Managing Software Risk in Agile Projects

by

Edzreena Edza Odzaly BMIS (Hons), MSc

A thesis submitted as part of the requirements for the

Doctor of Philosophy

School of Electronics, Electrical Engineering and Computer Science

Queen's University Belfast

July 2014

Abstract

Risk management in software engineering has become a recognized project management practice but it seems that not all companies are systematically applying it. At the same time, agile methods have become popular, partly because proponents claim that agile methods implicitly reduce risks due to for example, more frequent and earlier feedback, shorter periods of development time and easier prediction of cost. Therefore, there is a need to investigate how risk management can be usable in iterative and evolutionary software development processes.

This research work aims to answer this need by building an appropriate and realistic model of risk management and to support this with a tool for managing risk in agile projects. The approach can be characterized as lightweight risk management which provides the needs of risk management but limits the human effort expended. This is achieved by using software agents to carry out risk identification, risk assessment and risk monitoring, the agents making use of data collected from the project environment.

This thesis describes a new solution approach supported by an Agile Risk Tool (ART) which includes a model of the risk environment and support for risk management in agile development environments. In the approach used, the project manager has to define these elements: project goals, problem scenarios, consequences, risk indicators, project environment data as well as specifying risk rules using a predefined 'Rule template'. Therefore risk can be explicitly managed in the early phase of the project, leaving the designated software agents to monitor the rest. The ART model and tool support is evaluated using two case studies, both from student projects. Evidence is therefore provided for the feasibility and applicability of the approach.

Overall, the research contributes a new method for risk management in agile software processes, the necessary tool support to demonstrate the method in practice as well as providing evidence to support the efficacy of the approach. In addition, an example is given of the use of software agents as a potential means to reduce the burden of risk management in software projects.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Des Greer for his never ending support and trust, valuable guidance that he contributed throughout this remarkable PhD journey. He has always been my inspiration, especially in undertaking his work, always available whenever needed and stands beside me through tough times. I would also like to extend my gratitude to my second supervisors, Dr. Darryl Stewart for his valuable input and collaboration towards completing the projects and I have not forgotten Dr. Paul Sage who was a great advisor for the first few years of my PhD study.

I would also like to express my special thanks to all who have kindly participated in the research studies, to my colleagues, the lecturers and member of staff at Queen's University Belfast who have made my stay here pleasant and welcoming.

My exceptional appreciation goes to my beloved husband, Mansur Ussaimi Mohd Salleh, who has always belief in me, for his endless support, patience and motivation and most importantly who understands me for better or worse. This accomplishment is impossible without his sacrifice, leaving his career to let me achieve mine. The greatest gifts from God are my lovely daughters who will always be my bundles of joy: 'Adaleia, 'Aqleia and 'Areiqa who also provided a welcome distraction of attention at many times.

I would also like to extend my appreciation to my beloved mum, for her love, prayers and unconditional support throughout this journey and her always being a helping hand. To my family in law, who always understand and deliver their love and prayers from afar, thank you. I have not forgotten my siblings, relatives and friends who have always been understanding and supportive.

Finally, I would like to acknowledge the financial assistance provided by my sponsor Universiti Teknologi MARA (UiTM), during the course of this PhD work.

Table of Contents

List of Tables

List of Figures

Declaration of Authorship

I, Edzreena Edza Odzaly, declare that the thesis entitled

MANAGING SOFTWARE RISK IN AGILE PROJECTS

and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research.

I confirm that:

- ☑ this work was done wholly or mainly while in candidature for a research degree at this University;
- ☑ where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- ☑ where the research work in this thesis as described in chapters 3 to 7 is my own work, therein I have made clear exactly what was done by others (some assistance in data collection) and what I have contributed myself ;
- ☑ parts of this work have been published as below and presumably some text from these is included in the thesis:

  Odzaly, E.E., Greer, D. & Sage, P. "Software Risk Management Barriers: An Empirical Study", Proc. 3rd ACM/IEEE Empirical Software Engineering and Measurement (ESEM), IEEE Computer Society Press, pp. 418-421, 2009.

  Odzaly, E.E., Greer, D. & Stewart, D. “Lightweight risk management in Agile projects”, Proc. International Conference on Software Engineering & Knowledge Engineering, (SEKE), 2014.

Signed : ……..

Date :……

Chapter 1 - Introduction

1.1 The Thesis

"If you don't actively attack risks, they will actively attack you" (Gilb, 1988). This statement shows that the importance of Software Risk Management has been recognized for several decades. Ever since its evolution into a recognized software process activity, risk management has been seen as essential to increase the chances of project success, by avoiding cost overruns and project delays (Boehm, 1989; Keil et. al., 1998; Charette, 2005). Despite many well known risk management approaches having been developed, it was still reported as not being widely practiced (Ibbs and Kwak, 2000; Pfleeger, 2000) often due to the required activities being effort intensive (Hall, 1998). At the same time, agile methods have become popular, partly because proponents claim that agile methods implicitly reduce risks due to, for example, more frequent and earlier feedback, shorter periods of development time and easier prediction of cost (Schwaber and Beedle, 2002; Cohn, 2005). Risk management requires data collection and defined processes as part of a rigorous approach. Compare this with the expectation that agile approaches should be lightweight and it can be easily seen that agile methods and risk management are not often considered a good fit.

The main argument in this thesis is that a risk management approach can be usable in modern agile software development environments. It is assumed that there is a need for

a lightweight risk management with an inherent reduction in human effort. The proposal is, to manifest this by using software agents to carry out the necessary tasks to identify, assess and monitor risks, using input and data collected from the project environment.

1.2 What is Risk?

The Oxford dictionary defines 'risk' as an exposure to danger, harm or loss. In other words, risk can be seen as an event with a negative impact that may or may not occur in future. In the field of engineering, Bell (1989) defined risk as "*a combination of the probability of an undesirable event with the magnitude of each and every foreseeable consequence (damage to property, loss of money, injury to people, and lives lost)*". Risk can be considered as intangible because it refers to a possible future event or situation. The event that causes the risk is not definite and the impact is not certain.

In estimating and measuring risk, Boehm (1989) defines Risk Exposure as a fundamental concept that can be used to quantify risk:

Risk Exposure (RE) = Prob(UO) x Loss(UO) where;

Prob (UO) refers to the probability of an unsatisfactory outcome

Loss (UO) refers to the impact of the unsatisfactory outcome

The quantitative statement of probability is expressed as a number, p, where $0 \leq p \leq 1$ whereas an impact can be quantified as potential loss that is usually expressed in monetary units (Fairley, 2008). For example, a risk that has a probability p = 0.5 of happening and a possible loss of L = £25,000 will have a risk exposure of RE = £12,500. A lower risk exposure can be achieved by reducing the probability or by reducing the associated loss.

1.3 The Importance of Risk Management

Risk management has been in place not only in software engineering but in many other areas such as business and finance, security, health and safety, and property. Software Risk Management as defined by Boehm (1989) involves a set of processes to manage risk items for which failure to perform will cause threats to successful software project outcomes or an unreasonable amount of rework. The process involves identifying risk, assessing and prioritizing risk, as well as monitoring and controlling risk.

1.3.1 Risk Management as a Best Practice to ensure Project Success

Risk management has for a long time been acknowledged as a best practice in software development projects (Wiegers, 1998). A viable risk management process can make the difference between the success and failure of software projects. For example, suppose two different projects are using the same risk management methods, where project A has carefully evaluated risk at an early stage of the project and continues to monitor them throughout. Project B also detects possible risk but fails to monitor the risk. Sometimes the risk will occur and in such cases for project A, where the risk is known, it can be prevented unlike in project B. Consequently, applying effective risk management is a process which will increase the likelihood of project success.

