planning’s and so on. But she is not very technical. So when we need answers for technical issues we usually go to the writer of the IPs” [P1][I23-1:00:45]

They also have demos but the team was disappointed with the level of participation in the demos especially from some stakeholders, which were other product units that would use the functionality.

“So the most important I think is to get the stakeholders to participate because they are ones that want this feature and they are the ones supposed to have issues and comments on what we have implemented.”[P2][I23-1:06:39]

They felt that this might be because there was not always an end-to-end feature to demo.
“Sometimes we didn't because we hadn't anything exciting to show them. Usually they want to see the whole feature and not like since last two weeks we have been implementing this button. And if they see the value with that I think they will come.” [P1][I23-1:07:29]

**Retrospectives** were held at the end of each sprint.

“We have demo and retrospective before lunch and after we have the planning and usually it is 2 hours.” [P1][I23-1: 12:13]

While the team agreed that **Agile way of working** was good, they had a concern about the fact that they were driven by non-Agile stakeholders and that this might compromise the way of working.

“I have seen some things that I am not that impressed of from my point of view. In the OSS we don't run our own business in because we are driven by stakeholders that are not agile.” [P2][I23-0: 9:43]

“For me personally it is better to do all things directly rather than stretch it out because when you stretch it out I think it is more common that you don't get to the goal.” [P1][I23-0: 7:26]

This team wanted to **continue working with Agile methods**, but there were some improvements or refinements needed.

“Not go back to what we had. Maybe take a few steps back within the agile introduction and be - I felt it hard to for example with the QD process the points of view from us working daily with this has not come up and accepted as much as I wanted to. Its have been a distinct line between us working daily with this and the management working with the processes.” [P2][I23-1: 36:59]

Team 5 also felt that the code had a better structure and that a **quality** improvement existed. From a **time** perspective there didn’t seem to be an obvious difference.

“I would say from the perspective of the code how it looks and if you break something you know it. I would say on that level it is much better.” [P1][I23-1: 16:48]

“I wouldn't say we have any evidence of improvement but no evidence of getting worse either. I think we have really bad measurements for deciding what’s on time and what’s not.” [P2][I23-1: 19:13]

The team mentioned the closer co-operation between the developers and the testers as something they felt was most useful and the **biggest benefit** of agile.

“On the team basis I feel this integration of test together with the design team - that closer work is a great benefit. We can we are we can learn much from each other in that way and as
I said before this we have the possibility to capture the unnecessary fault much much earlier.” [P2][I23-1: 32:55]

It was felt that while there was a lot of focus of automating test, it was important not to de-value the “traditional” role of a tester in running tests and this was a negative aspect of agile.

“I have a feeling it's a lot of focus on the test automation and I think it is important do not forget what a live tester can see and think because automated tests tend to test the system as it is supposed to work and not go outside the frames so it is very important to keep testers and the manual test view as well.” [P2][I23-1: 35:33]

6.2.6 Team 6
Team 6 were based in Sweden. This was a new team with 6 developers, 2 of which were senior level. There were developing a new application and it was of low complexity.

As in the case of Team 5 this team weren’t previously doing formal Design Analysis documents, so they experienced no real change with respect to the level of documentation.

“Well we are doing design documentation so I think, Design Specification, we are doing updates during the sprints for those and also for the CPI [Customer Product Information]. We are trying to keep the number of document down anyway but I don't think that is anything to do with agile.” [P2] [I24-17:18]

The team had an open plan area, with testers and designers co-located and systems were close. The team would have liked if System Test were closer to them since System Test was carried out in a different country.

They didn’t find the environment any more pressurised than before.

“I don't think it is more pressure doing it this way. It has been some sprints where we have been over allocated but in general it is not adding any more pressure.” [P3] [I24-3:00]

They did pair programming from time to time but not frequently.

“We are doing it from time to time but not frequently.” [P1][I24-10: 35]

The team were trying test-driven development and were using it about 20 – 25% of the time in their estimation. They thought it was beneficial but found problems because they are dependant on another product.

“Its good to do it but I don't think it suits every time. We are quite dependent on another product and so on.” [P3][I24-12: 19]
The team had **informal code reviews** and even though they did not do that much TDD, they had code coverage up to the Qrank (Quality Ranking ambition set by the project) specified levels.

“We are doing code reviews or desk checks when you are coding, not meetings like inspections.” [P1][I24-11:08]

The team hadn’t had **User Stories** specified in the IPs to date, but planned to do so in the future.

“The plan going forward was to have user stories directly in the IP. That hasn't been the case for all the features lately but that is how we plan to do it going forward ” [P1][I24-07: 27]

They experienced difficulty specifying the User Stories at the correct level and size.

“But also we tried to use user stories but we also need to learn how to slice these in a good way which is something you constantly need to improve and now and then we have some struggle but” [P1][I24-07:41]

They also became aware of the importance of specifying the acceptance test criteria.

“Small enough, clear enough and in the last now I think we try to define test cases more directly with the user stories to make them clearer” [P1][I24-08:03]

**Stand Up meetings** were running and working well.

“We are trying to keep them short and on topic. I think it works quite good. Sometimes of course there will be discussions but usually not.” [P1][I24-10:15]

**Continuous Integration** was working with a limited amount of tests being run.

“Yes some tests are run all the time” [P2][I24-15: 37]

**Automation of Acceptance Testing** was just beginning in this team and in the last Sprints they had been doing that for some User Stories.

“In the later sprints for my team at least we have tried to do automated for each and every user story, the last sprints.” [P2][I24-15: 28]

“Yeah it differs, but that was also the ambition to automate the acceptance test but of course there may be a few that not might be suitable for. But it would be a criterion to do that within the sprint if that was suitable. That was what we strived towards.” [P3][I24-16: 14]

This team did have a product backlog and it was maintained, but they felt that they had not settled how the “**Product Owner**” role should work.

“This is an area for where we haven’t really settled how it should work. When you do prioritise you should you need to consider more than the customer - you need to consider technical risk and system test possibilities and other stuff, transfer.” [P2][I24-0: 25:05]

The team had three-week sprints and performed a demo at the end of each sprint.
Retrospectives were held at the end of each sprint and generally lasts 2 hours. They felt that the adoption of Agile methods was the correct thing to do and they also felt that the transition was gradual enough. They had some seminars and discussions prior to the adoption of the methodology.

“My view is that everyone thinks its good.” [P2][I23-0: 4:24]

“Its more open and more people get involved.” [P1][I23-0: 4:30]

“I think it has been gradual. We haven’t taken on everything at once. We had seminars and short courses and discussions.” [P3][I24-0: 4:55]

The team wished to continue to use Agile methods. “I think it would be a big disappointment if we said we would go back to some old way of working.” [P2][I24-1:02:30]

“That's a general thing. The designers .. , I think this method agile or the scrum method in some way acknowledges the difficulty with software development in a way that they appreciate. They feel that this is done for software development and that's why and of course they difficulty is how you translate into some kind of software factory picture from that like upper management might want to have on software development.” [P3][I24-1:02:47]

The team thought that quality might have improved, but they were previously doing a lot of unit testing and continuous integration prior to adopting agile, so they had already been focused on code quality. Interesting they felt that pair programming was efficient because we already saw that some teams quoted lack of time as a reason for not doing pair programming.

“I mean we are - we had already the mindset in the people working here so we were doing the unit tests, we were doing continuous integration. Perhaps it has improved somewhat but I think the mindset was already in the people here.” [P2][I24-0: 49:24]

“What I had seen is that pair programming is very efficient, because you get two persons doing the code and also when you have sick leaves and stuff like that you are not having one expert” [P2][I24-0: 51:42]

“Close co-operation. That's really boasting the productivity.” [P1][I24-0: 52:47]

They expressed the view that they were more motivated and that they were working together as a team as the biggest benefit.

“Well involvement I would say and motivation and productivity as well and all together, sitting together, co-operating I would say. That is the most important.” [P2][I24-0: 58:20]

Regarding the negative aspect of agile they mentioned that after feature test, the code is delivered to the next test phase which is System Test. System Test were still following a
traditional approach so this team felt they would like if more of the organisation especially System Test were under the agile umbrella. This would reduce the amount of agile – non-agile boundaries.

“The next step for us would be to have Systems Test more integrated and getting the code earlier.” [P1][I24-0: 49:52]

6.2.7 Team 7

Team 7 were based in Sweden. It consisted of a team of 8 developers (2 senior level). They majority of the team had been together for more than one year. They were working with a legacy application that could be classed as medium complexity but with high dependencies to areas outside the team.

Apart from not doing formal Design Analysis documents, the documentation level was similar to what it was prior to Agile.

“We don't have any design specification or anything like that. We still have the same documents we had before Agile, CPI, PRI, and programmer guide.” [P2] [I25-24:02]

They had an open plan area. A feature tester sat with the design team. Systems were close by and participated in sprint planning, stand up meetings at times and in user story meetings. They had no issues with this arrangement.

The team didn’t experience any difference in terms of pressure due to short iterations.

“I would say no difference.” [P1] [I25-2:15]

This team practiced pair programming up to an estimated 80% of the time. Pairing was usually inexperienced with experienced and the pairs changed frequently.

“From the beginning we had fixed pairs during the sprint and we decided during the sprint planning which pairs we were going to have. Now we have more like dynamic pairs that can shift during the sprint so we don't have any fixed pairs anymore. Its difficult for some people because they are off to meetings, for a couple of people it is difficult to pair but 80% of the team is pairing.” [P1][I25-14: 11]

The team applied test first as much as they could and were testing up to the level of the Q Rank. Their estimate was 40%-50% depending if there was legacy code of GUI code involved. So they had adopted TDD but there was more work to do.

“We definitely try to - its not always possible or convenient but we try to. We had a couple of seminars working with it, our agile coach has a number of seminars that the designers went to, not fully got there yet in terms of test case first.” [P1][I25-17: 15]
The team felt that the pair programming compensated for code reviews. “We feel the pairing compensates.” [P2][I25-16: 12]

Again the team didn’t have User Stories specified in the IPs to date, but planned to do so in the future.

“Also systems now they start to produce User stories in the IP so it takes time to adopt.” [P1][I25-08:13]

The Systems representative felt that instead of putting too much detail on the User Stories in the IP it was better if Systems worked closely with Design on the formulation of User Stories a bit later on.

“It might be the case that even if the IP would have contained User Stories just to start up with they would need to be reformulated and I do not want to say that this is a discrepancy between how Systems think and how the Design teams think but on the level its really useful to do it that way.” [P2][I25-10:35]

Stand Up meetings were happening daily and were now running for an average duration of 15 minutes down from a longer duration initially.

“Well we have them every morning. From the beginning when we were new to this way of working, they went on for quite a while those meetings, everybody wanted to say and discuss and it never ended. We were aware of that and we started to control it and now the length is around 15 minutes.” [P1][I25-13: 27]

It seemed that the team were building their own code that was checked in nightly and running some unit tests. They did not have much automated testing at that point.

“We build every night what is checked in into the integration branch, and unit tests every night.” [P2][I25-21: 37]

The team had automated the testing of some of the key features so the tests could be run daily or nightly. With respect to acceptance testing, the tester manually ran the test first and tried to automate it later but that didn’t always happen.

“The way its been working so far we have identified which kind of features and during the user story breakdowns we try to identify those kind of automated test cases but so far we have just implemented two or three. [P1][I25-22: 27]

The team had a number of sub-teams each which their own Product Owner, who was a Systems person. All of the Product Owners worked together with the team leaders to prioritise and maintain the Product Backlog.

“Together with team leaders we pick and chose and prioritise what team should do which items in the backlog.” [P3][I25-0: 49:39]
They also did **demos**, but the usefulness of the demo was not discussed.

**Retrospectives** were held at the end of each sprint after the demo and before the planning of the next sprint. This team felt that there was a lot of overhead between the demo, retrospective and planning for a two-week sprint. This resulted in almost two full days of “non-coding” activity each sprint.

“Because we are having a lot of overhead in a sprint we have 2 days of meetings, one day with the demo, the retrospective and then another day where we read the IP’s for new sprints and we have sprint planning and we have 2 days of overhead for each sprint.” [P3][I25-0: 55:16]

Again this team said that the **introduction of Agile methods** was correct and that the adoption was not too aggressive for them since they were already using some XP practices.

“The designers at least are very positive to agile. I haven’t anybody don't like the new practices.” [P3][I25-0: 8:00]

“And it wasn't so big a change as we already had the XP practices. As well the Scrum, the daily stand up, the burn down chart, the planning well it was a change, it is not that is was not a change but it was not so aggressive for us maybe.” [P1][I25-0: 8:01]

“No actually how we worked didn't change that much it was just that we made things more visual. We could see how things works and what we had to do and so it was a really big positive.” [P2][I25-0: 8:27]

Here we have the response when asked if they would like to go back to the traditional way of working.

“No way I would say.” [P3][I25-1: 15:58].

The team felt that there were **quality** benefits but more **time** was used in the agile wow.

“For test I think the benefit was that we test early that you won’t have to write so many TRs because you if I talk about the pilot where we had the test resource working that you have a better quality at delivery to LSV. And drawback is that it is we can't afford it, it takes too much time, it costs more than what it takes.” [P1][I25-0:57:25]

“I think one reason why agile WOW was introduced was they looked at the RNO and that they worked in Xtreme Programming and that we had a little better quality than other Product Areas and so in that regard there is a measurable difference between working agile and non-agile.” [P3][I25-1: 12:56]

As the **biggest benefit of agile**, this team felt that the communication aspect of agile was superior.
It enables communication and that's fun. I don't like sitting alone in my room looking at a computer screen. It's more fun working in pairs and a lot of interaction around the table and a lot of interaction with your system guys and with testing." [P2][I25-1:03:30]

"It's only positive things I think. Much more interactions and discussions and I think we are finding flaws in systemisation much earlier because design are much more involved earlier."[P3][I25-1:05:00]

The team felt that a negative aspect of agile was that they were not truly “self-managing”.

“I think one core value or very important thing in agile is that you have a self managed team. So it should be up to the teams to decide how to work and you can say could work in pairs it is a very good practice and perhaps we can arrange a course for you to learn about those practice if you are not sure what it means. You shouldn't force the teams to work in a specific way.” [P3][I25-1:06:14]

6.2.8 Team 8

Team 8 were based in Sweden. The team consisted of 9 developers (3 senior level) and the team had been established for 2 years. They were updating a legacy application and the application could be classed as medium complexity.

Team 8 experienced no change in the level of documentation produced as a result introducing Agile Methods. They felt that all the documents they were writing were useful.

“I think all documents we write today are useful. I can't recall any that are not.” [P3] [I26-28:00]

“No we haven’t changed our documentation because of the agile initiative.” [P3] [I26-28:05]

This team had an open plan area, but design and test were not co-located, so the team viewed this as a drawback.

“This has been a minus. With the areas like this we couldn't fit testers and designers in the same area.” [P1] [I26-2: 02]

This team expressed a similar opinion to Team 1 in that the shorter iterations removed a level of uncertainty and reduced the pressure.

“It might actually be the other way around, that you are feeling less pressure now because you have a more controlled, you have your two week sprint you can plan your two weeks, previously, at the end of the long shipment you always saw how much backlog you had that you had to do before delivery but now you have smaller chunks to deliver and I think it is less pressure.” [P4] [I26-4: 10]
This team practiced **pair programming** up high degree. All work except builds and writing IP was being done in pairs. They pairs switched roughly twice a week.

“So today we switch pairs once or twice a week, not every day but every other day.”  

They had recently decided to limit pair programming to 6 hours per day because it was intense.

“Yeah it tends to be or it can be very intense so if you sit there pair programming for seven or eight hours a day that's very intense or very tough.”  

They have had no issues with people not wanting to pair program.

“But I have never met one designer that wants to leave us because we are doing pair programming.”

Some staff found the team area noisy initially, but the team themselves preferred to work in the open area now.

“I heard that some consultants in their first weeks that they have problems working in the team area but we are working in a team area for so long time so we can't work in separate rooms anymore.”

The team used **Test Driven Development** on all new code.

“Now we write all new code with Test Driven Development”.  

They experienced a steep learning curve when adopting TDD.

“You have to train to become efficient at TDD and you can't just start it one day. You have to get used to it.”

Pair programming was taking the place of **code inspections**.

“We don't have code reviews at all. We think that the desk checking is done by the pair programming partner.”

This team had **User Stories** in all the IPs to date and were able to break them down to sufficiently small items.

“So yeah all IPs that we write today they contain user stories.”

“So far they have been all smaller than a week, a couple of days.”

**Stand Up meetings** were happening daily and were working well. Again there was a conscious effort to keep them on track.

“Well we try to timebox it to less than 15 minutes but sometimes we start some design discussions as well. But we try to keep them short.”

The meetings have replaced the project meetings, but the team felt that some feedback from the project manager is possibly missed.
“Something you might miss is that before scrum we had a team meeting every week, with some of the same information as the daily standup, but we have also got some information from the project manager then I think you might miss the information from the project manager.” [P3][I26-15: 37]

There was a **nightly build** rather than at check in because the build took 4 hours.

“That is the main reason but I think there are plans in the future to introduce CI when we have decreased our build times. And perhaps build the code and then run unit tests after each change” [P3][I26-25: 44]

The level of **automated acceptance** testing was not defined, but only some Automated Acceptance tests were written for the design environment and the team had some time allocated to improving their automated test suite.

“So we automate some acceptance tests in the design environment and but in the real environment or installed environment there we only have regression tests.” [P3][I26-26: 20]

A Systems person performed the **Product Owner** role and again they prioritised and maintained the backlog. They did have **demos** but there was no discussion on the value or worth of the demo.

“I as product owner somehow explain and make sure that everybody understands and then when everybody is clear with the user stories I leave.” [P2][I26-0: 31:54]

**Retrospectives** were held at the end of each sprint after the demo. Changes to the process could come from the retrospective and the retrospective generally took one hour to complete.

“We have the retrospectives they can change even slightly the process.”[P1][I26-0:28:41]

“Demo is one hour and the retro is one hour.” [P2][I26-0: 33:51]

This team felt that the **introduction of Agile methods** was good.

“Yeah I think so, because Function Test has become more of a process of the design work and I think it is a bit higher quality when we get the delivery, so I think it is very good.” [P2][I26-0: 6:22]

Similarly to other teams they felt that if you introduced to practices in a more gradual way you would loose momentum.

“I think it is hard to get real change if you just do it one part at a time. If you change the whole way of working at the one time it can be hard in the beginning, but on the other hand you have to really learn a new way of working.” [P2][I26-0: 7:22]

The team wanted to **continue working with agile methods** and to improve on the use of the methodology.
“For a while at least to get the chance to get better at it to get more out of it - it was a good start - we became better and better so something is good about it that you can evolve in it and do good things.” [P3][I26-0: 51:42].

This team thought that the quality was better with no marked difference in the time taken. “Impression is that function test gets higher quality because it has already been tested to some point so we don't get stuck in the situations where we can't start testing at all because of some silly fault or build fault or all that stuff - a lot better now.” [P2][I26-0: 44:29]

“My gut feeling is that we are not slower producing new functionality at least.” [P1][I26-0: 48:21]

Team 8 mentioned the improved team working and the quick feedback as what they saw as the biggest benefit of agile.

We are more working as a group together, function test, systems and design in a way that was not done before and I think that is definitely a positive side. [P3][I26-0: 49:09]

“That’s very appealing part of it for me that we get quick feedback and we can adopt rapidly and also that we get more natural contact with the systems and function testers.” [P1][I26-0: 50:02]

On the negative side of agile they mentioned some overhead and extra work in function test planning as a result of the shorter iterations.

“The negative part is the planning for function test, as we work now that is hard to do”. [P2][I26-0: 50:43]

6.2.9 Team 9

Team 9 were based in Sweden. They were a new team with ten developers, which included one contractor and 3 senior level developers. They were updating a legacy application with large chunks of new functionality. Two or three of the developers were experienced with the base application.

The team felt that they did not do a lot of documentation. With regard to learning the System and knowledge transfer, they felt that the unit tests were very useful.

“Also something to not forget is the tests that are there with the code as a form of documentation. I mean they are explaining how the code is behaving and that is the most perfect documentation that you can have. Basically running that code and see what it does cause you have the expectations you have really what it is supposed to do. What is expected
from each and every class. That is very good documentation for the newcomers as well.

“[P1] [I27-41: 58]

They had a team room where the designers and function testers sat. They felt this worked well.

“You can hear all the discussions between the designers and designers and testers. You can hear everything that is going on, so it was good.” [P2] [I27-3: 45]

Systems/Product Owner initially sat in the team room, but moved out because they felt they were getting too involved in the technical details of the features and now sit in the team area for defined periods of time.

“And as product owner originally I sat with the team or maybe I sat with the team for half of the features but I found myself being too involved in the design specifics and the technical details. Instead I tried to let the team handle that internally and develop their own design and how to work with it and I moved out from the area and they had to come and get me and ask questions and also try to be in the team area for a while in the morning and a while in the afternoon. I think that way worked good for me”. [P3] [I27-3: 57]

The opinion of this team was similar to Team 4 in that any extra pressure due to the shorter iterations was more of a positive thing.

“I think it was positive. I didn't feel it as a lot of pressure because if we didn't finish a story that we should have finished it wasn't the whole world so but it is good to have a deadline when you are supposed to be finished. I liked it.” [P2] [I27-7: 24]

This team tried to do almost everything in pairs and was switching pairs practically every day. They found the pairing good for competence transfer.

“I think it was really good to spread the knowledge in the team also.” [P2][I27-25: 00]

They also found the pair programming tiring and intense.

“What might have been a problem is that is pretty intense to pair program so you get kind of tired some time.”[P2][I27-26: 38]

They generally paired experienced with inexperienced and have found no issues with personality conflicts or people not wanting to pair.

“No, no personality issues no.”[P1][I26-28: 14]

The team was using Test Driven Development on all new code. They tested 95% of new code. The also stated that it takes time to learn and adopt TDD.

“Yeah and sending them to a course for two weeks, two days is not enough either. You have to work with it for a while. You have to dig in to the code and really do it to learn and I know
from those other guys that have not worked with TDD before, now they see the benefit of it.”[P1][I27-38: 26]

This team felt that they were successful with TDD and that TDD improved the quality of the code.

“That is in general or whatever tip I can give is to have one good really experienced guy in the team that have worked with TDD before otherwise you will take short cuts probably.” [P1][I27-37:00]

Because the team used pair programming they didn’t do formal code inspections.

“We had pairs with one experienced and one inexperienced - it is not so important to have code reviews but if you have pairs with two inexperienced then it is good to have a code review.” [P2][I27-29: 44]

The team had User Stories in the IPs, but they still had to be refined during Design. They felt that they had managed the User Stories well, but it was key competence to be able to split features into User Stories on the correct level and size and to specify the Acceptance criteria.

“With user stories I think we managed quite well and kept them quite small.”[P1][I27-17: 23]

“Size is one thing and well defined acceptance criteria is the most important thing because without the acceptance criteria you don't actually know when you are finished and it is impossible to estimate it.”[P3][I27-17: 40]

“We shouldn't under estimate the effort needed to do good user stories and have knowledge about how to split them up into small deliverable items and I think it takes practice and working with them.”[P1][I27-18: 50]

Stand Up meetings were working well and the team were working to keep them focussed.

“I think we have done daily standup meetings - not sufficient every time but I think it has worked at the end very good at least. And the team has been very good - they said ok a time box should we really discuss this right now something like that.” [P1][I27-21: 49]

Continuous Integration was running. After every check in, build results were published on the web. Tests were run nightly.

“After every check in and the good part was that is was always on the web, so we always published the build so the testers could take it and verify it TR fixes very easily.” [P1][I27-44: 51]

It involved a lot of effort to get the Continuous Integration working, but the team felt it was extremely beneficial.

“It was a lot of work to get it up and running so that is could be something to think about to help other teams that starts up that wouldn't have to spend so much time.” [P2][I27-45: 20]
Automated acceptance tests were defined at the requirement level, rather than User Story level because it was taking too much time to write all the Acceptance Tests at User Story level and the tester was one week behind the designer.

“But for the GUI tests it was too fine grained and the testers were always one week after the designers with tests and so on. So after a while we decided to write acceptance tests for different requirements, which were broken down to what you would call epic story level or functional units. So I think that worked well.” [P3][I27-33: 34]

Systems took on the Product Owner role and seemed to work closely with the team and were aware that more was demanded from Systems than in the traditional approach.

“If we have the System managers, System Product Owner close to the team that is something that we can discuss during our journey.” [P1][I27-0: 55:35]

“Yeah and I mean a lot of things with the IS2.0 feature we actually figured out and redesigned during the execution phase because we found out new different criteria's or discovered the reality wasn't what we thought when we wrote the IP and so on. It was very good for flexibility. It required a lot more of systems though than what is traditionally needed.” [P2][I27-0: 55:41]

Retrospectives were held and there was evidence that they were working from statements such as the following when referring to improvements needed in other practices.

“That is something that came up in the retrospectives quite a few times.” [P2][I27-0: 22:07]

This team agreed that Agile methods was most appropriate for their project, even or especially with new and inexperienced developers.

“Yeah I agree with you we have a lot of answers with a lot of complexity within those that project - there were new guys in the team, inexperienced for SHM in general - we didn't know much about requirements - there was a new IP writer and so forth but you have some kind of isolated framework to work with. I think Agile was the most appropriate one.” [P3][I27-0: 13:28]

They also thought that introducing a number of practices together worked for them. However in some situations for example with legacy code it might not be possible to do a lot of Test Driven Development.

“I think it depends on the project, but with our project it was good to do all practices. If you have a lot of legacy code maybe it is hard to do test driven development. It is better to leave it out then.” [P2][I27-0: 14:40]

This team was in no doubt regarding their view on continuing with agile.

“I think you know the answer.” [P1][I27-1: 16:55].
They felt that both time and quality had improved. “Product quality definitely” has improved. [P3][I27-1:08:27]
“The initial design might have been slower but I think we got that back when we didn't have to fix a lot of TRs.” [P2][I27-1:08:36]
“I think it feels more efficient because when we are done we are done. You don't get a lot of other stuff afterwards that we have to do.” [P1][I27-1:09:16]

This team also thought that that teamwork and quick feedback as a result of agile was the biggest benefit of the methodology.
“I mean you work as a team, you have one goal for every two weeks and you complete something in two weeks.” [P1][I27-1:11:49]
“It is really nice to get feedback about what you have done.”[P4][I27-1:12:47]

When asked about negative points, this team said they would like to eliminate the agile to non-agile boundary between themselves and System Test.
“I mean the Integration &Verification part and the System, the whole process is not adopted for agile way of working right now, this quality door and so forth. [P1][I27-1:11:32]

6.3 Summaries and Discussion

6.3.1 Team vs. Individual View

After completing the analysis both at individual level and team level some observations are notable. First of all, both teams and individuals were very open and honest about their experiences. Within the team or group interviews consensus was developed and the statements of fact were sometimes stronger or more committed than in the one-to-one interviews. The feedback from the one-to-one interviews broadly concurs with the group interviews. The differences where they arise are largely due to the fact that the individual interview participants were drawn from teams in Ireland only and the five of the group interviews were with teams based in Sweden. There was a stark difference in the approach to pair programming between Ireland and Sweden in that the practice was far more widespread in Sweden. With regard to documentation, the traditional process used by the teams in Sweden was slightly different to the teams based in Ireland so the Swedish teams did not feel that they were producing superfluous documentation. The formalities around their Design analysis had not been so rigid in the traditional way of working. In Sweden some of the teams had very well laid out open plan agile areas and this was seen as very positive.
6.3.2 Documentation

Table 34: Did the amount of documentation reduce with Agile?

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Outcome Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No change but documents now mean more</td>
</tr>
<tr>
<td>2</td>
<td>Less design analysis document which had an adverse effect</td>
</tr>
<tr>
<td>3</td>
<td>No change</td>
</tr>
<tr>
<td>4</td>
<td>No change but less formality in design analysis</td>
</tr>
<tr>
<td>5</td>
<td>No change</td>
</tr>
<tr>
<td>6</td>
<td>No change</td>
</tr>
<tr>
<td>7</td>
<td>No change but less formality in design analysis</td>
</tr>
<tr>
<td>8</td>
<td>No change</td>
</tr>
<tr>
<td>9</td>
<td>No change</td>
</tr>
</tbody>
</table>

The evidence was that there was a slight reduction in the total amount of documentation but not by the amount they had initially anticipated. The teams based in Ireland (1-4) have anticipated a larger reduction in documents when agile methods were introduced. The other teams, which were based in Sweden, did not feel that they had much documentation that could be eliminated from the outset.

6.3.3 Physical Work Environment

Table 35: Opinion on co-location, open plan

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Outcome Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>More co-location would be better</td>
</tr>
<tr>
<td>2</td>
<td>Had an open area, would have preferred non open area</td>
</tr>
<tr>
<td>3</td>
<td>More co-location would be better</td>
</tr>
<tr>
<td>4</td>
<td>More co-location would be better</td>
</tr>
<tr>
<td>5</td>
<td>More co-location would be better</td>
</tr>
<tr>
<td>6</td>
<td>More co-location would be better</td>
</tr>
<tr>
<td>7</td>
<td>Design and Test co-located, System close, working fine</td>
</tr>
<tr>
<td>8</td>
<td>More co-location would be better</td>
</tr>
<tr>
<td>9</td>
<td>System, Design and Test co-located, working fine</td>
</tr>
</tbody>
</table>

The Open Plan area produced a negative response initially. This was in part attributed to the fact that one of the teams based in Ireland had an open area that was not well laid out. Two of the teams in Sweden had a purpose built open area with adjustable height furniture and plenty of space. Open Plan area was generally accepted to improve communication.
6.3.4 More Pressurised Environment

Table 36: More pressurised environment in agile?

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Outcome Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Less pressurised, more certainty</td>
</tr>
<tr>
<td>2</td>
<td>More pressurised, missing Sprint goals and deadlines</td>
</tr>
<tr>
<td>3</td>
<td>Less pressurised, more certainty</td>
</tr>
<tr>
<td>4</td>
<td>Less pressurised, more certainty</td>
</tr>
<tr>
<td>5</td>
<td>No more than before</td>
</tr>
<tr>
<td>6</td>
<td>No more than before</td>
</tr>
<tr>
<td>7</td>
<td>No more than before</td>
</tr>
<tr>
<td>8</td>
<td>Less pressurised, more certainty</td>
</tr>
<tr>
<td>9</td>
<td>Less pressurised, more certainty</td>
</tr>
</tbody>
</table>

Overall teams felt that agile introduced more certainty in terms of code quality, which resulted in a less pressurised environment.