As reported by Charette (2005), the U.S government spent 60 billion dollars on software, indicating a 5% failure rate and speculated an increase in this of up to 15% depending on the size of the investment. Looking at the total investment on software over five years he estimated that the project failure cost could reach up to 75 billion dollars. In 2004, the UK Inland revenue office reported software failures for systems that were supposed to provide a tax-credit system for poor families. The system errors resulted in nearly two million people being overpaid, a consequence which impacted mostly on poor families (Charette, 2005). Further in the UK, the failure of the largest ambulance service in the world; the London Ambulance Service Computer Aided Dispatch (LASCAD) system operational failure resulted in 8.6 million pounds waste and threatened the lives of many people. In that case, the failure was that incomplete software with serious system flaws was put into use (Beynon-Davis, 1995). According to one study of 600 companies developing software, 35% of them had at least one 'runaway' software project, where 'runaway' means any project that completely exceeded the target budget and time and yet did not make any satisfactory deliverable (Rothfeder, 1988). One of the indicators found was that a high level of enthusiasm during the early phases of projects led to disregard of the high risk elements at the start. Keil *et. al.* (1998) claimed that project managers not taking steps to identify, assess and manage risks have led to higher failure rates. Besides this, Ewusi-Mensah and Przasnyski (1991) provided evidence to show that 35% of abandoned projects are not abandoned until the implementation stage of the project's life cycle. All of this suggests historically that many project managers are doing a poor job of identifying or terminating projects that are likely to fail.

Many recent empirical studies discuss the contributions of risk management in relation to project success (Huang and Han, 2008; Han and Huang, 2007; Wallace et. al., 2004). Each of them applied a similar risk management approach to evaluate generic risks for future projects. However, they use different definitions of project success. Huang and Han (2008) emphasize that it is important that project managers effectively manage risk in timely manner. Meanwhile Han and Huang (2007) relate project success to performance of the project. Wallace et. al. (2004) sticks to the traditional definition of project success where meeting requirements, the cost and the time taken are the only important criteria to project success.

Nevertheless, risk management in software projects has been identified as important for many years (Boehm, 1989; Charette, 1989) and ample studies done showing the utility of risk management in software projects, no matter how excellent the project team (Wallace et. al., 2004; Han and Huang, 2007; Odzaly et.al., 2009). According to Boehm (1991), the discipline of software risk management is an attempt to formalize these risk-oriented correlations of success into a readily applicable set of principles and practices. However, these principles and practices should be deployed as an integral part of the software process.

1.3.2 To Complete the Software Engineering Process

There are numerous models in existence for the software process. Among them are Waterfall, Spiral, Iterative and Incremental Development (Larman and Basili, 2003), and Agile Development. The Spiral model (Boehm, 1988) is particularly relevant here since it assumes risk management as a central activity applied at regular stages in software development cycles. This model includes four different dimensions; determining objectives, identifying and resolving risks, development and finally testing and planning for the next iteration. This risk driven and cyclic approach can be manifested as waterfall, prototyping or iterative approaches. The risk driven approach, has yet failed to be advanced to popular adoption. According to Boehm (2006), the risk driven approach contained insufficient guidance on how to keep all of these concurrent activities synchronized and stabilized. While there have been many studies done about improving risk management, there are few dealing with iterative approaches.

Nonetheless, risk management has been recognized as an important aspect in contemporary software development due to its benefits and contributions in software

engineering area. Kwak and Stoddard (2004) emphasized the great benefits of implementing effective risk management tools and techniques in software development projects. They proposed that an effective risk management process will succeed by changing the organizational culture. Ropponen and Lyytinen (2000) provide evidence that the use of risk management can address risks when tailored to their development environment. Many studies describe risk management as crucial in software development projects (Kontio, 1999; Han and Huang, 2007; Dey et. al., 2007).

1.3.3 Other Advantages of Risk Management

Boehm (1989) describes risk management as important for four major reasons:

- avoiding software project failures, including runaway costs and schedules;
- avoiding rework which caused by flawed, missing or vague requirements, design or code, which typically can consume 40-50% of the total cost of software development;
- avoiding overkill by detecting and preventing risks early;
- stimulating a win-win situation in which both parties, customer and vendor, gain benefits.

Different projects will have different risks. Generic descriptions of risks can however be established in order to assist analysis in diverse projects. Thus, software risk knowledge can be reused across projects enabling the building of a risk profile (Ramamoorthy et. al., 1993). Established risks can be kept, reviewed and updated in a database.

1.4 Motivation

The motivations of this thesis are as follows.

I. Motivating scenario

In 2004, the author was involved in a development team of MyRAM (Malaysian Public Sector Information Security Risk Assessment Methodology) project in a company based in Malaysia. MyRAM was an initiative from the government of Malaysia to provide government agencies with a qualitative methodology for implementing a detailed information security risk assessment within the predefined scope of the agencies. The

developed methodology was embedded in the MyRAM tool and later was installed in each of the agencies. During the implementation of the MyRAM tool the author observed that the research team was having difficulty in finding a suitable process on which to base the MyRAM methodology. The team took approximately 18 months to establish a framework due to the fact that there was no established best practice or risk assessment standard that was suitable for the government agencies. Feedback on the tool usage later in 2007 revealed that there were problems regarding the multifaceted process of assessing risk, in particular the effort required in doing risk management, but also the risk management knowledge and expertise needed to understand the process. As a result, the tool was not being used efficiently by the agencies. Those using the tool were observed to have an awareness of the need for risk management but this was countered by the effort-intensive process involved and the difficulties of understanding the risk management process itself. Although this is only anecdotal evidence, the motivation to investigate risk management further was established, the first step being to establish from industry and from published literature if these problems were generic and widespread.

II. Research gaps

Over the years, substantial attention has been paid to risk management as evidenced by the number of risk management methods that have been developed and evaluated, such as RISKIT (Kontio et. al. 1998), SoDIS (Gotterbarn and Rogerson, 2005), and many more. More recently there have been fewer contributions and not as many research papers have been presented in this area. This is backed up by referring to journals from the IEEE and ACM digital libraries. These provide a result of 4539 papers, over a period of 2003 until 2013 for a search of 'Software AND Risk' in the abstract, title text and indexing terms. Narrowing the result with the search "risk management" provides 224 hits and out of this sum only 29 papers were produced in 2012 and 23 papers to date in 2013. At the same time a cursory read through the computing related press will reveal that failures are still common in software projects, implying that risk management is not being applied universally satisfactorily or that it is not working. According to Kontio (2002), risk management needs to change and a new software risk management discipline should be invented. In his paper he suggested risk management hypotheses and new agendas which include the following characteristics; business attitude to risk-taking and opportunity, the need for unbiased risk methods, the need for easy and cost effective approaches, situation specific approaches, to emphasize

communication and clarity of the results and lastly, researchers to provide practical results while consultants need to upgrade their risk knowledge. Although the need for change in risk management has been addressed, it was found that there are still gaps between risk management in research and practice (Bannerman, 2008) and it is still an immature discipline (Islam, 2011).

One common perception nowadays is that failure has become an accepted occurrence in software development project. Glass (1998) discussed that the term 'runaway' project, a term that has become frequently referred to. This is due to failure to identify and control risks early on.

The SEI Risk Management program was initiated in the early 1990s with a goal to assist in success for software projects by improving the practice of risk management for software-intensive projects**.** SEI studies showed that only a few programs were managing risk in a systematic way, and that the approaches of the programs that did manage risk tended to be ad hoc, undocumented, and incomplete (Kirkpatrick et. al., 1992). However, from 2000, the SEI Risk Management structured process had started to be recognized and was manifested as three different methods: Software Risk Evaluation (SRE), Team Risk Management (TRM) and Continuous Risk Management (CRM) (Williams, 2003).

Despite the acceptance that risk management methods enhance system development performance, nonetheless little support is to be found on the provision of these methods (Ropponen and Lyytinen, 1997). It was argued that the methods of managing risk in software development are not comprehensive as they deal with specific types of risk (Bandyopadhyay et. al., 1999). Besides, despite many well known risk management approaches having been introduced, risk management was still reported as not being well practiced (Ibbs and Kwak, 2000; Pfleeger, 2000). Bannerman (2008) discussed the most common risk management approaches found in the literature and highlighted practices such as checklists, analytical frameworks, process models and risk response strategies. Many researchers have conducted research in tailoring risk management, providing various approaches. However, only a few studies have been reported to integrate risk management with contemporary software development. Nyfjord and Kajko-Mattsson (2008) discovered that there was still plenty of work to be done due to the fact that the integration of risk management and software development process was still at its initial stages.

As early as 1996, Higuera and Haimes (1996) stated that risk management is difficult to implement due to its complexity. Hall (1998) argued that there is an attitude that assumes risk management to be an extra activity, less important and even an assumption that it was not the project manager's responsibility at all. Rather, the argument she gives is that risk management is everyone's responsibility not just the focus of a project manager. These barriers to effective risk management may still apply and, in addition, other possible explanations found in the literature are listed below.