6.5.5 Pair Programming

Table 37: Pair Programming

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Outcome Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Carried out 40% of time, no personality issues</td>
</tr>
<tr>
<td>2</td>
<td>Not carried out, time constraints</td>
</tr>
<tr>
<td>3</td>
<td>Not carried out, not in ambition level</td>
</tr>
<tr>
<td>4</td>
<td>Not carried out, not in ambition level</td>
</tr>
<tr>
<td>5</td>
<td>Carried out 90% of time</td>
</tr>
<tr>
<td>6</td>
<td>Carried out “from time to time”</td>
</tr>
<tr>
<td>7</td>
<td>Carried out 80% of time</td>
</tr>
<tr>
<td>8</td>
<td>Carried out “a lot”, no personality issues</td>
</tr>
<tr>
<td>9</td>
<td>Carried out 100% of time, no issues</td>
</tr>
</tbody>
</table>

In the first stage interviews, developers expressed the view that there was a benefit in pair programming mainly from a knowledge transfer point of view, but there was a much concern regarding personality conflicts. There was a considerable difference between the level of pair programming carried out in Ireland and Sweden as shown in Table 37 above. One possible explanation for this was that XP practices were previously used in some areas in Sweden so developers would be more familiar with the concept.

Four teams performed pair programming to a high level, 3 teams did not carry out the practice and 2 teams did pair programming about 30% - 40% of the time. Generally the pairing was between experienced and inexperienced developers. From analysing the second round of interviews the main barrier to using it was pressure on resources. The teams felt that
they could not afford the time to do it in all cases. The teams that did not pair program were asked specifically if there were people that didn’t want to pair, but the only reason stated for not implementing pairing was the upfront investment from a time perspective. The issue of personality conflicts and not wanting to pair had simply faded away.

### 6.5.6 Test Driven Development

Table 38: Test Driven Development

<table>
<thead>
<tr>
<th>Team No.</th>
<th>% TDD Performed</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>New/Modified Code</td>
<td>Struggling with mindset change, tests performed late</td>
</tr>
<tr>
<td>2</td>
<td>No TDD</td>
<td>Lack of time</td>
</tr>
<tr>
<td>3</td>
<td>Some TDD</td>
<td>Too much stub code needed</td>
</tr>
<tr>
<td>4</td>
<td>About 35% TDD</td>
<td>Problems with GUI code</td>
</tr>
<tr>
<td>5</td>
<td>95% of new code</td>
<td>New developers adapt quicker</td>
</tr>
<tr>
<td>6</td>
<td>About 25%</td>
<td>Problems with dependencies</td>
</tr>
<tr>
<td>7</td>
<td>40-50%</td>
<td>More work to do</td>
</tr>
<tr>
<td>8</td>
<td>All new code</td>
<td>Experienced steep learning curve</td>
</tr>
<tr>
<td>9</td>
<td>95% of new code</td>
<td>Found to be very beneficial</td>
</tr>
</tbody>
</table>

Again TDD was implement to a higher degree in the teams based in Sweden (teams 5-9), mostly likely due to the fact that they tried XP practices there in previous projects. All teams saw value in TDD and struggled initially with the mindset change of writing the test case first. Getting test harnesses in place for legacy code was an issue and the issue of GUI code, which is difficult to unit test. Some teams that implemented a lot of TDD said it was one of the crucial success factors in improving quality.

### 6.5.7 Improvement in Code Inspections and Unit Test

Table 39: Situation with code inspections

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Code inspection only if code is not pair programmed, still happens too late but improving</td>
</tr>
<tr>
<td>2</td>
<td>No PP or TDD, too many faults slipping though to test</td>
</tr>
<tr>
<td>3</td>
<td>No PP and Unit Tests not good enough, but increase focus on unit testing</td>
</tr>
<tr>
<td>4</td>
<td>Situation improving</td>
</tr>
<tr>
<td>5</td>
<td>Pair programming effective way of code reviewing</td>
</tr>
<tr>
<td>6</td>
<td>Doing informal code reviews</td>
</tr>
<tr>
<td>7</td>
<td>Pairing compensates for code reviews</td>
</tr>
<tr>
<td>8</td>
<td>Pairing compensates for code reviews</td>
</tr>
<tr>
<td>9</td>
<td>Pairing compensates for code reviews</td>
</tr>
</tbody>
</table>
Developers felt that in the old traditional process that time ran out for unit testing and code inspections and that by using agile methods pair programming and test driven development should improve the situation.

There was variation between the feature teams with respect to the amount of test driven development and pair programming performed, as outlined in Table 37 and Table 38. Some teams didn’t carry out any pair programming and other teams did quite a lot. Some teams only performed code inspections when the code was not pair programmed. There were issues with the availability of test harnesses for legacy code and difficulty automating GUI tests and while there was still work to be done in this area, the focus on quality heightened dramatically. And teams that didn’t have such high code coverage were becoming aware that they needed to reduce dependencies in they system and have less coupling in order to improve the ease of testability. So the original expectation of improving the code inspection and unit testing was realised to an extent when pair programming was performed and where there was a high level of code coverage with test driven development.

### 6.5.7 User Stories

Table 40: User Stories

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User Stories not defined by Systems</td>
</tr>
<tr>
<td>2</td>
<td>Sometimes defined by systems but more detail had to be added during design.</td>
</tr>
<tr>
<td>3</td>
<td>IPs written prior to agile adoption. Difficult to specify User Stories due to system coupling and dependencies</td>
</tr>
<tr>
<td>4</td>
<td>Sometimes defined by Systems</td>
</tr>
<tr>
<td>5</td>
<td>Not defined in IP – IP written too long in advance</td>
</tr>
<tr>
<td>6</td>
<td>Not defined in IP – difficult to specify at correct level and size</td>
</tr>
<tr>
<td>7</td>
<td>Not defined in IP – more cooperation between systems and design needed</td>
</tr>
<tr>
<td>8</td>
<td>User Stories in IPs, no problem breaking them down</td>
</tr>
<tr>
<td>9</td>
<td>User Stories in IPs but had to be refined during design</td>
</tr>
</tbody>
</table>

Here we document the experiences of the teams in adopting the practice of User Stories and any issues they had. Most teams did not get the User Stories defined in the Implementation Proposals (IP) as planned in the Agile process. Sometimes this was because the IP had been written before the agile adoption began. Sometimes a “non-agile” area in the organisation might have written the IP. Developers found that initially it was not easy to
define User Stories of the appropriate size that could be implemented in a few days. Sometimes this was as a result of very “flat” system architectures that already existed. Also some IPs were written by “non-Agile” parts of the organisation who were not adopting the User Story concept. The formulation of User Stories improved over time but it took a lot of practice. Almost all teams felt that it took a lot of time, dedicated effort & practise to get definition of User Stories right. Many stated that acceptance criteria should be specified at the same time as the User Story itself and that the interaction between Systems, Design & Test is very important to cover all perspectives.

6.5.8 Stand Up Meetings

Table 41: Stand Up Meetings

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Meetings held, working well</td>
</tr>
<tr>
<td>2</td>
<td>Not held consistently</td>
</tr>
<tr>
<td>3</td>
<td>Meetings held, conscious effort to keep them focussed</td>
</tr>
<tr>
<td>4</td>
<td>Meetings held, conscious effort to keep them focussed</td>
</tr>
<tr>
<td>5</td>
<td>Meetings held, conscious effort to keep them focussed</td>
</tr>
<tr>
<td>6</td>
<td>Meetings held, working well</td>
</tr>
<tr>
<td>7</td>
<td>Meetings held, working well</td>
</tr>
<tr>
<td>8</td>
<td>Meetings held, conscious effort to keep them focussed</td>
</tr>
<tr>
<td>9</td>
<td>Meetings held, conscious effort to keep them focussed</td>
</tr>
</tbody>
</table>

All teams found the Stand up meetings useful and noted the importance of “time-boxing” them. This was a widely adopted practice

6.5.9 Continuous Integration

Table 42: Continuous Integration

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CI working</td>
</tr>
<tr>
<td>2</td>
<td>No CI – problem with build system</td>
</tr>
<tr>
<td>3</td>
<td>Daily/Nightly Build</td>
</tr>
<tr>
<td>4</td>
<td>15 minute build</td>
</tr>
<tr>
<td>5</td>
<td>CI working</td>
</tr>
<tr>
<td>6</td>
<td>CI working</td>
</tr>
<tr>
<td>7</td>
<td>Daily/Nightly Build</td>
</tr>
<tr>
<td>8</td>
<td>Daily/Nightly Build</td>
</tr>
<tr>
<td>9</td>
<td>CI working</td>
</tr>
</tbody>
</table>

The teams were asked about the practice of Continuous Integration. Four out the nine teams had Continuous Integration, with three teams having nightly builds, one team had a build every 15 minutes and one team had difficulty with the automated builds.
6.5.10 Automated Acceptance Testing

Table 43: Automated Acceptance Testing

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Auto Acc tests for CLI, not for GUI</td>
</tr>
<tr>
<td>2</td>
<td>No Auto Acc. Test – problems with GUI tests</td>
</tr>
<tr>
<td>3</td>
<td>Some tests</td>
</tr>
<tr>
<td>4</td>
<td>Starting to implement it</td>
</tr>
<tr>
<td>5</td>
<td>Starting to implement it</td>
</tr>
<tr>
<td>6</td>
<td>Starting to implement it</td>
</tr>
<tr>
<td>7</td>
<td>Auto Acc tests for key features</td>
</tr>
<tr>
<td>8</td>
<td>Auto Acc tests for key features</td>
</tr>
<tr>
<td>9</td>
<td>Auto Acc tests for key features – problems with GUI test</td>
</tr>
</tbody>
</table>

Teams 1, 2 and 9 identified difficulties in testing of GUIs as a problem. Teams 4, 5 and 6 were just starting with the practice and Team 7 had automated testing of key features only and Team 8 had allocated some time to improve their test suite. Teams had achieved various degrees of test automation but all were trying to broaden the test suites and concentrating on the acceptance test to start. Some teams felt that automation was important to allow time for “happy testing”.

6.3.11 Product Owner Role and Demo

Table 44: Product Owner Role

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Systems as Product Owner, informal demo’s</td>
</tr>
<tr>
<td>2</td>
<td>Systems as Product Owner, no demo’s</td>
</tr>
<tr>
<td>3</td>
<td>Systems as Product Owner, one demo held</td>
</tr>
<tr>
<td>4</td>
<td>Product Owner Role unclear</td>
</tr>
<tr>
<td>5</td>
<td>Systems/IP author as Product Owner, demo’s but lack of participation</td>
</tr>
<tr>
<td>6</td>
<td>Systems as Product Owner, demo’s held</td>
</tr>
<tr>
<td>7</td>
<td>Systems as Product Owner, demo’s held</td>
</tr>
<tr>
<td>8</td>
<td>Systems as Product Owner, demo’s held</td>
</tr>
<tr>
<td>9</td>
<td>Systems as Product Owner, demo’s held</td>
</tr>
</tbody>
</table>

Teams did not have any access to the “real customer”, so a Systems Engineer generally took on the role of Product Owner although the extent of role was a little uncertain in some teams. The task of the Product Owner was to maintain a prioritised backlog, participate in Sprint Planning and participate in a product demo at the end of each sprint.
6.3.12 **Retrospectives.**

Table 45: Retrospectives

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Held at end of each Sprint</td>
</tr>
<tr>
<td>2</td>
<td>Held at end of each Sprint, but no value because Sprint goals not met.</td>
</tr>
<tr>
<td>3</td>
<td>Held at end of each Sprint</td>
</tr>
<tr>
<td>4</td>
<td>Held at end of each Sprint</td>
</tr>
<tr>
<td>5</td>
<td>Held at end of each Sprint</td>
</tr>
<tr>
<td>6</td>
<td>Held at end of each Sprint</td>
</tr>
<tr>
<td>7</td>
<td>Held at end of each Sprint</td>
</tr>
<tr>
<td>8</td>
<td>Held at end of each Sprint</td>
</tr>
<tr>
<td>9</td>
<td>Held at end of each Sprint</td>
</tr>
</tbody>
</table>

Here we see if retrospectives were carried out at the end of each Sprint and if the teams had any thoughts on the usefulness of the retrospectives. All teams were having retrospectives at the end of the Sprints and they generally took about 2 hours. Most teams held the demo, followed by the retrospective, which in turn was followed by the planning meeting for the next sprint. The teams that had adopted agile practices to a greater degree found the retrospectives to be of more use.

6.3.13 **The Agile Adoption Strategy.**

Table 46: Adoption Strategy

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Correct strategy, transition gradual enough</td>
</tr>
<tr>
<td>2</td>
<td>Correct strategy in general, but didn’t work for us</td>
</tr>
<tr>
<td>3</td>
<td>Correct strategy, transition fast as process evolving in parallel</td>
</tr>
<tr>
<td>4</td>
<td>Correct strategy, transition gradual enough</td>
</tr>
<tr>
<td>5</td>
<td>Correct strategy, may be compromised by non-Agile stakeholders</td>
</tr>
<tr>
<td>6</td>
<td>Correct strategy, transition gradual enough</td>
</tr>
<tr>
<td>7</td>
<td>Correct strategy, transition gradual enough</td>
</tr>
<tr>
<td>8</td>
<td>Correct strategy, transition gradual enough otherwise it could lose momentum</td>
</tr>
<tr>
<td>9</td>
<td>Correct strategy, transition gradual enough, practices complement each other</td>
</tr>
</tbody>
</table>

Many XP and Scrum practices were introduced together in a single project. All teams strongly agreed that adopting Agile was the correct thing to do, apart from Team 2 who
thought it was “OK”, but it hadn’t worked for them. Teams 6, 7, 8, 9 worked in an area that had tried some XP practices before so from that point of view they felt that they had a gradual transition. Teams 1 and 4, also felt that the transition was gradual enough, and that if the transition was less gradual it might not have gained enough momentum. Team 3 felt that the process was evolving in parallel with the adoption instead of being defined in advance. Teams 5 saw the meeting of the agile and non-agile parts of the organisation as causing some overhead or duplicate work.

All teams found coaching very important & even experienced teams saw benefit in having an experienced, independent view. It is believed to be good to have one experienced person who acts as team “conscience”, questions the way of working & checks that team is following the practices & doing what it agreed.

6.3.14 Continuing with Agile

Table 47: Continue with agile

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Continue</td>
</tr>
<tr>
<td>2</td>
<td>Continue but try it on a smaller project</td>
</tr>
<tr>
<td>3</td>
<td>Continue and refine it</td>
</tr>
<tr>
<td>4</td>
<td>Continue</td>
</tr>
<tr>
<td>5</td>
<td>Continue and refine it</td>
</tr>
<tr>
<td>6</td>
<td>Continue</td>
</tr>
<tr>
<td>7</td>
<td>Continue</td>
</tr>
<tr>
<td>8</td>
<td>Continue</td>
</tr>
<tr>
<td>9</td>
<td>Continue</td>
</tr>
</tbody>
</table>

Everybody wanted to continue working with agile methods and most teams mentioned that there were improvements they would make. Nobody wanted to revert to the tradition process. Team 2 had some reservations, but said they would try it on a different type of project. They didn’t attribute the difficulties in their current project to agile, but the introduction of new technologies and a new process in parallel didn’t help either.

6.3.15 Time or Quality Improvements

Table 48: Time or Quality Improvements

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Quality Improved – Time Similar</td>
</tr>
</tbody>
</table>
The teams were asked to give an intuitive opinion as to whether there were making either time or quality improvements when using agile methods. This was a “gut feeling” without having done any analysis on number of bug reports or man-hours used. All teams (with the exception of Team 2) felt that the quality of the code had improved. Some teams referred to the maintainability aspect of the code as well as the bugs it manifested. Generally teams felt that they weren’t any slower when using agile methods and also felt that the payback on time might be realised later when there are less faults reported by the customers after the product became generally available.

### 6.3.16 Biggest Benefits of Agile.

Table 49: Biggest Benefit of Agile

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Communication, Feedback</td>
</tr>
<tr>
<td>2</td>
<td>Faster Feedback</td>
</tr>
<tr>
<td>3</td>
<td>Faster Feedback</td>
</tr>
<tr>
<td>4</td>
<td>Faster Feedback</td>
</tr>
<tr>
<td>5</td>
<td>Close co-operation</td>
</tr>
<tr>
<td>6</td>
<td>Motivation, Work together</td>
</tr>
<tr>
<td>7</td>
<td>Communication</td>
</tr>
<tr>
<td>8</td>
<td>Work together, Quick Feedback</td>
</tr>
<tr>
<td>9</td>
<td>Short term goals, Feedback</td>
</tr>
</tbody>
</table>

Teams were asked to comment on what they saw as the biggest benefit of using agile methods. The answers here were interesting in that **communication, quick feedback, working as a team** were the predominant answers.
6.3.17 **Negative Aspects of Agile.**

Table 50: Negative Aspect of Agile

<table>
<thead>
<tr>
<th>Team No.</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Interfacing to non-Agile parts of the organisation</td>
</tr>
<tr>
<td>2</td>
<td>Design Analysis phase missing</td>
</tr>
<tr>
<td>3</td>
<td>Difficulty with specifying User Stories</td>
</tr>
<tr>
<td>4</td>
<td>Not doing enough TDD</td>
</tr>
<tr>
<td>5</td>
<td>Role of live tester overlooked in favour of automation</td>
</tr>
<tr>
<td>6</td>
<td>Interfacing to non-Agile parts of the organisation</td>
</tr>
<tr>
<td>7</td>
<td>Today teams are not self managing</td>
</tr>
<tr>
<td>8</td>
<td>Test Planning is difficult</td>
</tr>
<tr>
<td>9</td>
<td>Interfacing to non-Agile parts of the organisation</td>
</tr>
</tbody>
</table>

When asked for negative aspects of agile, the teams did not find anything negative regarding agile methods themselves, but mentioned things that should be improved on in their teams, such as the product architecture, more test driven development or more test automation. Two teams mentioned the boundary with the non-agile part of the organisation. One team felt that there was no Design Analysis phase in agile.

### 6.4 Compare with initial views and new observations

The results here are very similar to Tables 32 and 33 which was the analysis based on the one-to-one interviews. Some differences arise from the fact the some of teams involved in the group interviews were using TDD and pair programming to a greater extent and could comment on its effect.
Table 51: Materialisation of initial concerns - Teams

<table>
<thead>
<tr>
<th>Before Adoption</th>
<th>10 months into the adoption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personality conflicts will be an issue with pair programming</td>
<td>Pair Programming was not implemented widely in all teams, but there was no evidence of personality conflicts</td>
</tr>
<tr>
<td>TDD requires a change of mindset that will take time</td>
<td>Level of TDD varied between teams and some were struggling with the change of mindset</td>
</tr>
<tr>
<td>TDD will be difficult for legacy code and for GUIs</td>
<td>Increasing the amount of TDD with GUI and legacy code was proving problematic</td>
</tr>
<tr>
<td>Too much distraction in an open plan area</td>
<td>Most teams want more co-location and open areas</td>
</tr>
<tr>
<td>Pair programming will be intense and tiring</td>
<td>At least one team mentioned that they limited pair programming to 6 hours per day as a result of it being tiring.</td>
</tr>
<tr>
<td>Time management – getting “pulled off” your agile task and interruptions from phone, e-mails and other meetings.</td>
<td>Not raised as an issue</td>
</tr>
<tr>
<td>More pressure to deliver due to the short iteration</td>
<td>Teams found there was less pressure because they were more confident of the quality of the delivery</td>
</tr>
<tr>
<td>Will there be sufficient support from management</td>
<td>Not raised as an issue. Sometimes the boundary to the non-agile part of the organisation was mentioned.</td>
</tr>
</tbody>
</table>

Table 52: Materialisation of initial expectations - Teams

<table>
<thead>
<tr>
<th>Before Adoption</th>
<th>10 months into adoption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile will reduce the level of documentation as a result of the face-to-face communication</td>
<td>No major change in the level of documentation, but the documents meant more</td>
</tr>
<tr>
<td>Pair programming will help the situation where code inspections are being skipped</td>
<td>This was found to be true</td>
</tr>
<tr>
<td>Pair programming will be good especially for complex work and also for competence building and knowledge transfer</td>
<td>This was found to be true</td>
</tr>
<tr>
<td>TDD has lots of benefits in terms of testing and quality of the code</td>
<td>This was found to be true for the teams that had implemented a lot of TDD</td>
</tr>
</tbody>
</table>

The new observations listed here are drawn from the one-to-one interviews and the group interviews. There were no outstanding differences arising between both sets of interviews based on the individual vs. team factor. However it was easier to reach a consensus in the group interviews.
It was and would not be possible to get User Stories in the IPs 100% of the time. Sometimes the IP could be written by a non-agile part of the organisation by another unit that has an impact on the product.

User Story definition is an activity that takes time and practice. Some initial difficulty was experienced getting them to the correct size.

The System Architecture and System Coupling and dependencies becomes an issue for TDD

New or inexperienced developers adapt more quickly to TDD

Standup meetings were found to be very beneficial and teams needed to concentrate on keeping them focused

In general TDD, CI and Automated Acceptance Testing were practices that took some time to implement

Developers felt that there was some duplication or double work related to the agile and non-agile boundaries in the organisation

Developers were of the opinion that the quality of the code produced had improved

Developers felt that they were taking approximately the same amount of time to deliver code

The biggest benefits of agile were cited as quick feedback, communication, working together, motivation and co-operation

The only disadvantage raised was the all practices were not fully implemented at that stage.

### 6.5 Discussion and Related Work

Agile is not a tool-based technique that can be easily rolled out across a large organisation. It is value based and needs buy-in. Craig Larman points out that “Many confuse the heart of ‘Agile’ with practices rather than values” (Elssamadisy 2007). He also states that “Yet increasingly, we see “top down” mandated or forced adoption of these methods or practices (“You will adopt Scrum”). These are signs of people not understanding the core values and principles of the Agile Manifesto, and instead, focusing on the myriad surface practices that may support agility. This is a grave mistake” (Elssamadisy 2007). In a large organisation there has to be an element of “You will adopt Scrum”. The challenge then is to shift the focus from the surface practices to the core values of agile. In chapters 3 and 4, in the initial phase of the agile adoption process, we note that the surface practices were very much to the fore in the minds of the developers. Pair Programming, Documentation, Test Driven Development, Unit testing and code inspections, Open Plan area all emerged as themes in the analysis of the interviews.

Looking at the changing views of the developers, keep in mind the question “Are they focused on surface practices or values?”
Research exploring aspects of agile development teamwork initiatives associated with positive socio-psychological phenomena exists. Whitworth and Biddle (2007a, 2007b) examine motivational and social aspects of agile teams. A survey by Cockburn and Highsmith (2001), for example, found that agile methodologies were rated higher than other methodologies in terms of morale. While some authors like Cockburn (2002), indicate that scaling of agile methods is difficult, high degrees of motivation and job satisfaction can be experienced in large Scrum teams, as evidenced by Tessem and Maurer (2007). There is evidence of increased job satisfaction in this study because everybody wants to continue with the agile way of working with developers referring to it as “being more fun”. A number of teams noted an improvement in team motivation but the ones, who experienced “success” in implementation of many practises and are conducting effective retrospectives, are the most significant. The general opinion was that it takes training, coaching, time, personal effort (e.g. self learning) to be successful. Several teams asked for a network to facilitate communication, sharing of experiences & help each other. As stated before, the transition does not happen over night. Teams had made varying levels of progress in terms of their adoption of agile methods. One team reverted to traditional methods when they felt under time pressure. When considering how the views of the developers have changed towards agile since the initial interviews and questionnaires, we can conclude that in this study, developers initial concerns and expectations were mainly related to themselves as individuals and are primarily on a superficial level related to the practices. For example, the concerns that the open plan would be distracting, pair programming resulting in personality conflicts, less documentation to do, more pressure due to deadlines. What happened in reality was that these individualistic concerns didn’t really materialise. Take the documentation example; the document that developers initially felt was worthless became a valuable document in the agile world.

After working with agile for a period of time, the focus of the developer’s shifts from individual concerns to principles and values that effect the whole team.

Beck and Anders (2004) outlines the differences between values, principles and practices. We see that the thinking of the developers shifted from a surface level opinion about the practices, and after working for some time in an agile way, the interviews show an emerging understanding of the core values and principles. “XP embraces five values to guide development: communication, simplicity, feedback, courage and respect” (Beck and Anders 2004). We see evidence of the appreciation of communication and quick feedback, mentioned as the biggest benefits of agile. Simplicity is not explicitly mentioned in the interviews, but
the developers see the need for less class dependencies and more focus on architecture with respect to making the unit testing more robust. They realise that the division of user stories makes them “more manageable”. “Sometimes courage manifests as a bias to action. If you know what the problem is, do something about it” ” (Beck and Anders 2004). Developers are taking initiatives, and continuously trying to improve their processes. The retrospectives were working in all teams except for one team. And respect is evident, as the contribution of the whole team working together has been acknowledged.

Therefore it can be seen that in the second stage interviews, the developers talk mainly about values when they were asked about the biggest benefits of agile methods. Their focus has shifted from the practices to values.
7 Agile Methods: The Impact on the code
7.1 Introduction

Software maintainability and software re-usability are key concerns and this chapter asks the question “What impact does the introduction of an agile methodology have on the code?” “The total cost of maintaining a widely used program is typically 40 percent or more of the cost of developing it.” according to Brooks (1975). Other more recent work concurs with this, particularly for large organisations and systems (Coleman et al. 1995). Agile methods are based on a set of principles and practices that value the interactions among people working together to produce high-quality software that creates business value on a frequent basis (Beck et al. 2001). Because agile methods are fundamentally people oriented, we felt that it would not be optimal to analyse code in isolation without considering, the views of the developers, and the level of adoption of agile methods within the teams producing the code.

In this chapter, object-oriented metrics were used to evaluate and characterise the source code of applications developed using agile methods. Applications from three different teams were analysed. We also analysed and compared code that had been previously produced by one of the teams using the traditional methodology. Using the data presented in chapters 5 and 6 from the interviews with the developers we devised an “agile measurement index” based on work by Sidky (2007b). The “agility index” was used to look for correlations between the level of agile practices and the results obtained from the metric based analysis of the code.

7.2 Software Metrics and Data Collection

7.2.1 The Overview Pyramid

Quality metrics for OO Design has been the subject of many studies. With regard to maintenance effort, the Chidamber and Kemerer (1994) CK suite of quality metrics has been used and validated in the literature. Li and Henry (1993) studied the link between CK metrics and the maintenance effort. Basili et al. (1996) found that some of the CK metrics were associated with fault-proneness of classes. The Overview Pyramid introduced by Lanza and Marinescu (2006) is a simple way to characterise a design based on some well-known metrics, such as, the Cyclomatic Number defined by McCabe (1976) and the CK metric Weighted Method Count (WMC) (Chidamber and Kemerer 1994). Because the Overview Pyramid is grounded in proven metrics and also has a visual aspect to it, it seemed to be
suitable for our task of comparing the characteristics of two or more software designs. It uses direct metrics and computed proportions, which are independent of the size of a product, and this makes it very suitable for comparisons. The Overview Pyramid shows key metrics for the source code along with comparisons to industry-standard ranges for those metrics. As shown in Figure 24, the Pyramid uses complexity, coupling and usage of inheritance to evaluate the structure of an object-oriented system.

![Inheritance](image)

**Fig. 24. Overview Pyramid – Structural Aspects (Lanza and Marinescu 2006)**

**Size and Complexity:** The following direct metrics are used for the Size and Complexity part of the pyramid.

- **NOP – Number of Packages**
- **NOC – Number of Classes** (not counting library classes)
- **NOM – Number of Methods**
- **LOC – Lines of Code**
- **CYCLO – Cyclomatic Number**

The direct metrics above are then used to compute the following proportions, which are independent of product size and can easily be used for comparisons.

- **High-level Structuring (NOC/Package)** This provides the high level structuring policy of the system.
- **Class Structuring (NOM/Class)** This reveals how methods are distributed among classes.
- **Method Structuring (LOC/Method)** This is an indication of how well the code is distributed among methods.
- **Intrinsic Method Complexity (CYCLO/Code Lines)** This ratio characterises how much conditional complexity we are to expect in methods (e.g. 0.2 means that a new branch is added every five lines)

Figure 25 shows an example of the size and complexity characterisation from the Overview Pyramid.
**System Coupling**: The following direct metrics are used for the System Coupling part of the pyramid.

- **CALLS – Number of Method Calls**: This metric counts the total number of distinct method calls (invocations) in the code, by summing the number of methods called by all the user-defined methods.

- **FANOUT – Number of Called Classes** (Sum of the FANOUT as defined by Highsmith (2002))

Based on the direct metrics, the following proportions result:

- **Coupling intensity (CALLS/Method)**
- **Coupling dispersion (FANOUT/Method Call)**

Figure 26 shows an example of the system coupling characterisation from the Overview Pyramid.

**Inheritance**: We did not use the Inheritance part of the pyramid in our evaluation.

### 7.2.2 Detection Strategies

Because metrics alone cannot answer all the questions about a system, Lanza and Marinescu (2006) also proposed a number of detection strategies to detect design problems. The detection strategies are based on the work of Fowler et al. (1999), Martin (2002) and Riel (1996). Four design disharmonies that were detected during analysis of the code are described below.

**God Class**

A God Class performs too much work on its own. This has a negative impact on the reusability and the understandability of that part of the system. This design problem is
comparable to Fowler’s “Large Class” bad smell (Fowler et al. 1997). Taking the “God Class” (Riel 1996) design flaw, its properties are class complexity, class cohesion and access of foreign data, therefore Lanza and Marinescu (2006) chose the following metrics to detect a God class.

- **WMC – Weighted Method Count**
- **TCC – Tight Class Cohesion**
- **ATFD – Access to Foreign Data**

Figure 27 shows the graphical representation of the “God class” detection strategy. Lanza and Marinescu derived the limits used for comparison based on data from eighty industrial and open source projects.

![God Class Detection Diagram]

**Brain Method**

A Brain Method tends to centralise the functionality of a class, in the same way as a God Class centralises the functionality of an entire subsystem or system. A Brain Method will be hard to understand and debug and practically impossible to reuse. This design problem is based on three simple code smells (*long methods, excessive branching and many variables used*) described by Fowler et al. (1999). Lanza and Marinescu use the following metrics to detect a Brain Method.

- **LOC – Lines Of Code**
- **CYCLO – Cyclomatic Number**
- **MAXNESTING – Maximum Nesting Level**
- **NOAV – Number Of Accessed Variables**

Figure 28 shows the graphical representation of the “Brain Method” detection strategy.
Brain Class

The primary characteristic of a Brain Class is that it contains a Brain Method. Figure 29 shows the graphical representation of the “Brain Class” detection strategy.