- risks are not understood well (Ropponen & Lyytinen, 2000);
- there are too many risks to manage within resource limitations e.g. time, cost and personnel availability (Keshlaf and Hashim, 2000); or
- there is a lack of motivation among developers to perform risk management (exemplified by developers being too optimistic or simply not prioritizing risk high enough) (Kwak & Stoddard, 2004).

1.5 Aim and Objectives

The aim of this research is to present a new and sound risk management discipline built on a usable risk management approach in modern software development environments. Specifically, the aim is to resolve the research problem by building a novel and reliable model of risk management and to support this with a tool for managing risk in agile projects. While there are many diverse types of tools and techniques presented before, none of these integrate risk management with iterative software processes such as agile development. On the other hand, many studies in risk management have been done previously addressing improved techniques and tools. What is envisaged is that risk management should be easy to carry out, and parts of it, where possible, automated. Given that current practices are highly centred on human effort, (Odzaly et. al., 2009) there is a need to investigate how current research on software agents could be applied to risk management. Such an approach would add to software agent research, thus providing an initial demonstration by providing an example of its successful application to project management tools. The hope is to further the discipline for risk management by supplying new methods and tools that make it more likely to be applied. One approach proposed in this thesis is the link to software agent research. Software agents

are appropriate because unlike conventional software components, such as those that are object based they are not passive and so can act on behalf of people to achieve goals. Intuitively, this is exactly what is needed, given the problem of human effort in software risk management. In such a scenario, it is possible to give agents the necessary intelligence to make judgments and decisions regarding risks. Adding intelligence equivalent to a human risk manager would be very ambitious. However, the first step is to demonstrate that software agents could be given goals and then autonomously assess and monitor risk. Ultimately, the aim is to extend an agent system to the point where it can make intelligent judgment and support or even make decisions. The proposed approach therefore includes a risk tool that can used with minimal human intervention to identify, analyse and monitor risk, then react intelligently and dynamically to changes in the project environment. The reactions used for example are to alert another agent or human about a risk event or condition, and to respond to changes in the project environment. This would reduce the burden and lower the barriers discussed previously, thus helping organizations to manage the complex risk management steps in a software development project specifically in agile projects. As discussed in the next chapter it will show that risk management is in practice rarely formally employed in agile methods.

In developing the new solution approach the following elements of investigation were included.

(i) An investigation of the issues in software risk management including a regional empirical assessment of the current state of the art for risk management application in industry.

(ii) An investigation of the issues in agile projects that can be considered a threat to a project.

(iii) An investigation into which project environment data in typical agile projects can be used in software risk management on which to base automation of risk management.

All of these elements will be discussed further in Chapter 3 and 4.

Based on the summaries of investigations carried out previously, there is a need to define the following objectives arising from the research problem.

(i) Develop an investigation into what hinders the software companies from doing risk management.

(ii) Develop a new risk management model and tool support which is lightweight, non-intrusive and with the ability to make use of data collected from the software development environment.

(iii) Demonstrate the applicability of the model and tool support in practice providing information on its feasibility as well as improving knowledge on managing risks in an agile environment.

1.6 Contributions

Software engineering as defined by Sommerville (2011) *"Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use. ... Engineers make things work. They apply theories, methods, and tools where these are appropriate".* In this research work, the aim is add to and reinforce existing theories of software risk management and to provide a new method and tool to support software risk management, thus contributing to the software engineering body of knowledge.

This aim is manifested in a primary objective and main contribution of this research as being a new proposed approach and tool support, called the Agile Risk Tool (ART). The model provides significant characteristics that are embedded in the ART process model which includes the definition of project or sprint goals and decomposing the goals into problem scenarios that can be considered threats to a project. The novelty in the ART model tool support is that it provides the means to capture data from a project environment and use this for risk management thus obviating some of the issues causing non-performance of risk management, chiefly the issue of the excessive effort required in capturing risks. The new method and tool support are evaluated using case studies and evidence given that they improve knowledge on managing risks. The ART model and its process will be discussed in detail in Chapter 4.

Besides its unique characteristics, the ART model and tool support contributes to an implementation of Continuous Risk Management (Dorofee et. al., 1996) but where the

cycle is done intelligently thus reducing human effort in managing risks. This claim will be supported in detail in Chapter 4.

Another contribution is that to the author's knowledge, this work provides the first model to integrate risk management and agile development, making use of software agents. In summary the contribution has three components:

(i) A New Lightweight Risk Management Approach

(ii) Risk Management has been integrated into agile development

(iii) Autonomous Computing - software agents provide continuous risk management. Ultimately, each software agent will achieve a designated goal i.e. to identify, assess, prioritize and monitor risk and to operate independently

The evaluation of the ART model and tool support is provided in two case studies which provide evidence that risk can be easily managed and data in relation to the risk can be collected non-intrusively. This will be described in Chapter 5. Empirical evidence of the feasibility and applicability of the ART model and tool support is provided in Chapters 6 and 7. Within these chapters a number of interesting risks results relating to people and process are also presented. These risk results have contributed to evidence presented on how non-compliance with established agile practices generates risk and affects product quality.

1.7 Thesis Structure

This thesis is organized into eight chapters followed by references and appendices. The overview of the overall content in this thesis is illustrated as in Figure 1-1 below.

This chapter introduced the thesis, discussed the definition of risk, important values in risk management, research motivation and inspiration, research aims and objectives, research contributions and the structure of the thesis.

Chapter 2 contains a literature review and also provides more explanation of current software risk management models and agile development methods. Related work similar to the research work is also given detailed attention.

Chapter 3 discusses the problem statement and reports on the current state of software risk management in a regional software industry survey. Based on the problems identified, a set of research questions are defined in this chapter.

Chapter 4 discusses in detail the development of the proposed solution approach. This is accompanied by the description of the tool that is integral to the process.

Chapter 5 describes the methodology used in the research work accompanied by a description of two case studies used to evaluate the solution approach.

Chapter 6 presents the results and analysis for the first case study.

Chapter 7 presents the results and analysis for a second case study which integrates improvements based on lessons learned from the first case study.

Chapter 8 concludes the thesis. The novel contributions are discussed along with the research questions and an evaluation of how they have been answered. Finally, the limitations of the work and the possibilities for improvement and future work are discussed.

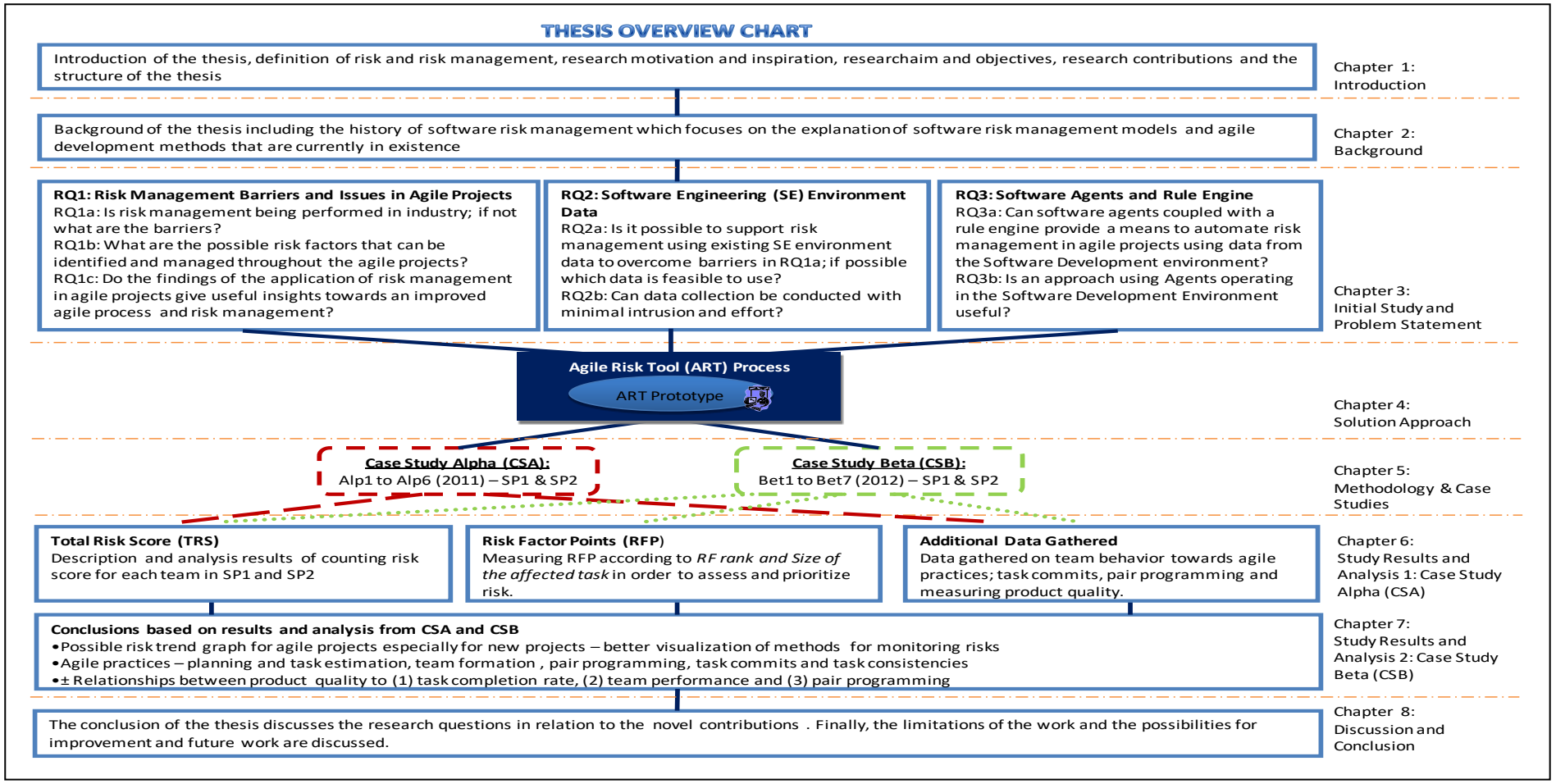


Figure 1-1: An overview on the overall content of the thesis

Chapter 2 - Background

2.1 Introduction

Risk management has a wide-ranging meaning and has been applied in a number of diverse disciplines. Its application has been in practice for many years. However, risk management in the software engineering domain is a relatively young discipline (Brown, 1996; Islam, 2011) and still developing (Misra et. al., 2007). Nonetheless, there has been substantial work aiming to establish risk management, both implicitly in the software process and also as explicitly as various activities in project management. Agile methods could be argued to be one such implicit risk reducing process. Agile adherents argue that agile methods are inherently risk reducing. Generally, agile methods are considered to be lightweight, a property exemplified in the agile manifesto's stated preference for face to face communication over documentation (Fowler and Highsmith, 2001). Nonetheless, there are no claims that agile methods remove all risks and so in order to make risk management effective some necessary risk management activity may still need to be added. To avoid adding unnecessary weight to the process this thesis will consider the possibility of moving towards automation or even autonomy in some aspects of risk management.

This chapter covers the three important software engineering areas that relate to this research work. Firstly, this starts with an introduction to software risk management and its background and history, including popular risk management models. Secondly, discussions on the background of agile development including the two most popular approaches, eXtreme Programming and Scrum. This is followed by a brief description of autonomous computing.

2.2 Introduction to Software Risk Management

According to Charette (2005), software projects are high risk activities, generating variable performance outcomes. Charette stated that industry surveys suggest that only about a quarter of software projects succeed outright – that is that they complete as scheduled, budgeted and specified. Billions of dollars are lost annually through project failures or projects that do not deliver promised benefits. Success is however open to interpretation and Cohn (2005) coming from an agile methods perspective has a different view on defining project success and failure. Success is taken as relating to the critical elements in development of a project which are estimating and planning. In his view failure would be to deliver the initially specified project without any advancement on it as the project progresses.

Boehm (1989) defines software risk management as a set of principles and practices aimed at identifying, analysing and handling risk factors to improve chances of achieving a successful project outcome and/or to avoid project failure. This practice however, seems almost unattainable as evidenced by Kwak and Ibbs (2000) who identified risk management as being the least practiced discipline among different project management knowledge areas.

Several software risk management techniques, models, tools and standards exist to address the need for more effective risk management. Prominent examples are: the Spiral Model (Boehm, 1988), Software Risk Evaluation (SRE) and SEI Continuous Risk Management methods from the U.S Software Engineering Institute (SEI, 2008), the IEEE 1540 Risk Management Standard (IEEE1540, 2001) and the Project Management Body of Knowledge (PMBoK) (PMI, 2008).

Padayachee (2002) stated that software risk management generally follows the same structure as prescribed in risk management standards and approaches in other

disciplines. The first step in a typical risk management program is the identification of risk, usually involving such tools as the use of checklists, questionnaires or brainstorming sessions. The next step is the analysis of the risks identified in such a way that they can be prioritized in a meaningful manner.

This is followed by the development of countermeasures to either prevent risks from affecting the project or to reduce their impact. The final step is risk monitoring, which ensures that the risk reducing methods are implemented effectively and determines whether or not the risk-reducing tactics are in fact reducing risks.

2.3 The History of Software Risk Management

Misra *et. al.* (2006) has discussed the popular risk management models put forward by various groups and individuals.

Barry Boehm is one of the main pioneer researchers in this area, his main contributions coming from the mid-eighties. Boehm also well known for his work on software cost estimation, risk management and the risk-driven Spiral model of the software development process. In the Spiral Model (Boehm, 1988), risks are handled at the early stages of software development, so avoiding project problems at the later, more costly stages. The original Spiral Model has been refined and extended to the 'Theory W (Win-Win) Model' which contains new elements and more details of their goals (Boehm et. al, 1998). The model focuses to determine stakeholders win conditions using negotiation which seek mutual satisfaction between stakeholders. This model is coupled with a risk management framework that can be integrated into the original Spiral, or the Win-Win Model which supports risk identification, risk resolution, and the continuous monitoring of risks (Boehm, 1991). Boehm and Bose (1994) emphasized the key contribution is that the win conditions have engaged the stakeholders towards in the risk management process in order to ensure project success.

The RiskMan methodology (Carter et. al., 1994) advances Boehm's steps with a process for risk management that is closely linked to historical data and a project management tool. The methodology discussed five important steps:

i. Risk Identification where risks are identified from a database of standard risks.

ii. Risk Assessment where risks are assigned to an estimation of probability and cost for each project activity.

iii. Mitigation Action Definition where a particular risk is assigned to defined mitigation action.

iv. Risk Follow-up is where risk indicators are defined and monitored throughout the project.

v. Knowledge Capitalization is where new knowledge learned during the project is added to the knowledgebase.

The Software Engineering Institute (SEI) at Carnegie Mellon University has been one of the biggest contributors to software risk management research and practice. They initiated the Software Risk program in 1996 and, arising from that, provided a comprehensive risk management framework comprised of the following three groups of practices: Software Risk Evaluation, Continuous Risk Management, and Team Risk Management. The Software Risk Evaluation conducts four primary functions which include detection, specification, assessment and consolidation. Throughout its lifecycle, the approach utilizes a risk taxonomy, which consists of constructs used for organizing risk information. The taxonomy helps in providing with an instrument in form of questionnaire to stimulate different classes of risks (Higuera and Haimes, 1996). The SEI Software Risk Program has since been incorporated to the SW-CMM which provides guidance and procedures for an organization in managing and developing software processes (Higuera and Haimes, 1996). Latterly, this work further has been incorporated into the Capability Maturity Model Integration (CMMI) in which risk management is one of sixteen 'key process areas' (SEI, 2008).

Hall (1998) introduced an alternative view of software risk management by categorizing four different factors that influence the success or otherwise of a project. These factors are People, Process, Infrastructure, and Implementation ("P^2I^2"). The 'People' factor refers to the people who participate in the risk management process and who will communicate issues that can turn into risks. The 'Process' factor includes executing the processes that should be carried out to manage risks in order to reduce uncertainties in a project. The 'Infrastructure' factor refers to the establishing environment, requirements,

resources and results required to perform risk management activities. Finally, the 'Implementation' factor refers to the plan and methodology that will guide the actual implementation of risk management activities such as: (i) having a high quality plan that maps resources to risk management activities and (ii) mechanisms, techniques, automated tool to support risk management implementation.