Data Class

A Data Class is a data holder on which other classes strongly rely. Detection of a Data Class is a sign of a design that lacks encapsulation and has poor data-functionality proximity. The **WOC – Weight Of Class** metric is used in the detection strategy. Figure 30 shows the graphical representation of the “Data Class” detection strategy.
7.3 Data Collection

iPlasma (integrated PlAtform for Software Modelling and Analysis) is a software tool used to generate the overview pyramid, class blueprints and detect design flaws (LOOSE Research Group; Marinescu et al. 2005). Source code from the case companies VOB (Version Object Base) was made available.

7.4 Levels of Agility

7.4.1 Agility Measurement Index

Research by Sidky (2007b) includes a value-based agile measurement index, known as the Sidky Agile Measurement Index (SAMI), as part of a process framework for the adoption of agile practices. Sidky’s research was motivated by the lack of structured approaches in the public domain to guide agile adoption efforts. According to Sidky (2007b), “Organisations aspiring to become agile want to know when they are considered “agile,” as well as what it means to be “agile”. An example of this is found in the case study in the following quotation from one of the teams in the 2nd interviews. “I don’t think there’s a clear line in the sand and if you are one side agile – I think if you are practising – improving communication with stand-up meetings and working on implementation from the start – I think that is agile – its part of agile – that was one of the things we could have handled differently. Instead of having a clear definition – if you are all of these steps you are Agile so” [119]. The team was referring to the fact that the organisation had a red light/green light type evaluation of whether a team was “agile” or not. So the team might be labelled as “non-agile” even though they had adopted a certain number of agile practices, but not quite as many as defined by the organisation. Lack of clarity about what agile is was evident in the following statements “They were only taking the bits they wanted from it, so if they do that I am not sure if some of
it can work” and also “I am not really 100% sure of how or are we "bastardising" what agile is meant to be to.” In his proposal Sidky (2007b) addresses four main issues:

- Introducing structure in a complex and unpredictable process like that of agile adoption
- Measurement and assessment of agility independent of a particular agile methods
- Accommodating project and organisational characteristics influencing agile adoption efforts
- Ensuring that the framework guides the adoption effort in an efficient and effective manner.

Sidky (2007b) defines five levels of agility as shown in Table 54 below. Each of the agile levels is made up of a set of agile practices, which will help accomplish the level’s objective.

Table 54: Agile Levels defined by Skidy (Sidky 2007b)

<table>
<thead>
<tr>
<th>Agile Level</th>
<th>Level Name</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Encompassing</td>
<td>Establishing a vibrant and all-encompassing environment to sustain agility</td>
</tr>
<tr>
<td>4</td>
<td>Adaptive</td>
<td>Responding to change through multiple levels of feedback</td>
</tr>
<tr>
<td>3</td>
<td>Effective</td>
<td>Developing high quality, working software in an efficient an effective manner</td>
</tr>
<tr>
<td>2</td>
<td>Evolutionary</td>
<td>Delivering software early and continuously</td>
</tr>
<tr>
<td>1</td>
<td>Collaborative</td>
<td>Enhancing communication and collaboration</td>
</tr>
</tbody>
</table>

The main component of Sidky’s Agile Adoption Framework (Sidky 2007b) is a 4-Stage process. The first stage of the process helps determine whether organisations are ready to undergo agile adoption efforts. The second and third stages provide a means for projects and organisations to assess their agile potential using the Sidky Agile Measurement Index (SAMI). The last stage, Stage 4, suggests a final set of Agile Practices for organisations to adopt by reconciling any differences between the Agile Levels identified in Stage 2 and Stage 3. The SAMI is the measurement index used to identify the highest level of agility a project can reach as part of Stage, to identify the level of agility the organisation is ready to adopt (Stage 3), and in reconciling any existing differences (Stage 4). Sidky argues that his approach is different in that it identifies the agile potential of a project. The SAMI measurement is based on Agile Practices. Each Agile level consists of a set of different agile
practices. Table 55 (Sidky 2007b) shows how agile practices under each agile principle are identified to populate the table.

Table 55: SAMI populated with principles and concepts (Sidky 2007b)

<table>
<thead>
<tr>
<th>Agile Principles</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Embrace</td>
<td>Plan and</td>
<td>Human-centric</td>
<td>Technical</td>
<td>Customer</td>
</tr>
<tr>
<td>Change to Deliver</td>
<td>Deliver Software</td>
<td></td>
<td>Excellence</td>
<td>Collaboration</td>
</tr>
<tr>
<td>Customer Value</td>
<td>Frequently</td>
<td>Ideal agile</td>
<td>TDD</td>
<td>Frequent face-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>physical setup</td>
<td>Pair Programming</td>
<td>to-face</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>interaction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>between</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>developers &amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>users (collated)</td>
</tr>
<tr>
<td>Level 5</td>
<td>Level 4</td>
<td>Level 3</td>
<td>Level 2</td>
<td>Level 1</td>
</tr>
<tr>
<td>Encompassing</td>
<td>Adaptive</td>
<td>Effective</td>
<td>Evolutionary</td>
<td>Collaborative</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>requirements</td>
<td>Reflect and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Continuous</td>
<td>tune process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Delivery</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Planning at</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>different levels</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Level 5 |  |  |  |  |
| Encompassing |  |  |  |  |
| Low process ceremony | Agile project estimation | Ideal agile physical setup | TDD |
|                  |                  |                  |                  |
|                  |                  |                  |                  |
| Level 4 | Adaptive |  |  |  |
| Client driven iterations | Smaller and more frequent releases (4-8 weeks) | Adaptive planning | Daily progress tracking meetings |
| Continuous customer satisfaction feedback |                  |                  | Agile documentation |
|                  |                  |                  | User stories |
| Level 3 | Effective |  |  |  |
| Risk driven iterations | Self-organizing teams | Continuous Integration | Continuous improvement (refactoring) |
| Plan features not tasks |                  |                  | Unit tests |
| Maintain a list of all features and their status (backlog) |                  |                  |                  |
| Level 2 | Evolutionary |  |  |  |
| Evolutionary requirements | Continuous Delivery | Software configuration management | Customer contract reflective of evolutionary |
| Planning at different levels |                  |                  |                  |
| Level 1 | Collaborative |  |  |  |
| Reflect and tune process | Collaborative Planning | Collaborative Teams | Customer commitment to work with developing team |
|                  |                  | Empowered and motivated teams |                 |
|                  |                  | Knowledge sharing tools |                 |
|                  |                  | Task volunteering |                 |
|                  |                  |                  |                 |
The goal was to get an agility measurement for the teams with respect to each other and not on a global cross-company level, so an adaptation of Sidky’ SAMI table was used. This adaptation took into account that the teams in this study had no access to a real customer so the practices and concepts related to customer collaboration were not included in the adapted SAMI table. Other issues that hadn’t been assessed during the interviews also had to be omitted. This is acceptable since our objective is to compare the teams to each other. Table 56 shows the adapted version of the SAMI table. The letter in brackets after each practice or concept is used for reference in relation to 59.

Table 56: SAMI table adapted for this study (adapted from Sidky 2007b)

<table>
<thead>
<tr>
<th>Agile Principles</th>
<th>Embrace Change to Deliver Customer Value</th>
<th>Plan and Deliver Software Frequently</th>
<th>Human-centric Excellence</th>
<th>Technical Excellence</th>
<th>Customer Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 5</td>
<td>Encompassing</td>
<td>Ideal agile physical setup (h)</td>
<td>TDD (i)</td>
<td>Pair Programming(j)</td>
<td></td>
</tr>
<tr>
<td>Level 4</td>
<td>Adaptive</td>
<td>Daily progress Tracking meetings f)</td>
<td>User stories(g)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 3</td>
<td>Effective</td>
<td>Maintain a list of all features and their status(backlog) (c)</td>
<td>Frequent face-to-face communication (d)</td>
<td>Continuous Integration (e)</td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td>Evolutionary</td>
<td></td>
<td>Software configuration management (b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1</td>
<td>Collaborative</td>
<td>Collaborative Teams (a)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sidky (2007b) suggests that a range result is determined for the characteristic being assessed and that range result is mapped to a nominal value as shown below in Table 57. The ranges shown can be modified by any organisation using this method depending on the goals of the organisation.
Table 57: Nominal Values (Sidky 2007b)

<table>
<thead>
<tr>
<th>Nominal Value</th>
<th>Result</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Achieved</td>
<td>0%-35%</td>
<td>0</td>
</tr>
<tr>
<td>Partially Achieved</td>
<td>35%-65%</td>
<td>1</td>
</tr>
<tr>
<td>Largely Achieved</td>
<td>65%-85%</td>
<td>2</td>
</tr>
<tr>
<td>Fully Achieved</td>
<td>85%-100%</td>
<td>3</td>
</tr>
</tbody>
</table>

7.4.2 Agility Assessment

**Collaborative teams**

Table 44 (Product Owner Role) and Table 40 (User Stories) were used to obtain the scores shown below in Table 58. The Product Owner Role involves working with Systems.

Table 58: Collaborative teams

<table>
<thead>
<tr>
<th>Status</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Owner Role unclear</td>
<td>0</td>
</tr>
<tr>
<td>Systems as Product Owner, but no or informal demo</td>
<td>1</td>
</tr>
<tr>
<td>Systems as Product Owner, demo’s held</td>
<td>2</td>
</tr>
<tr>
<td>Systems as Product Owner, demo’s held and User Stories coming from systems in IP</td>
<td>3</td>
</tr>
</tbody>
</table>

**Software Configuration Management**

All teams were using ClearCase and at code level was under configuration management. The broader change request handling mechanism was not taken into account.

**Maintain a List of features and their status (backlog)**

All teams except team 4 were maintaining a backlog.

**Frequent face-to-face communication**

The data on stand up meetings from Table 41 was used here.

Table 59: Frequent face-to-face communication

<table>
<thead>
<tr>
<th>Status</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not held</td>
<td>0</td>
</tr>
<tr>
<td>Not held consistently</td>
<td>1</td>
</tr>
<tr>
<td>Meetings held, conscious effort to keep them focussed</td>
<td>2</td>
</tr>
<tr>
<td>Meetings held, working well</td>
<td>3</td>
</tr>
</tbody>
</table>
**Continuous Integration**

Data from Table 42, (Continuous Integration) was used.

Table 60: Continuous Integration

<table>
<thead>
<tr>
<th>Status</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>No CI</td>
<td>0</td>
</tr>
<tr>
<td>Daily/Nightly Build</td>
<td>1</td>
</tr>
<tr>
<td>15 minute build</td>
<td>2</td>
</tr>
<tr>
<td>CI working</td>
<td>3</td>
</tr>
</tbody>
</table>

**Daily progress tracking meetings**

The data from the stand up meetings, Table 41 was used here.

Table 61: Daily progress tracking meetings

<table>
<thead>
<tr>
<th>Status</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not held</td>
<td>0</td>
</tr>
<tr>
<td>Not held consistently</td>
<td>1</td>
</tr>
<tr>
<td>Meetings held, conscious effort to keep them focussed</td>
<td>2</td>
</tr>
<tr>
<td>Meetings held, working well</td>
<td>3</td>
</tr>
</tbody>
</table>

**User Stories**

Table 62: User Stories

<table>
<thead>
<tr>
<th>Status</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not defined by Systems</td>
<td>0</td>
</tr>
<tr>
<td>Sometimes defined by Systems</td>
<td>1</td>
</tr>
<tr>
<td>Refined during design</td>
<td>2</td>
</tr>
<tr>
<td>Specified at correct level</td>
<td>3</td>
</tr>
</tbody>
</table>

**Ideal Agile Physical Setup**

Table 63: User Stories

<table>
<thead>
<tr>
<th>Status</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systems, Design Test not co-located</td>
<td>0</td>
</tr>
<tr>
<td>Design, Test co-located</td>
<td>1</td>
</tr>
<tr>
<td>Systems, Design, Test co-located</td>
<td>2</td>
</tr>
<tr>
<td>System, Design, Test co-located with open area</td>
<td>3</td>
</tr>
</tbody>
</table>
**TDD**

Values based on Table 38 with limits as suggested by Sidky (2007b).

Table 64: TDD

<table>
<thead>
<tr>
<th>Status</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not carried out or &lt;35%</td>
<td>0</td>
</tr>
<tr>
<td>Carried out 35%-65%</td>
<td>1</td>
</tr>
<tr>
<td>Carried out 65%-85%</td>
<td>2</td>
</tr>
<tr>
<td>Carried out 85%-100%</td>
<td>3</td>
</tr>
</tbody>
</table>

**Pair Programming**

Values based on Table 37 with limits as suggested by Sidky (2007b).

Table 65: Pair Programming

<table>
<thead>
<tr>
<th>Status</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not carried out or &lt;35%</td>
<td>0</td>
</tr>
<tr>
<td>Carried out 35%-65%</td>
<td>1</td>
</tr>
<tr>
<td>Carried out 65%-85%</td>
<td>2</td>
</tr>
<tr>
<td>Carried out 85%-100%</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 66: Agility Assessment per Team

<table>
<thead>
<tr>
<th>Team</th>
<th>Practise</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
<th>#7</th>
<th>#8</th>
<th>#9</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td></td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td></td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>f</td>
<td></td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>g</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>h</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>i</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>h</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>17</td>
<td>9</td>
<td>12</td>
<td>11</td>
<td>20</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 66 shows that there are varying degrees of implementation of the agile practices and concepts across the nine teams in the organisation.
7.5 Analysing the code

Code from a number of teams was analysed using iPlasma. In deciding what code or applications to analyse, the fact that many teams were working with legacy code was a factor. The amount of legacy code, the number of times it had been updated and other project specific issues would influence results. To minimise the effect of the legacy on the metrics, it was decided to concentrate the analysis on new code. One legacy application was considered.

Three separate cases are considered:

- Team 9 were a highly motivated agile team that scored a high agility index. Code for a new feature developed by team 9 using agile methods was compared with the code for a similar feature developed by the same team using traditional methodology. Three to four of the team were involved in both the development using traditional and the development using agile method.
- Team 3 developed two new applications. These applications are analysed and compared with the outcome from team 9
- The code for Team 1, which had a lot of legacy code was analysed to compared the metrics with the limits defined by iPlasma.

7.5.1 Case One – Traditional vs. Agile Comparison

This section contain a metrics based comparison of a software module developed by a team using traditional methods with the metrics for a software module that contains similar functionality and was developed by the same team using an agile methodology. The results obtained from the source code produced using agile methods are compared with the results for source code produced for a similar type of application by the same team using a traditional methodology. The contrast is stark. In this case agile methods guided the developers to produce code that manifests better quality and maintainability characteristics.

Size and Complexity

Table 67: Size and Complexity – Direct Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Traditional Methods</th>
<th>Agile Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>NOC</td>
<td>10</td>
<td>42</td>
</tr>
<tr>
<td>NOM</td>
<td>233</td>
<td>425</td>
</tr>
<tr>
<td>LOC</td>
<td>8516</td>
<td>3495</td>
</tr>
<tr>
<td>CYCLO</td>
<td>803</td>
<td>593</td>
</tr>
</tbody>
</table>
The direct metrics listed in Table 67 show that when the agile methods were employed, the Cyclomatic number has decreased from 803 to 593. The number of lines of code has almost halved from 8516 to 3495 and the total number of classes increased from 10 to 42.