Karolak (1998) has developed a 'Just-In-Time' approach which endeavours to reduce the risks involved and at the same time improving the contingency strategies when facing challenging predicaments. It is a risk-driven approach and promotes the theory of managing risk at the beginning of a project life cycle so that project cost and time can be reduced as well as an improvement of customer expectations. The initial step in this approach is to identify a set of high-level risk categories and associate these risk categories to risk factors, risk metrics, and a set of questions. These questions act as a checklist that can be used to identify different classes of risks.

Kontio (2001) proposed a risk management methodology, 'Riskit'. Riskit was developed based on a theoretical framework of risk management that proposed a goal and stakeholder-oriented approach. The approach endeavours to manage risks by addressing risks particularly associated to stakeholders' behaviours towards achieving the project goals. Thus, risk can be identified if any of the behaviour acts negatively towards the goal. As a result of its implementation, it is claimed that this method helps project managers with the accurate and timely dissemination of project information, opportunity, and risk to different stakeholders, thereby enabling them to make critical decisions for the overall success of the project. The Riskit method uses a Riskit Analysis Graph to analyse all aspects of risks; risk factors, risk events, risk outcomes, risk counter-actions, risk effects, and utility loss that could arise as a result of risk event. In addition to that, he proposed a risk management process improvement framework adapted from Victor Basili's Experience Factory (Basili, 1993). Thus, understanding of the Riskit Process Management Improvement (PMI) Framework requires an understanding of an 'Experience Repository'. The main reason for this is that the Riskit PMI Framework employs experience and information from previous software development projects for managing risks in the current project.

Charette is another well-known expert in this area, and is an internationally acknowledged authority and pioneer in risk management. Most of his papers are based on his experience working with the industry and with real world case studies (Charette,

1996, 2005). Charette (2006) believes that a universal risk management standard is unnecessary or undesirable since the nature of risk will change, based on the economic situation and on human inventiveness. Charette has also contributed two seminal books in software risk management (Charette, 1989, 1991).

Other recent approaches are discussed briefly as follows. Foo and Murugananthan's (2000) approach is called Software Risk Assessment Model (SRAM) based on a set of questionnaires to assess risk and predict outcomes of software projects. Deursen and Kuipers' (2003) proposed a risk assessment approach through a concept of 'primary facts' and 'secondary facts' where any gaps interpreted from the two facts are formed into a concise risk assessment. Misra et. al. (2005) proposed an approach that can be used by project managers to model and control risk in software projects. The approach applies a concept of investigating the root cause at the start of the project and focuses on dependencies between actors, motivations and activities in the project. Each of these approaches proposes risk analysis methodologies rather than a complete risk management framework, unlike the approaches described above.

2.4 Software Risk Management Models

Westfall (2001) describes the risk management process as an on-going activity for managing the software development process which contains constant feedback after each activity. This allows the use of additional information and updating of risk status takes place in order to refine the project's risk list and risk management plans. The following sub sections briefly discuss four well known risk management models that have similar structure and processes. Rabbi and Mannan (2008) argued that there is no actual dominating risk management approach, as different kinds of approaches have been introduced over time by different researchers and according to their needs.

2.4.1 Software Risk Evaluation Method (SRE)

William et. al. (1999) outlined Software Risk Evaluation (SRE) as a tool that supports project decision making in order to distinguish specific project risk statements derived from various sources like product, process and constraint. SRE was initiated from the Software Engineering Institute (SEI) which offers a structured early warning mechanism for anticipating risks at the beginning of a project. It also presents a set of

activities that can initiate the risk management process and enhance this by integrating the activities with the existing methods.

The SRE method practices Continuous Risk Management (CRM) which is a constituent of the SEI risk management paradigm (SEI, 2008) which offers software engineering practices with processes, methods and tools in managing project risks. This includes three main steps for managing risks; continuously assess risks, determine the priority of the risks and carry out strategies to attend to the risks. Dorofee et. al. (1996) implies that SRE can be more efficient if used together with CRM.

The CRM model provides a context upon which all risks can be managed continuously and iteratively throughout the project lifecycle. As depicted in Figure 2-1, the cycle includes the following elements; identify, analyse, plan, track, control and communicate which cycle through until the project ends. Therefore, new risks can be identified, analysed and planned at the same time as while the old risks are tracked and controlled.

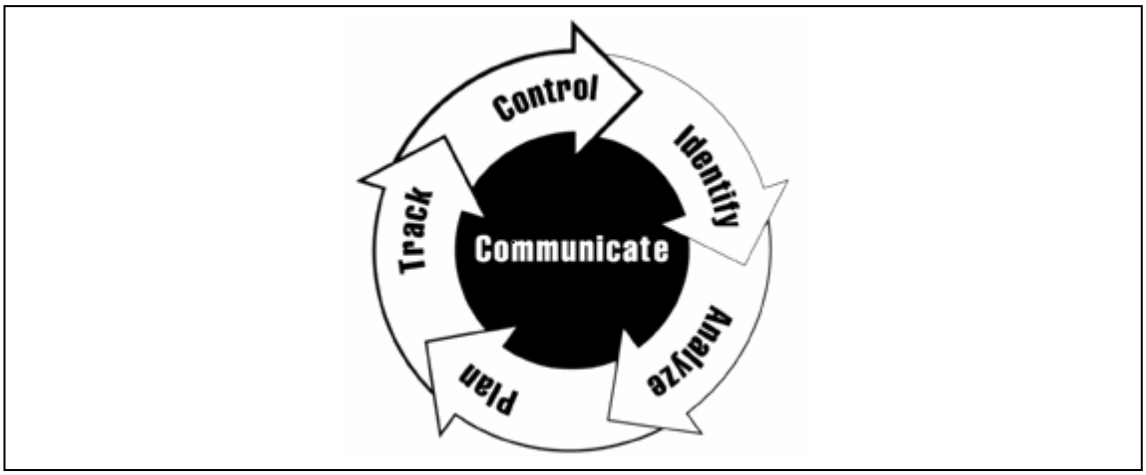


Figure 2-1: Continuous Risk Management process (Source: SEI, 2008)

2.4.2 Project Management Body of Knowledge (PMBoK)

The Project Management Body of Knowledge (PMBoK) was initiated by the Project Management Institute (PMI) (PMI, 2004). PMBoK was claimed by the Institute to be accepted as one of the best practices within the project management discipline and suitable to be practiced in any type of project, not only software projects. PMBoK consists of a collection of processes and knowledge areas which are recognized globally via IEEE Standard 1490-2003, providing the fundamentals of project management irrespective of any type of project.

Within PMBoK, risk management is included as one of the nine main knowledge areas of project management (PMI, 2004). Risk management in the PMBoK is defined as "*the systematic process that identifies, analyses and responds to risk in projects*" (PMI, 2004). The process is aimed to maximize the likelihood of positive effects and minimize the likelihood of the adverse impact towards project goals.

The knowledge area concerns project risk management and describes six processes that are necessary for conducting risk management as described below (PMI, 2004):

- a *Risk management planning process* for determining the approach, plan and execution of the risk management activities;
- a *Risk identification process* for defining risks at an early stage, before the risks become a threat to the project, and recording the risks in the Risk Register;
- a *Qualitative risk analysis process* for prioritizing the identified risks for subsequent analysis or action by assessing and combining their probability of occurrence and impact;
- a *Quantitative risk analysis process* for numerically analysing the effect on overall project objectives of the identified risks;
- a *Risk response planning process* for generating possibilities and execution plan to improve chances to succeed and to minimize threats to project; and
- a *Risk monitoring and control process* for tracking identified risks, monitoring risks, identifying new risks, executing risk response plans, and assessing their efficacy throughout the project life cycle.

2.4.3 AS/NZS 4360 Risk Management Standard

The AS/NZS 4360 Risk Management Standard is a standard that offers a basic procedure that can be applied at many levels in an organization whether at strategic and operational levels as well as all stages in the life of an activity, function, project, product or asset. (Standards Australia, 2004).

The standard illustrates risk management as a vital aspect in order to gain an outstanding management practice. It involves an iterative process consisting of steps, starting with establishing the context, identifying, analysing, evaluating, treating, monitoring and communicating risks which will facilitate frequent development in

decision making when engaged in sequence. Within each cycle, risk criteria can be reinforced to achieve increasingly better levels of risk management. On the other hand, risk management is viewed as more for identifying opportunities rather than avoiding or mitigating losses.

The AS/NZS 43 60 risk management process contains the main elements below and as depicted in Figure 2-2 (Standards Australia, 2004):

i. *Establish the context* in the form of establishing where risk management needs to be applied the criteria for and how risks will be analysed.

ii. *Identify risks* in classifying what, why and how things can arise as the basis to investigate further.

iii. *Analyse risks* in order to regulate control of the risks and examine risks based on consequence and likelihood of occurrence.

iv. *Evaluate risks* to set a priority for the risk and compare estimated levels of risk against the pre-established criteria.

v. *Treat risks* meaning to accept and monitor low-priority risks. Otherwise, a specific management plan needs to be developed.

vi. *Monitor and review* the enactment of the risk management system and observation of changes which might affect it.

vii. *Communicate and consult* among project members whenever possible in relation to the risk management process and any concern of risk issues.

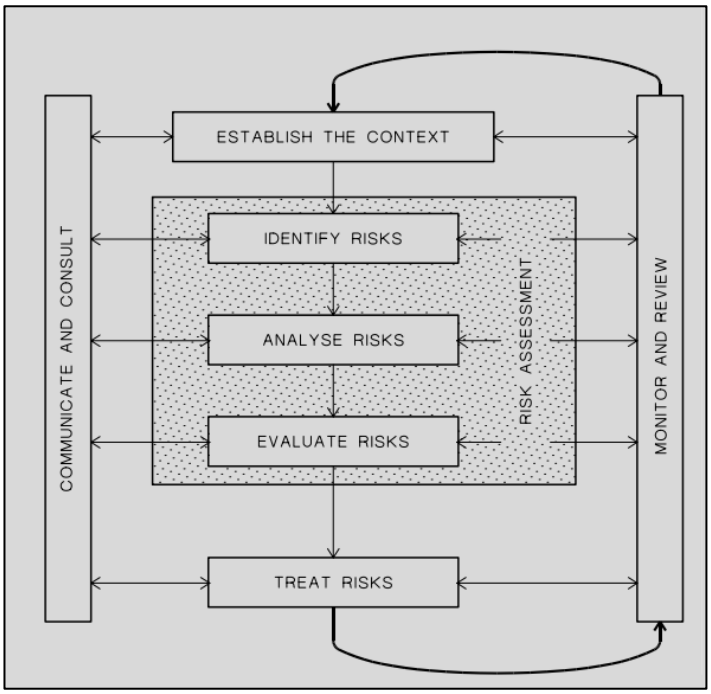


Figure 2-2: Overview of the AS/NZS 4360 risk management process (Source: Standards Australia, 2004)

2.4.4 IEEE 1540 Risk Management Standard

The IEEE 1540 Risk Management Standard describes software risk management as an important discipline for providing effective decisions and helping to ensure positive results according to their needs within software organizations (IEEE1540, 2001). The rationale for risk management according to the standard is to identify in advance possible management and technical problems so that actions can be taken that decrease or eliminate the likelihood and or impact of these problems if they are to occur.

This standard claims a process for continuous software risk management process that is suitable for all software-related engineering and management disciplines. However, according to Charette and O'Brien (1999), the IEEE 1540 is a standard principally designed for risk management within engineering. This standard collected all required and commonly used terminology required to communicate risk within engineering fields and grouped existing risk management standards in engineering to prevent redundancy. The risk management process itself is made up of several activities and tasks that function in an iterative manner. The process defines the minimum activities of a risk management process, the risk management information required to be captured, and its use in managing risk. The risk management process defined in this standard can be adapted for use at an organization level or project level, for different types and sizes

of projects, for projects in different life cycle phases, and to support diverse stakeholder perspectives.

An overview of the IEEE 1540 risk management process and its phases is depicted in Figure 2-3.

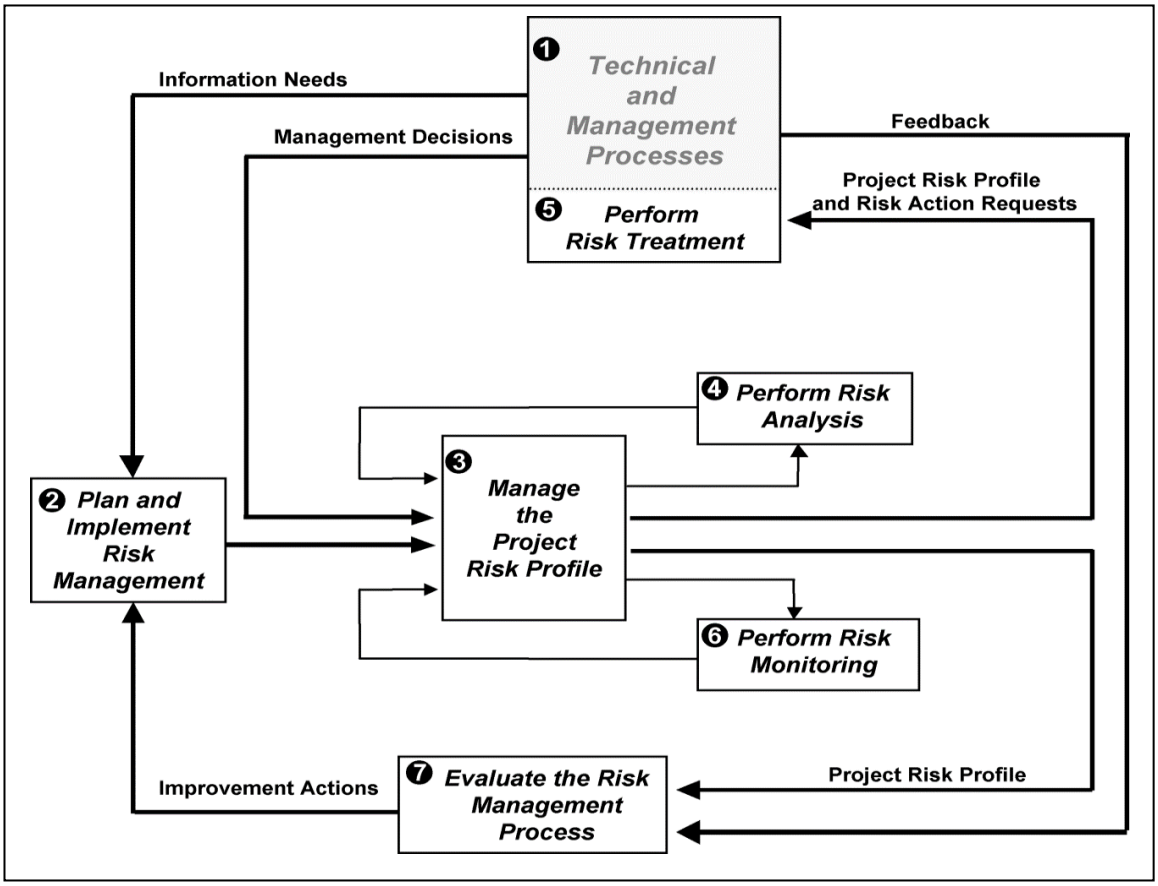


Figure 2-3: IEEE 1540 Process Model (Source: IEEE1540, 2001)

In conclusion, despite many well-known risk management models having been developed over the past years, there is a need to tailor the process to suit the situation where it is needed. A risk management model should consider looking at different perspectives and taking into consideration other software processes in order to make it more specific. This will be discussed further in the following chapter on why there should be change to the traditional risk management model. Additionally, the current risk management model should be more dynamic in order to fit into a contemporary software development model.

2.5 Introduction to Agile Development

The traditional, sequential lifecycle model including rigorous planning and upfront design as well as constant care to ensure conformity to the plan became well established since the seventies or possibly before (Royce, 1970). Alternative models were also developed such as the V-Model in 1979, the Spiral model in 1986, Rapid Application Development (RAD) in 1991 (Martin, 1991), Rational Unified Process (RUP) in 1990s (Kruchten, 2004) and more recently since 2001, Agile Methods.

Agile Methods derived from the result of the agreement of seventeen practitioners who claimed as "*lightweight methodologists*" at the time (Cohn, 2005). The agreement gathered a collection of different techniques or practices that were considered to have common ground and value the same basic principles. In the Agile Manifesto, agile methods are stated as a reaction to traditional ways of developing software and a significant transfer from heavyweight document-driven software development methodologies (http://www.agilemanifesto.org).