Table 68 shows the computed proportions. Unlike the direct metrics, which are absolute values, the computed proportions allow easy comparison of projects independent of their size. The L, A, H signify if the value is Low, Average or High respectively according to the thresholds used in iPlasma. The thresholds set in iPlasma are based on an analysis of 45 Java and 37 C++ open source and commercial systems.

CYCLO/LOC (Cyclomatic Number /Lines Of Code) characterises how much conditional complexity exists in the code. The value increased when using agile methods, but it is still lower than average and its increase is attributed to the large decrease in Lines Of Code. The Lines of Code is less than half when agile methods are used. Later we will see an example of how duplicate lines of code have been removed.

The LOC/NOM (Lines of Code/Number of Methods) is an indication of how well the code is distributed among methods and this value has decreased from what was classed as a very high value to an average value. NOM/NOC (Number of Methods/Number of classes) reflects the quality of class design because it shows how well methods are distributed among classes. Again there is a big improvement here where the value has decreased from 23 to 10.11. NOM/NOC is still higher than the average threshold. The average threshold is 7. NOC/NOP (number of Classes/Number of Packages) did not vary much and it is below the average value, which is 17.

**System Coupling**

Table 69: System Coupling – Direct Metrics
Table 70: System Coupling – Computed Proportions

<table>
<thead>
<tr>
<th>Metric</th>
<th>Traditional Methods</th>
<th>Agile Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALLS/NOM</td>
<td>4.96 (H)</td>
<td>2.6 (A)</td>
</tr>
<tr>
<td>FANOUT/CALLS</td>
<td>0.59 (L)</td>
<td>0.54 (L)</td>
</tr>
</tbody>
</table>

Table 69 contains the direct metrics for the System Coupling characteristics. The computed metrics contained in Table 70 better characterise the coupling of the system. CALLS/NOM (Number of calls/Number of Methods) decreased from a value of 4.96 (classified as high) to 2.6, which is in the average band. CALLS/NOM (coupling intensity) denotes on average how many other methods are called from each method. Very high values would indicate excessive coupling. The FANOUT/CALLS (coupling dispersion) value was low to start with and didn’t change significantly. The FANOUT/CALLS is an indication of how much the coupling involves many classes (A value of 0.5 means that every second method call involve another class).

**Design Flaws**

Table 71: Design Flaws Detected

<table>
<thead>
<tr>
<th>Design Flaw</th>
<th>Traditional Methods</th>
<th>Agile Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brain Class</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Brain Method</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td>God Class</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Data class</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 71 shows that the number of design flaws detected by iPlasma reduced dramatically in the code that was designed using an agile methodology.

**Re-structuring of a God class**

We now look at a class that was detected as a God class (called *AdjustSoftwareTask*) in the code designed using traditional methods. In the code designed using agile methods there is a class (*AdjustTask*) that has a similar purpose, but it is no longer classified by iPlasma as a God class. This code is part of a Network Management Application and in general terms both classes would:

- Communicate with a Network Element (NE)
- Read an inventory of software from the NE in XML format
- Parse the XML information that is returned from the NE
- Store the information returned from the NE in a database
The classes also have to handle any error conditions that occur (for example an error when connecting to the database). The basic difference between the two classes is that they deal with different types of Network Elements. On examining the classes we saw that in the AdjustSoftwareTask class, the class was handling everything to do with all the points listed above by itself, but in the package structure containing the AdjustTask class, new packages had been added to the structure. The new packages handled the connection to the database, error cases when connecting, writing to the database and the xml parsing. This meant that AdjustTask was more of a coordinator and it was much simpler. There was less coupling and a greater possibility that code could be re-used in the future. If for example the format of the XML file changed in the future, then the change would be easier to accommodate because the XML parsing was placed in a separate package.

First iPlasma was run on the individual classes AdjustSoftwareTask and AdjustTask and the results analysed.

Table 72: Size and Complexity – Direct Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>AdjustSoftwareTask (Trad.)</th>
<th>AdjustTask (Agile)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOC</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NOM</td>
<td>54</td>
<td>15</td>
</tr>
<tr>
<td>LOC</td>
<td>1095</td>
<td>188</td>
</tr>
<tr>
<td>CYCLO</td>
<td>146</td>
<td>54</td>
</tr>
</tbody>
</table>

Table 72 shows that the Cyclomatic number has significantly decreased from 146 to 54. The amount of code in the class decreased from 1095 lines to 188 lines.

Table 73: Size and Complexity – Computed Proportions

<table>
<thead>
<tr>
<th>Metric</th>
<th>AdjustSoftwareTask (Trad.)</th>
<th>AdjustTask (Agile)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOM/NOC</td>
<td>54 (H)</td>
<td>15 (H)</td>
</tr>
<tr>
<td>LOC/NOM</td>
<td>20.27 (H)</td>
<td>12.53 (H)</td>
</tr>
<tr>
<td>CYCLO/LOC</td>
<td>0.13 (L)</td>
<td>0.12 (L)</td>
</tr>
</tbody>
</table>

From Table 73 we can see that while the cyclomatic number per line of code hasn’t changed that much, the lines of code per method has decreased even though AdjustTask still has a value that is slightly higher than the average value defined by Lanza and Marinescu (2006).
Table 74: Design Flaws Detected

<table>
<thead>
<tr>
<th>Design Flaw</th>
<th>AdjustSoftwareTask (Trad.)</th>
<th>AdjustTask (Agile)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brain Method</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>God Class</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Data Class</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 74 shows that the class developed using the agile methodology has no design flaws detected by iPlasma whereas the class developed using traditional methods had ten design flaws. AdjustSoftwareTask, which was developed using traditional methods has a number of nested classes, which caused some of the problems. When traditional methods were used, the God classes detected were AdjustSoftwareTask itself and its nested class called XmlSoftwareParser. The separation of the XML parsing functionality into another package has eliminated this problem for AdjustTask. Using traditional methods, the Data Classes were again nested classes called XmlNodeItem, XmlSoftwareModel, XmlSwItem and XmlUpItem. Again these classes were used to process the XML data and certainly the choice to move the processing of the XML data to a separate package was the correct thing to do.

Another observation made from studying the code was that a tendency to unnecessarily duplicate code no longer existed in the code produced using agile methods. This fact is supported in the metrics because the Lines of Code value decreased from 1095 to 188. Here we see a code example from the AdjustSoftwareTask (traditional methods) where there is code duplicated unnecessarily.

```java
catch (final CppCsNodeNotConnectedException e) {
    failed = true;
    LOGGER.warning("Exception caught: " + SmUtil.getMessage(e))
    failureMessage = SmUtil.getMessage(e);
    reportMessage(SmUtil.getMessage(e));
    retryableReported = true;
    operationFinished(false);
    SM_LOGGER.error(SmErrorTypes.EXCEPTION,
    SmSeverityEnum.MINOR, "USER=" + userId + ";
    NE=" + neName+ "; MSG=startOperation(): Exception caught:" + SmUtil.getMessage(e));}

catch (final CppCsException e) {
    failed = true;
    LOGGER.warning("Exception caught: " + SmUtil.getMessage(e));
    failureMessage = SmUtil.getMessage(e);
    reportMessage(e);
    operationFinished(false);
    SM_LOGGER.error(SmErrorTypes.EXCEPTION,
    SmSeverityEnum.MINOR, "USER=" + userId + ";
    NE=" + neName + "; MSG=startOperation():Exception caught:" + SmUtil.getMessage(e));
```
A similar case in the code produced using agile methods is shown below:

```java
....

private void handleException(final String message, final Exception exception){
    failed = true;
    logger.warning("Exception caught: " + SmUtil.getMessage(exception));
    failureMessage = message + " : " + SmUtil.getMessage(exception);
    reportMessage(message, exception);
    operationFinished(false);
    smLogger.error(SmErrorTypes.EXCEPTION, SmSeverityEnum.MAJOR, "USER=" + userId + ";
    NE=" + neName + ";
    MSG=startOperation(): Exception caught:"
    + SmUtil.getMessage(exception));}

PMD (SourceForge.net 2012) is a software application that scans Java source code and looks for potential problems such as dead code, duplicate code and possible bugs such as empty try/catch statements. The AdjustTaskSoftware (traditional methods) had 17 PMD violations and the AdjustTask (agile methods) had 6 PMD violations. Due to the difference in the size of the two classes the results are similar.

<table>
<thead>
<tr>
<th>Violation Type</th>
<th>AdjustTaskSoftware (trad)</th>
<th>AdjustTask (agile)</th>
</tr>
</thead>
<tbody>
<tr>
<td>More than one return in a method</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>Empty catch block</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Avoid Duplicate Literals</td>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>

CheckStyle (SourceForge.net 2011) is another IDE plugin that can checks the Java source code against a set of coding standards. In terms of the types of non-adherences found both pieces of code had similar results. A common non-adherence in both classes was the placing of curly brackets e.g.

```java
public void start()
{
    ThreadPoolFactory.getThreadPool().add(startRequest);
}
```
instead of