Schwaber and Beedle (2001) discuss that agile starts with the idea that software development is too ambiguous and unpredictable to be planned exactly and exhaustively in advance. For this reason, one must adapt the software development process to cope with uncertainties.

The manifesto refers to value statements in relation to developing software as follows:

"

- **Individual and interactions** over **processes and tools;**
- **Working software** over **comprehensive documentation;**
- **Customer collaboration** over **contract negotiation;**
- **Responding to change** over **following plan."**

The agile manifesto emphasises the agile values listed on the left, while the items listed on the right are still considered valuable. The idea of the four values is further explained in twelve principles[1] as repeated below:

[1] http://www.agilealliance.org/the-alliance/the-agile-manifesto/the-twelve-principles-of-agile-software/

“

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

Due to the nature of this research aimed at applying risk management to agile environments, it is essential to refer to these principles in order to comply with agile philosophies. In developing the proposed solution, two of the principles are particularly relevant. Firstly “*working software is the primary measure of progress*” applies in, for example, the use of Risk Burndown Chart (Cohn, 2010) in order to measure risk in days lost rather than in monetary terms. Secondly, the definition of *simplicity* as *“the art of maximising the amount of work not done*” is pertinent since it supports the idea that complex traditional risk management is less suitable for integration with agile methods.

Agile software development practices iterative and incremental development employing the fundamental principle of simplicity as one of its core ideas (Beck, 2004). Agile adopts flexibility in expanding software requirements and design and values direct

communication rather than the written documentation of the traditional plan-driven approach. Within the agile process, rapid feedback is provided to remove uncertainties. Cockburn (2005) stated that the agile model is a process that uses short speculate-collaborate-learn cycles as opposed to the linear plan-design-build approach.

One interesting agile technique which is commonly used in managing risk is the Risk Burndown Chart as introduced by John Brothers in "Agile Times" in 2004 and further elaborated on by Mike Cohn in his forum (Cohn, 2010). This is where a downward risk exposure graph is computed based on the probability and size of potential loss in every sprint (Cohn, 2010).

According to Boehm and Turner (2005), in order for a software development process to be considered agile, it should have the following characteristics.

- It should have an iterative means to deliver software in small chunks of working functionality, meaning not delivering the entire product at once, but in frequent cycles.
- It should be simple, meaning easy to put into practice according to the project needs.
- It should be adaptive meaning allowing requirements changes.
- It should be self-organising meaning that the process enables clients, developers and any other required resources to work effectively together, without constant direction.
- It should be people-oriented meaning that people are considered as the most important driver of project success.

In 2006, Ambler (2006) reported that 41% of development projects have now adopted an agile methodology, and agile techniques are being used on 65 % of projects. Later in 2008, the proportion of agile adoption increased up to 69 % (Ambler, 2008). Recently, Bustard et. al., (2013) reported the maturity of agile principle and practice for companies within Northern Ireland between years 2010 and 2012. The report indicates a growth of more than 55 % in year 2012.

Currently there are a number of agile software development methods being practiced, among the best-known being eXtreme Programming (XP) (Beck, 2004), Scrum

(Schwaber and Beedle, 2002), Lean Software Development (Poppendieck, 2007), Dynamic Systems Development Method (DSDM) (Stapleton, 1997), Feature Driven Development (FDD) (Palmer and Felsing, 2002), Adaptive Software Development (ASD) (Highsmith, 2002) and Crystal (Cockburn, 2005). According to several studies, XP and Scrum are the most widely used agile methodologies (Ambler, 2006; Fitzgerald et. al., 2006). One study reported a survey stating that XP is most extensively used followed by Scrum (Vijayasarathy and Dan, 2008) while another study reported the reverse is true (Azizyan et. al., 2011). Although these methods differ in particulars they share the iterative approach. Figure 2-4 describes a generic agile software development life cycle that features an initial planning stage, rapid iterations which contain stages of sub iterations that are repeated consecutively, and deployment stage before release below.

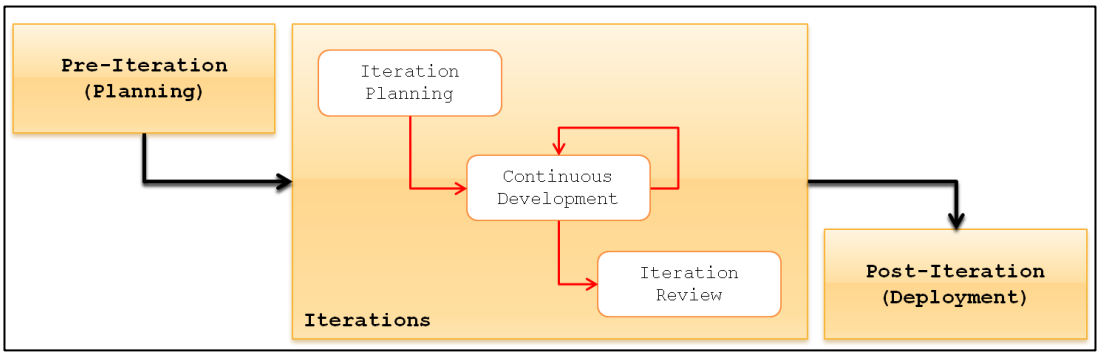


Figure 2-4: Generic Agile System Development Life Cycle[2]

The following sub sections will discuss the two most popular agile approaches: eXtreme Programming (XP) and Scrum, both of which are referred to in this research work.

2.5.1 eXtreme Programming (XP)

Extreme Programming (XP) is a software development approach that advocates rapid iterations, rigorously tested code and working closely with end users (Beck, 2004).

Beck (2004) describes XP as a "*lightweight*" methodology that dispenses with much of the usual application development process, such as lengthy requirements definition and extensive documentation. Its methods focus on addressing constraints in software development as well as adapting to vague or rapidly changing requirements. Beck and Fowler (2001) addressed the fact that XP emphasises keeping development teams small

[2] www.serena.com/docs/repository/solutions/intro-to-agile-devel.pdf

and the code simple. According to Rumpe and Schröder (2002) in their survey, XP has been largely successfully adopted in small software projects.

The fundamental idea of XP is based on its association between values, practices and principles. Values are universal and are seen as a precedent that underlie the reason behind all activities or so called practices in XP while practices refer to the method used to accomplish a project which is more on specific basis. Therefore, principles are used to guide what values and practices needed to accomplish a project (Beck, 2004).

Apart from its interrelation as above, there are other unique criteria that have supported and strengthened the XP approach. Paulk (2001) observed that the XP life cycle contains good practices which promote continual communication with customers and teams, maintaining simplicity, providing frequent feedback via testing, and dealing with problems proactively. One of its virtues includes allowing organizations to change and improve for different environments.

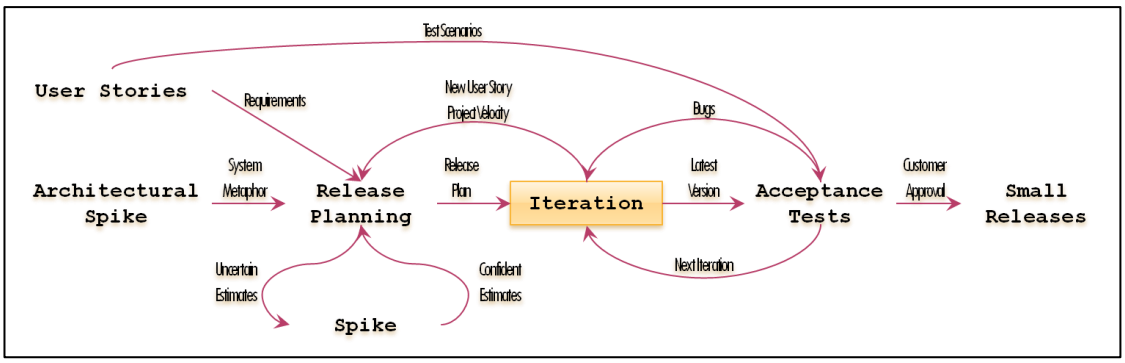


Figure 2-5: Extreme Programming Cycle (from XP, 2001)

As depicted in Figure 2-5, the XP cycle is executed when a new project is structured into short iterations, normally from two to four weeks iterations. The release planning phase refers to a session for preparation for work before the start of the iteration. During this session, the input for the project is collected. The input is composed of user stories (i) gathered from users or (ii) via architectural spikes or spike solutions. Spike solutions refer to simple prototype programs to explore potential solutions to tough technical or design problems. The spike has a purpose to reduce the risk of a technical problem or to increase the reliability of a story estimate (XP, 2001). User stories are written by customers specifying their needs for the system. Within the regular iteration planning meeting, user stories can be added or removed within this cycle.