```java
public void start(){
    ThreadPoolFactory.getThreadPool().add(startRequest);
}
```

In terms of coding standards the only noticeable difference was that the code produced using traditional methods consistently broke the 80 characters per line rule whereas the code produced using agile methods did not.

**Team Factors**

The code produced with an agile methodology and analysed using iPlasma was significantly better than similar code produced by the same team using more traditional methods.

As part of the interviews that were held 10 months into the adoption process (already documented in chapter 6), the team (Team 9) that developed the code was interviewed. A systems engineer, two developers and a tester represented the team. The interview transcript was analysed looking for factors such as developer attitudes, agile practices, level of agile adoption to which the findings could possibility be attributed and to put the findings in context. The factors listed below are felt to be relevant. A small number of example quotations in each section to give a better perspective. The time of quotation during the interview is used as a reference. For example [00:09:36] means 9 minutes and 36 seconds into the interview.

1 Overall the team was new to agile methods, but they had one developer that has previous experience with agile methods. The team felt that this was very helpful. “We had Matthias, yeah we had quite good I mean Matthias was very experienced from XP practices I think” [00:15:23]

2 The team members very motivated and eager to learn about agile methods. “For me I browsed a few books, read on the Internet and practised some and looked at some more books …” [00:20:41] “It is very motivating to have people around you that are interested – everyone is interested in what they are doing” [01:12:30]

3 Pair programming was happening quite naturally within the team and to a large extent. “I think everybody worked with everybody. In the beginning we had quite static pairs but then we tried to change each day.” [00:24:11] “We tried to do everything in pairs.” [00:24:37]
The team implemented test driven development. They felt that they mastered test driven development. They had a high level of code coverage in unit test. Having a person with previous experience in XP practices and test driven development, in particular, was seen to be beneficial and the team saw the benefits of test driven development. “For unit tests we had quite high coverage so... approximately 95% of the code we wrote” [00:30:27] “That is, in general, or whatever tip I can give is to have one good really experienced guy in the team that has worked with TDD before. Otherwise you will take short cuts probably.” [00:37:00] “I think it was one of the crucial parts of our success actually for this project really, TDD, this way of working” [00:38:08] “And I know from those other guys that have not worked with TDD before, now they see the benefit of it” [00:38:37]

The number of bug reports received on the code was very low and the team felt that the test driven development had a positive impact on quality. “The initial design might have been slower but I think we got that back when we didn’t have to fix a lot of TRs (bug reports)” [01:08:36]. But I mean quality look at I mean we delivered over 2 thousand IP (Implementation Proposal) hours and we have 3 or 4 TRs (bug reports) and 3 of them are not even at code level”[01:12:53]

The team were unanimous in their endorsement of agile methods and were keen to continue in that way of working and continuously improve their practices. When we look at the “agility index” for Team 9, we can see that they were rated at a value of 25. This was the highest score of any of the teams that took part in the study.

**Discussion and Related Work**

Studies exist that propose different metric suites to assess an Object Oriented Design. Sato et al. (2007), analyse and evaluate OO metrics from seven projects with different approaches of agile adoption. Their findings showed that a project with less agile practices presented higher complexity and coupling measures. Concas et al. (2008) have similar findings, where an improvement in quality metrics was correlated to the use of agile practices by skilled developers. Moser et al. (2007) proposed a method for assessing the evolution of maintainability during software development called the Maintainability Trend (MT). However they could not conclude “in absolute terms” that XP results in more maintainable code since the metrics were not run on code produced using more traditional methodologies.

Marchenko and Abrahamsson (2007) have investigated the use of static analysis tools to predict software defects.
A number of studies focus on the practice of Test Driven Development (TDD) and more over its effect on defect density. Nagappan et al. (2008) report on four products where the defect density decreased between 40% and 90% relative to similar projects that did not use TDD. Maximillien and Williams (2003) found that the application of TDD reduced the defect rate by 50%.

The comparison shows that the code analysed using iPlasma was significantly better than code produced by the same team using more traditional methods. There have been improvements in terms of:

- Overall Complexity
- High Level Structuring
- Method Structuring
- Number of Design Flaws

The team studied was overall an inexperienced agile team, but they had at least one developer with previous experience of XP practices. Out of the nine teams interviewed, this team referred to as Team #9, appeared to be the most successful. Based on the “agility index” they scored the highest of the nine teams. The team had a very positive experience in the adoption of agile methods. For a team that was new to agile methods, they adopted pair programming and test driven development to a high degree. As well as having improved the code structure, only a few bugs were found during the test phases. So the improvement evident in the metrics based static analysis of the code did actually manifest itself as a real quality improvement in terms of less bug reports. The team themselves felt that the test driven development played a large part in the quality improvement experienced. They felt that the test driven development was critical to their success.

Our results concur with the findings of Sato et al. (2007) and Concas et al. (2008). Our research also shows that in this case the improvements in metrics resulted in a tangible result in terms of much fewer bugs being reported, as stated by the team during the interview.

7.5.2 Case Two - Agile vs. Agile Comparison

This section contains a metrics based analysis of 2 software applications (Application #1 and Application #2) developed by Team 3 using agile methods. The applications were completely new, so there was no legacy code involved. The results achieved were compared to the figures obtained from Team 9 when using agile methods. The results from Team 9 were selected as a baseline or a comparison because the code from Team 9 showed a
significant improvement over the code they previously produced using traditional methods. The analysed code produced by team 9 was also completely new code i.e. there was no design base or legacy code used. The notation App#1 and App#2 is used below to denote Application#1 and Application#2 in the table headings. In the computed proportions shown in Table 60, the High, Average and Low indications as calculated by iPlasma are specified.

**Size and Complexity**

Table 76: Size and Complexity – Direct Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>App#1 (Team 3)</th>
<th>App#2 (Team 3)</th>
<th>Team 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>14</td>
<td>22</td>
<td>4</td>
</tr>
<tr>
<td>NOC</td>
<td>107</td>
<td>119</td>
<td>42</td>
</tr>
<tr>
<td>NOM</td>
<td>812</td>
<td>1972</td>
<td>425</td>
</tr>
<tr>
<td>LOC</td>
<td>10658</td>
<td>19158</td>
<td>3495</td>
</tr>
<tr>
<td>CYCLO</td>
<td>1700</td>
<td>2696</td>
<td>593</td>
</tr>
</tbody>
</table>

Table 77: Size and Complexity – Computed Proportions

<table>
<thead>
<tr>
<th>Metric</th>
<th>App#1 (Team 3)</th>
<th>App#2 (Team 3)</th>
<th>App#3 (Team 9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOC/NOP</td>
<td>7.64(L)</td>
<td>5.40(L)</td>
<td>10.5(L)</td>
</tr>
<tr>
<td>NOM/NOC</td>
<td>7.58(A)</td>
<td>16.57(H)</td>
<td>10.11(H)</td>
</tr>
<tr>
<td>LOC/NOM</td>
<td>13.12(H)</td>
<td>9.71(A)</td>
<td>8.45(A)</td>
</tr>
<tr>
<td>CYCLO/LOC</td>
<td>0.15(H)</td>
<td>0.14(A)</td>
<td>0.16(L)</td>
</tr>
</tbody>
</table>

The direct metrics listed in Table 76 show that Application#1 and Application#2 are significantly bigger in terms of lines of code than the module (Application#3) they are being compared with.

Table 77 shows the computed proportions. Unlike the direct metrics, which are absolute values, the computed proportions allow easy comparison of projects independent of their size. The L, A, H signify if the value is Low, Average or High respectively according to the thresholds used in iPlasma. CYCLO/LOC characterises how much conditional complexity exists in the code. The value is similar for all three cases. The LOC/NOM is an indication of how well the code is distributed among methods and this value is high for Application#1. NOM/NOC reflects the quality of class design because it shows how methods are distributed among classes. This time Application#2 has a high value. NOM/NOC for Application#2 is still higher than the average threshold, which is 7. NOC/NOP did not vary much and is still below the average value of 17, in all three cases.
**System Coupling**

Table 78: System Coupling – Direct Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>App#1 (Team 3)</th>
<th>App#2 (Team 3)</th>
<th>App#3(Team 9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALLS</td>
<td>2499</td>
<td>5446</td>
<td>1109</td>
</tr>
<tr>
<td>FANOUT</td>
<td>1618</td>
<td>3680</td>
<td>606</td>
</tr>
</tbody>
</table>

Table 79: System Coupling – Computed Proportions

<table>
<thead>
<tr>
<th>Metric</th>
<th>App#1 (Team 3)</th>
<th>App#2 (Team 3)</th>
<th>App#3(Team 9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALLS/NOM</td>
<td>3.07(H)</td>
<td>2.67(A)</td>
<td>2.6(A)</td>
</tr>
<tr>
<td>FANOUT/CALLS</td>
<td>0.64(A)</td>
<td>0.67(H)</td>
<td>0.54(L)</td>
</tr>
</tbody>
</table>

Table 78 contains the direct metrics for the System Coupling characteristic. The computed metrics contained in Table 79 better characterises the coupling of the system. CALLS/NOM is just inside the high band for Application#1 and in the average range for Application#2. CALLS/NOM denotes on average how many other methods are called from each method. Very high values would indicate excessive coupling. The FANOUT/CALLS is an indication of how much the coupling involves many classes (A value of 0.5 means that every second method call involve another class). FANOUT/CALLS is similar for all three applications with Application #2 just reaching the High threshold.

**Design Flaws**

Table 80: Design Flaws Detected

<table>
<thead>
<tr>
<th>Design Flaw</th>
<th>App#1 (Team 3)</th>
<th>App#2 (Team 3)</th>
<th>App#3(Team 9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brain Class</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Brain Method</td>
<td>22</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>God Class</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Data Class</td>
<td>11</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 80 shows the number of design flaws detected by iPlasma.

**Summary**

Overall in terms of the complexity measurement, the value is similar for the two applications produced by team 3 and the application produced by team 9. With regard to low level structuring (Lines of code per method), application #1 has a higher value than application#2 or the code from team 9. For the high level structuring application#2 has a higher value than application#1 or application#3. Again for the system coupling figures, it can be seen that application#3 (team 9) has better figures.
When it comes to design flaws and even taking into account the size of the applications, it is easy to see that application#1 has a far higher number of design flaws than application#3. The overall assessment is that application#3 shows better characteristics than application#2 or application#1 when analysed by iPlasma. Looking at the agility index determined earlier in this chapter, team 3 realised a value of 12 while team 9 had a much higher value of 25. Therefore in this case we can say that a possible correlation exists between the level of agile practices and the code characteristics in this case.

### 7.5.3 Case Three - Agile with Legacy Code

This section contains a metric based analysis of code produced by team 1 using agile methods. One important thing to note is that the application developed by team 1 was a large application with over 70K lines of code and much of that code was legacy code that would have been developed using traditional methods.

#### Size and Complexity

Table 81: Size and Complexity – Direct Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Team 1 (agile)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>31</td>
</tr>
<tr>
<td>NOC</td>
<td>71</td>
</tr>
<tr>
<td>NOM</td>
<td>4203</td>
</tr>
<tr>
<td>LOC</td>
<td>70446</td>
</tr>
<tr>
<td>CYCLO</td>
<td>13402</td>
</tr>
</tbody>
</table>

Table 82: Size and Complexity – Computed Proportions

<table>
<thead>
<tr>
<th>Metric</th>
<th>Team 1 (agile)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOC/NOP</td>
<td>2.21(L)</td>
</tr>
<tr>
<td>NOM/NOC</td>
<td>59.19(H)</td>
</tr>
<tr>
<td>LOC/NOM</td>
<td>16.76(H)</td>
</tr>
<tr>
<td>CYCLO/LOC</td>
<td>0.17(L)</td>
</tr>
</tbody>
</table>

The direct metrics listed in Table 81 show that there was over 70K lines of code in the application developed by team 1.

Table 82 shows the computed proportions. CYCLO/LOC is a low value but the lines of code/method and the methods per class are high values The legacy code would not have been
modified and is contributing to the high level of methods per class and lines of code per method.

**System Coupling**

Table 83: System Coupling – Direct Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Team 1 (agile)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALLS</td>
<td>19658</td>
</tr>
<tr>
<td>FANOUT</td>
<td>12236</td>
</tr>
</tbody>
</table>

Table 84: System Coupling – Computed Proportions

<table>
<thead>
<tr>
<th>Metric</th>
<th>Team 1 (agile)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALLS/NOM</td>
<td>4.67(H)</td>
</tr>
<tr>
<td>FANOUT/CALLS</td>
<td>0.62(A)</td>
</tr>
</tbody>
</table>

Table 83 contains the direct metrics for the System Coupling characteristic. The computed metrics contained in Table 84 show a high value for number of calls per method.

**Design Flaws**

Table 85: Design Flaws Detected

<table>
<thead>
<tr>
<th>Design Flaw</th>
<th>Team 1 (agile)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brain Class</td>
<td>33</td>
</tr>
<tr>
<td>Brain Method</td>
<td>145</td>
</tr>
<tr>
<td>God Class</td>
<td>7</td>
</tr>
<tr>
<td>Data Class</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 85 shows the number of design flaws detected by iPlasma. Again the code (including legacy) produced using agile methods has a very high number of design flaws.

**Summary**

This comparison illustrates that in this case an application modified using agile methods shows poor characteristics in terms of its low level structuring, system coupling and the number of design flaws detected. It would appear that no major re-factoring of the legacy code was undertaken in which case the characteristics will be determined by the quality of the code base.

7.6 Summary

This thesis makes two overall contributions, namely the impact of the implementation of agile software development on the developers (Contribution 1) and secondly the impact of the implementation of agile software development on the code (Contribution 2). During
interviews with nine different teams, it was found that some teams had implemented agile practices, particularly practices such as pair programming, automated acceptance testing and test driven development, to a greater extent than other teams. This is quantified in terms of an agility index, which is documented earlier in this chapter.

In order to determine the impact of the implementation of agile practices on the code, iPlasma was used to perform an object orientated metrics based analysis of the code. Four different comparisons have been documented above.

This **first** comparison shows that in this case study, the code as analysed using iPlasma was significantly better than code produced by the same team using more traditional methods. The team had the highest “agility index” relative to the other teams studied. There had been improvements in the code in terms of:

- Overall Complexity
- High Level Structuring
- Method Structuring
- Number of Design Flaws

The code produced using agile methods was also more concise. The team themselves attributed their success to Test Driven Development in particular.

“I think it was one of the crucial parts of our success actually for this project really, TDD, this way of working” [00:38:08]

“and I know from those other guys that have not worked with TDD before, now they see the benefit of it” [00:38:37]

The **second** comparison compared code from two teams using agile methods. Team 3 had a lower “agility index” than Team 9 and the results showed a possible direct correlation between the agility index and the code characteristics. In terms of high level, method structuring and cyclomatic complexity there was no clear distinction in the code characteristics between the code produced by team 3 and team 9 when both teams were using agile methods. However for the coupling and cohesion related metrics, code from team 9 showed better characteristics. Also in terms of design flaws, there were also far more design flaws detected in the code produced by team 3. The results show that a team with higher level of agile practices produced code with less design flaws and improved coupling/cohesion metrics.

The **last** analysis used code that was developed using agile methods but the code included a large amount of legacy code. The conclusion was that the code didn’t show any improvement in characteristics as the volume of legacy code was impacting on the result.
The overall conclusion is that code produced using agile methods tended to exhibit improved characteristics in terms of high level, low level structuring, coupling and cohesion and design flaws detected and there appears to be a correlation between the level of agile practices in the team and an improvement in the metrics based characterisation of the design.
8. Conclusion
The intention of this study was to investigate two aspects of a migration from a traditional software methodology to agile methods in a large software development.

<table>
<thead>
<tr>
<th>1</th>
<th>The impact that the introduction of an agile methodology has on the developers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>The impact that the introduction of an agile methodology has on the code being produced.</td>
</tr>
</tbody>
</table>

During the analysis of the data the empirical categories that emerged, led to the findings discussed in chapters 3 to 7. This is one case study and it does not claim to be a complete reflection on practice of agile methods in the industry. The strength of a qualitative study as described in chapters 3 to 6 is to help develop understanding of the phenomena of the research topic by providing rich descriptions, rather than a quantitative summary. In this final chapter, first of all we return to the literature to compare our findings. We then discuss the lessons learnt from combining two research questions i.e. focusing on both the developers and the code. We look at how structured frameworks to assist organisations become more agile and how agile adoption is linked to organisational change. Finally we summarise the contributions of this study.

8.1 Re-visiting the literature

8.1.1 The Traditional Methodology

The traditional methodology used at the case company was essentially a waterfall model. There were separate phases for requirements engineering, design & implementation, testing, release, and maintenance. Between all phases the artifacts for that phase had to pass a quality check, which was known as a “quality door”. FT denotes Feature Test and ST denotes System Test. There would be a number of iterations of the Design-Implementation-Feature Test-System Test cycle. System Test was not included in the agile adoption process.

![Fig. 31. Waterfall Development at the case company](image-url)
In chapter 3 we documented the initial concerns and expectations of developers when they started to migrate to agile methods. The expressed some expectations that some of the issues with their existing traditional methodology would improve. Existing literature documents a number of problems related to the waterfall model. There are not many recent studies on the waterfall model but some of our findings have been identified in existing literature, particularly the issue of documentation.

**Documentation**

The waterfall model is associated with high costs and efforts (Sommerville 2004; McBreen 2002). A large number of documents require formal approval. Changes are costly to incorporate and problems or unfinished work ends up being pushed to later phases (Sommerville 2004). In the case company developers felt that their traditional process was document heavy and some documents did not add value.

**Code Inspections and Unit Testing not finished**

In the traditional model, developers felt that code inspections and unit testing were getting pushed over the edge because they ran out of time at the end of the implementation phase. The literature has described the problems of finished phases left for later phases to solve (Sommerville 2004).

**Better prioritisation of requirements**

Developers thought agile methods would help with the prioritisation of requirements Jarzombek (1999), Johnson (2002) and Bahli and Zeid (2005) touch on the difficulty that the waterfall model poses in trying to manage a large scope of requirements that have to be baselined to continue with development.

8.1.2 Moving to an Agile approach

Dyba and Dingsoyr (2008) conducted an exhaustive systematic review on agile practices and identified a set of relevant literature describing the limitations and benefits of using agile methods. According to the systematic review a majority of the relevant related work focuses on XP (76 % of 36 relevant articles). Dyba and Dingsoyr (2008) raised the following main advantages and disadvantages which are all also evident in this case study. Some advantages and disadvantages related to the on-site customer mentioned by Dyba and Dingsoyr (2008) are not included here since the case company did not have on-site customers.
Communication and Feedback

Agile methods enable better communication and feedback. This is as a result of the short iterations and customer interaction (Svensson and Host 2005; Karlstrom and Runeson 2005; Bahli and Zeid 2005). Better communication enables transfer knowledge (Bahli and Zeid 2005). At the case company, communication and quick feedback was the most positive aspect of agile methods cited by the developers.

General work-environment

From a general day-to-day perspective agile methods are seen to be a preferred way of working (Mannaro et al. 2004). Again in the case company, none of the teams wanted to revert to the traditional approach. Agile was perceived as a better way of working.

Focus on architecture

Some studies identified a lack of focus on the system architecture as an issue (McBreen 2003; Stephens and Rosenberg 2003). Team 2 raised this when they felt that the Design Analysis phase was missing in the agile approach.

Agile does not scale

Bahli and Zeid (2005) make the argument that agile does not scale. That has not proven to apply to our case study. The developers expressed some initial concerns about the possibility to scale agile methods. In this case the company localised the adoption of agile methods to Systems (partly), Design and Feature Test. System Test still used the traditional way of working.

Testing and Integration

Early and continuous testing and integration is a key concept in agile methods. However it is not easy to create the correct environment and can be hindered by different platforms and system dependencies (Svensson and Host 2005). This was certainly a factor seen in our case study, where practices such as TDD, CI and Automated Acceptance Testing were taking some time to get underway.
Team member must be well qualified

According to Merisalo-Rantanen et al. (2005) on the team level team members have to be highly qualified. This was not borne out in our analysis, as all the teams were not more or no less qualified than any other team or individual in the organisation.

8.2 Research Approach

![Fig. 32. Research Approach](image)

As outlined at the beginning, this thesis primarily focusing on two areas:

1. The impact that the introduction of an agile methodology has on the developers.
2. The impact that the introduction of an agile methodology has on the code being produced.

The concept is depicted in Figure 32 above. Findings from the research on Item 1 can be used to explain possible findings with respect to Item 2. This two-pronged approach was chosen in order to obtain a better understanding of the findings. For example if there were no improvements in the characteristics of the code produced using agile methods –that be might be attributed to the fact that the developers were struggling with adopting the agile practices.

Having concluded the study we can say that the qualitative aspect of the study was the most interesting part and it provided a very rich description of what was being studied. The metrics based analysis of the code provided useful data but without having the qualitative information, the metrics would have been meaningless to a large degree. The problem with metrics in isolation is that they can be affected by many factors. First and foremost, it is crucial to know the level to which the agile practices are adopted. Out of the nine teams studied, in chapter 7 we see from Table 66 that this aspect varied quite a bit with a value
range of 9 to 25 for the agility index. For example some teams used pair programming 90% of the time while some teams didn’t use it at all. Other factors such as the amount of legacy code in the design base, the quality of the design base, the type of application being developed, the complexity of the features, Graphical User Interface code that is difficult to unit test, and of course the competence and experience of the team will impact on the quality of the code produced. Software projects by their very nature are unique and it would be virtually impossible even using a controlled experiment to replicate the exact same conditions for two projects. Therefore a level of qualitative assessment is needed. For example, one of the most interesting things was when the metrics found that the code produced by Team 9 using agile methods was significantly better in terms of low level and high level structuring and design flaws detected, we were able to use the qualitative data to see that the developers in the team concurred with that finding. The developer also pinpointed some of the factors that they felt were vital to their success such as TDD, having a least one person experienced with TDD. So the qualitative view was able to strengthen and explain the finding. We also need to consider that the agile adoption takes time, so improvements in metrics based analysis may only be visible over a longer time frame.

8.3 Guiding Agile Adoption in the Future

An important point illustrated in this research was that different teams in the same organisation were operating at different levels of agility. Some reasons for this phenomenon were differences in the types applications being developed, the amount of legacy code, the team composition etc. After concluding the case study and establishing a better understanding of the agile adoption process, the key question of how to help other organisations to become more agile remains. Is there a more structured way of guiding a project or organisation towards agility that can be applied? As already stated in chapter 1, it is evident from the literature that agile methods such as XP and Scrum have been successfully applied across many projects of different size and application domain type, (Takkeuchi and Nonako 1986; Layman et al. 2004; Hodgetts and Philips 2002; Lindvall et al. 2004; Braithwaite and Joyce 2005; Grossman et al. 2004; Nielsen and McMunn 2005). While there are numerous examples of successful agile adoption stories and experience reports and case studies, they are usually focused on a particular agile method such as XP or Scrum and targeted at a specific organisation with its own set of needs. However while the agile community has actively published and shared success stories, a weakness in my opinion is the lack of
publication or sharing of information on cases where agile adoption has failed. The literature has advocated the use of agile mentors and coaches (Sureshchandra and Shrinivasavadhani 2008) during the adoption of agile methods. However the knowledge and experience of the agile coach will depend largely on past experiences and the type of projects that they had previously worked on. Because agile methods are now becoming widely adopted on a large scale, the need for a structured set of guidelines for helping organisations adopt agile methods becomes apparent.

The Capability Maturity Model (CMM) or the improved Capability Maturity Model-Integration (CMM-I) (Emam and Madhavji 2004) were primary tools or drivers for process improvement with the more traditional or plan driven processes. Can some type of similar model or set of guidelines be applied with respect to the adoption of agile methods? The adoption of agile methods is a very complex process and great care is needed when enforcing an element of structure into that process.

8.3.1 Agile Adoption and Transformation Models

In his book called “The Enterprise and Scrum”, Schwaber (2007) documents his opinion on how to transition an organisation to Scrum. Schwaber et al. (2005) produced a whitepaper on Scrum adoption. ADAPT is a model from Cohn (2009) on the adoption of Scrum. Several general models on organisational change have been published such as the Marshall Model of Organisation Evolution (Marshall 2011), the Kotter Model for Organisational Change (Kotter 1996) and Rising and Manns (2004) documents techniques for introducing new ideas in an organisation. Research by Sidky (2007b) includes a value-based agile measurement index, known as the Sidky Agile Measurement Index (SAMI), and a process framework for the adoption of agile practices. Sidky’s research was motivated by the lack of structured approaches in the public domain to guide agile adoption efforts. One downside of Sidky’s framework in my opinion is that it has not been evaluated using a longitudinal study. Instead it has been substantiated using feedback from the Agile community in the form of a survey. However the result of the survey was in general that the SAMI index was sufficient to represent the stages an organisation would go through to achieve agility. SAMI has already been discussed in the previous chapter where an adaptation of Sidky’s SAMI index was used. We see the idea of “agile potential” in Sidky’s work. He suggests that teams are assessed first of all to evaluate their agile potential and then identify a target level for a project. This type of approach is necessary as agile becomes mainstream especially in large organisations. As
we have seen in this study teams within the same overall project, in some cases the same location and within the same organisation were working at different levels of agility. The final thing is that when you have set an agility level for a team, a set of measurement criteria to assess and evaluate if the team has reached the desired level is vital. Moe et al.(2009), present a tool for assessing and improving teamwork in agile software development along the dimensions of shared leadership, team orientation, redundancy, learning and autonomy. However, there is a clear need for more empirical research and longitudinal studies on the use of such frameworks in industry.

8.3.2 Agile Adoption as Organisational Change

According to the latest VersionOne “State of Agile Development Survey Results” (VersionOne 2011), which was based on 6,042 responses, in larger companies (more than 500 employees), only 13% of respondents in larger organisations said that nearly all their projects used agile. In these larger companies, respondents said that lack of management support (27%) and "general resistance to change" (26%) were major barriers to agile adoption. What is becoming apparent is the fact that in large-scale agile adoptions, general organisational change theory and change management becomes important. The survey also cited lack of understanding of the broader organisational change required (11%) as one of the leading causes of failed agile projects. The inability to change their organisation’s culture (52%) was the biggest barrier to further agile adoption.

Most change models e.g. Judson model (Judson 1991), emphasise the importance of communicating the proposed change in an organisation. As we saw in the study, developers had many concerns initially regarding the adoption of agile methods. Many of the concerns resulted from the fact that they were unclear about how certain practices would be adopted such as pair programming, so a key message is that in a large organisation, communication of the change is vital.

When discussing Systems Theory and Learning Organisations, Senge (1993) says that one of the most serious disabilities is when people form a strong identification with their position. What they do becomes a function of their position. They see themselves in specific roles, and are unable to view their jobs as part of a larger system. In our case study, we had three specific roles (Systems Engineer, Designer and Tester) involved in the adoption of agile methods. With the adoption of agile methods these three specific roles would tend to merge into a more generic developer role. Prior to agile adoption, Testers wrote test scripts, but did not write code and had little input into the specification of new features. Systems Engineers
specified the new features and how they would be implemented but did not get involved in coding. Their role could be seen to be somewhat threatened since the emphasis would now be on working code rather than upfront design. The role of the tester might be thought of as elevated since they had now more involvement in decision-making. This is one possible explanation why the cooperation between designers and testers progressed more than the cooperation between systems and the developers.

Schabracq and Cooper (1998) argued that individuals develop general automatic responses to work and life events. As changes begin to mount, coping and adapting are likely to become more difficult. Feelings of uncertainty, the potential likelihood of being unable to cope, and the difficulty inherent in developing revised skills, are all attributable to greater stress levels. Different individuals will have varying ability to learn new roles, stress is expected in the face of change as new skills and behaviors are required (Callan 1993). According to Schabracq and Cooper (1998), as changes begin to mount, coping and adapting are likely to become more difficult and feeling of uncertainty and the potential likelihood of being unable to cope attribute to greater stress levels. This behavior was evident in Team 2.

“When you get into the situation where you are not meeting your goals for each sprint and you are falling further and further behind - I think that's where the pressure really comes, though it is kind of accumulative over a period of time where the whole team starts to feel we are not delivering, we are not delivering and we are getting further behind” [P3] [I20-3: 45]

### 8.4 Contributions

Overall, the results of this case study contribute to a better understanding of the agile adoption process in large organisations. The specific contributions are outlined here.

The study describes how a plan driven approach was employed in a large software organisation and it outlines the organisation’s motivation for adopting an agile methodology. The study illustrates how a mixture of Scrum and XP practices were used in a large organisation.

An in-depth understanding of the key concerns and expectations of the developers regarding agile methods before they adopted them is provided.

The major difficulties posed by the plan driven approach as perceived by the developers are highlighted.

The 6 C’s coding model is used to generate theories related to the plan driven approach and to show how agile methods could rectify problems observed in the traditional methodology.
The study shows that the developers in this case study approached agile “with the appropriate combination or realism, enthusiasm and cautious optimism”.

We see in detail how the views of the developers changed during the migration phase, whether or not their initial concerns and expectations were valid and how new issues arose during the adoption period. The outcomes of the situation before (traditional approach) and after the migration (agile approach) were compared and discussed. This information was captured through interviews, and this illustrates the perception of the effect of the migration. We saw that the focus of the developers moved from surface level agile practices to the core values of agile during the adoption period.

Both one-to-one interviews and group interviews per team were used and the data from the one-to-one interviews and the group interviews broadly concurred but it appeared to be easier to develop consensus in the group interviews.

Issues related to the agile approach were discussed.

Nine different teams were interviewed after the migration and the results show through the use of an “agility index” that different teams progress towards “agility” at different speeds. Different teams within the same overall project were functioning at different levels of agility. The case study shows that iPlasma which is grounded in well proven metrics can be used to characterise and compare Object-Oriented designs.

Findings related to the code characteristics were discussed: In particular, code produced used agile methods showed improvements in terms of overall complexity, high level structuring, method structuring and design flaws detected. Code produced by a team with a lower agility index did not show as many improvements as that of a team with a higher agility index but it did show improvements over the code developed using traditional methods. Applications produced by a team using agile methods did not show significant improvements when the application contained a lot of legacy code. The factors that contributed to the success of a team that produced significantly improved code were also discussed.

The findings from this study are compared to the main advantages and disadvantages described in a systematic review of relevant literature describing the limitations and benefits of using agile methods.

Finally, the advantage of using a twofold approach of combining quantitative metrics analysis of the code with qualitative data obtained through interviews was discussed in the thesis.
References


Appendix A
Appendix A, Initial Concerns and Expectations of the Developers – Guideline Questionnaire

This document contains a list of questions used in a semi-structured interview situation with a cross-section of the software engineers. The interviews took place before the roll-out of the agile methods to take a temperature test of the designers’ attitudes towards Agile. The objective is to obtain detailed information on the following main issues:

1. How the designers feel about the current software design processes/methodology?
2. What they see as issues or problem areas in the current processes?
3. Do they feel that using Agile methods will be an improvement on methods currently used?
4. Are they aspects of Agile that they feel will not work?
5. If the use of Agile methods will make their job more/less enjoyable?

Detailed Questions

Q.1  How long have you worked in software design?
     <1 yr          1-3 yrs       3-5 yrs            > 5 yrs

Q.2  How long have you been employed by this company?
     <1 yr          1-3 yrs       3-5 yrs            > 5 yrs

Q.3  What is your role in the current project (e.g. design, test, systems etc.)?

Q.4  Describe the process that you currently follow for your part of the project lifecycle.

Q.5  Is the process strictly adhered to?

Q.6  In your opinion, what are the (2 biggest) problems that you see with the existing process and how do you think they can be addressed?

Q.7  What is your level of knowledge of Agile methods?
     1. informal presentations/articles
     2. formal training
     3. used Agile methods previously

Q.8  Do you agree with the introduction of Agile practices? If not why not?
Q.9 Do you think that the introduction of Agile methods will solve any of the main issues you see in the current processes? ref Q.6

Q.10 Agile methods introduce concepts such as pair-programming, test driven development, collective ownership. What is your view on those aspects?

Q.11 Are there any particular aspects of Agile methods that you are apprehensive about?

Q.12 Do you feel that your day-to-day work will be more enjoyable as a result of a move towards Agile methods?
Appendix B
This questionnaire is being used as part of the evaluation of the Introduction of Agile Methods in the Organisation. All answers are confidential.

Q.1 How long have you worked in software? Please tick ONE:

| Less than 1 year | 1-3 years | 3-5 years | > 5 years |

Q.2 How much experience has you had of Agile methods? Please tick ONE:

| What I have heard, read myself | Attended Presentations | Attended Training | Worked in a project (pilot) using agile |

For the following questions (Q.3 – Q22), please tick the ONE box that matches your view most closely.

Q.3 “Before introducing agile, the design process we are using is a document driven process.”

| Agree strongly | Agree | Neither agree nor disagree | Disagree | Disagree strongly |

Q.4 “Documentation is important from the point of view of maintaining the system and transfer of knowledge.”

| Agree strongly | Agree | Neither agree nor disagree | Disagree | Disagree strongly |

Q.5 “In the non-agile process we spend a lot of time doing documents just of the sake of the documents. Nobody uses them afterwards.”

| Agree strongly | Agree | Neither agree nor disagree | Disagree | Disagree strongly |
**Q.6** “The Design Impact Specification is not a valuable document.”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
</table>

**Q.7** “It would be easier to write the code itself than to prepare a document such as the Design Impact Specification that fully describes all the technical details.”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
</table>

**Q.8** “In the non-agile world, unit testing and code inspections are the last activity planned before the code is released to feature test and as a result these activities (code review and unit testing) are often compromised or skipped.”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
</table>

**Q.9** “A lot of the agile practices (Test Driven Development, Pair Programming, Stand-up meetings, Co-located and Cross functional teams, User Stories etc.) are just common sense. Calling it Agile is just putting a label on it.”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
</table>

**Q.10** “Agile is just a buzzword, promoted by training and consultancy companies in order to sell training courses and consultancy.”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
</table>

**Q.11** “As a software design organization, we need to try out new processes and continually improve our way of working.”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
</table>
Q.12 “We should concentrate on following the existing process correctly and properly checking the existing quality control points being instead of changing to Agile.”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.13 “There are two many new concepts in Agile to introduce it in one project lifecycle”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.14 “I feel that the introduction of Agile Methods will result in a more pressurised working environment as a result of the short iterations.”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.15 “It will be more difficult to work remotely (home working) in an Agile environment.”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.16 “With regard to pair programming, it will be beneficial for knowledge transfer and competence building.”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.17 “Pair programming is nothing new, we always worked in pairs or groups to solve problems or assist less experienced designers”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Q.18 “With regard to pair programming, conflict of personalities will be an issue”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.19 With regard to pair programming

<table>
<thead>
<tr>
<th>I think pair programming is beneficial</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I think pair programming has benefits, but it depends on how the pairing is done</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I don’t see any benefits in pair programming.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>There could be benefits, but personally I would not like that way of working.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.20 “Agile will

<table>
<thead>
<tr>
<th>solve a lot of the problems in our existing design process</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>solve some problems in our existing design process</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>not make much of a difference</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>introduce a new set of issues and problems that we are not yet aware of</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.21 “It is important that teams are co-located in close proximity to improve communications”

<table>
<thead>
<tr>
<th>Agree strongly</th>
<th>Agree</th>
<th>Neither agree nor disagree</th>
<th>Disagree</th>
<th>Disagree strongly</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q.22 “For me a very open plan environment similar to that used in the Agile pilot would be”

<table>
<thead>
<tr>
<th>something I wouldn’t mind</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>good for pair programming and working in group, but would be too much of a distraction at times</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a work environment that I wouldn’t like at all</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C
Guidelines for second interview with developers as part of Agile evaluation

In this interview we look at the expectations and concerns of the developers from the first interview and ask if their opinion is still the same. Then we also allow for any new issues to be raised. Some of the questions try to access the level of agility i.e. how well the agile practices are sticking?

GENERAL

1. Level of Experience with Agile
   >1 year, > 6 months, training course,

2. Was the physical work environment suitable? Comment?

3. In general was the work environment more/less pressurised?

4. Problems with multiple projects in parallel or other ongoing activities?

5. The Agile adoption process – Was it correct, too aggressive? Would a phased approach have been better?

6. Did you have sufficient support and mentoring during the agile adoption process?

AGILE PRACTICES

7. How well did each of the following agile practices work in your team?
   User- stories,
   Standup meetings
   Pair Programming, (Was personality conflict an issue?)
   Code Reviews and Unit Testing
   Automation of Acceptance Testing
   TDD and Unit Testing,
   Continuous Integration
PROCESS
8. Has the introduction of agile resulted in less documentation? Are there documents being produced now that are not being used? If answer is yes, are there any adverse consequences? And are there any problems with knowledge transfer or system maintenance?

9. Did you use the OSS process framework? Any problems with it?

CO-OPERATION

11. How did the interface between Design and Test work? Continuous supply of User Stories, Did the tester have enough time?

COMMENTS

12. Has Agile wow brought any perceived change in quality/time?

13. Any other positive or negatives that have not already been raised?

14. What support is needed from the organisation to continue this WOW?

15. Do you wish to continue this WOW?