Once planning of the iteration is completed, the iteration phase is executed which includes the development of the system. Throughout this phase, various XP techniques are applied to develop the system such as pair programming, testing and refactoring. Acceptance tests are created once user stories are specified. Series of test scenarios are implemented in order to verify requirements. The iteration ends with a formal customer approval before the developed functionality is released.

Beck (1999) presented 12 basic practices that are implemented in XP. These practices described as the 'circle of life' aiming adhere to the four values described earlier (Jeffries, 2001). However, some studies claim that not all of these practices are suitable to be applied in all contexts, the applicability depending on the size of the project meaning that some of them need to be modified to suit the project needs (Maurer and Martel, 2002; Cao et. al., 2004; Salo and Abrahamsson, 2008). These practices are explained in brief as below:

i. Planning game – includes the close collaboration between customers who provide business priorities and requirements and programmers who provide technical estimation.

ii. Small/short releases – Functionality is delivered in cycles, each cycle being very short and only providing the code for a small set of the overall required functionality.

iii. Metaphor – it is important to derive and understand the metaphor from both customers and programmers perspectives. This set of metaphors will guide all development by describing how the system works.

iv. Simple design – one of the key values in XP is simplicity therefore, simple design and implementation is preferred, whereby complex and unnecessary code is avoided.

v. Testing – testing involves both the customers and the programmers. The acceptance testing verifies the criteria that are accepted by the customer.

vi. Refactoring – it involves improving code without altering its behaviour which could include removing duplication or unused code.

vii. Pair programming – two people sitting together using one machine working on the same code. One person controls the keyboard and mouse focusing on writing

the code, while the other person inspects and considers if the functionality is being implemented in the best way.

viii. Collective ownership – any programmer can add value or change any portion of the code at any time. This gives benefits in that everyone in the team pays attention to the code in order to improve quality of the code.

ix. Continuous integration - code additions and changes are integrated with the baseline immediately or at least once a day. Therefore, the system can be integrated and rebuilt many times a day, but all tests need to be passed for the changes to be accepted into the baseline.

x. 40-hour week – the team is encouraged to work at sustainable pace. This includes restricting work hours up to maximum 40-hours a week.

xi. On-site customer – customer needs to be present and available all the time for the development team. This is because when any problem or issue arises, this can be referred immediately.

xii. Coding standards – This is to ensure that developers format code in the same way using the same style.

2.5.2 Scrum

Scrum was originally developed in Japan, where big companies like Honda, Canon and Fuji-Xerox emphasised using "*a scalable and team-based approach in order to produce world class results in product development*" (Takeuchi and Nonaka, 1986). It uses the metaphor of a rugby scrum to describe the product development game. Later in 1993, this method was extended by the combination of the concepts from iterative and incremental development as well as object-oriented development (Rubin, 2012). Scrum was first published in 1995 by Ken Schwaber at OOPSLA 1995 and from there, further publications produced by Schwaber and Sutherland (Schwaber, 1995).

Scrum shares the notion that the nature of software development involves intricate and unpredictable processes and, therefore, enforces planning at the early stage. However this can be reduced if empirical control is exercised for the process. Empirical process control consists of three important aspects; visibility, inspection, and adaptation. Visibility denotes that the outcome of the process should be noticeable and known by the person who takes charge of the process. Inspection requires frequent review of the

process so that any unsuitable changes in the process can be recognized. Adaptation is a necessity when any outcome from the process is out of limits which will effect unacceptable of the product thus acquires adjustment in order to minimize change. Hence, Schwaber and Beedle (2001) have argued that Scrum applies the ideas of industrial process control theory.

Rising and Janoff (2000) defined Scrum as a development process that basically possesses the following criteria: it is suitable for small teams; it includes short development phases called sprints which last up to four weeks. According to their report, elements in Scrum are almost the same as in Boehm's spiral model (Boehm, 1988) which involves incremental and small time-boxed iterations but includes the new element of a daily meeting – the Scrum.

Scrum also adds the important role of Scrum Master. According to Schwaber and Beedle (2001), a Scrum Master manages the sprint and is responsible for its successful outcome. It is important to select an appropriate Scrum Master to ensure that the team is working well together, impediments to progress are quickly removed and the team is moving towards the project goals (Cohn, 2010) or in other words, to coach and ensure compliance with the Scrum methods. During a sprint, the team holds short 'Daily Scrum' meetings led by the Scrum Master to discuss progress, plans and potential problems. Here, the three questions asked are: (i) What has been done since the last Scrum? (ii) What will be accomplished between now and next Scrum? and (iii) What problems got in the way of doing the work? The customer also participates in the sprint meetings, but is not allowed to influence the team in between the meetings. In Scrum, different environmental and technical variables such as time, quality, requirements, resources, technologies and tools, and the development process must be controlled constantly in order to be able to adapt to changes flexibly. This is achieved through an iterative and incremental development process (Schwaber and Beedle, 2001).

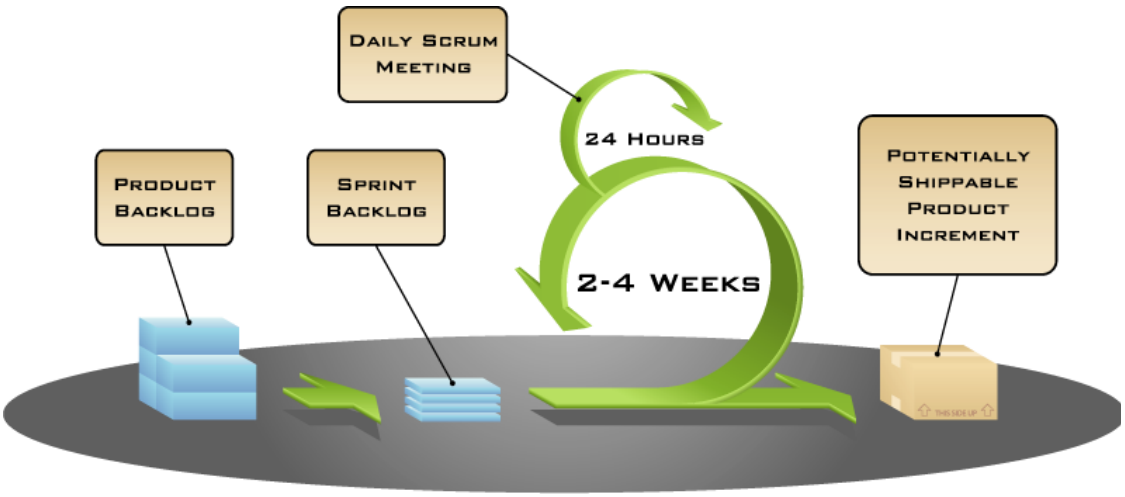


Figure 2-6: Scrum Development Process (Source: Mountain Goat Software, 2005)

Figure 2-6 depicts the Scrum Development Process. The project is started by creating the Product Backlog which contains a set of known requirements. At this stage, the items listed in Product Backlog are prioritized and divided into small time-boxed development periods called sprints. The backlog contains user stories that can be chunked into smaller tasks. The duration of each sprint can start with as little as one week and last up to four weeks. Each sprint requires sprint planning where the Product Owner and the team get together in order to specify what will be done in the next sprint. The next stage will be the development of a 'Sprint Backlog' which contains list of tasks that must be performed in order to satisfy a complete requirement by end of the sprint. During the sprint, the team take part in a 'Daily Scrum' meeting. Towards the end of the sprint, the team is required to present their results of the work implemented through a Sprint Review meeting. This is where the developed functionalities are reviewed by the stakeholders. The number of iterations for each sprint depends on the functionalities required by the Product Owner. Once all functionalities are agreed and developed as required, the Product is prepared to be released.

2.6 Software Agents

As part of the solution approach this thesis will describe a process that uses software agents. Hence a review of the relevant body of knowledge is included here. Even a cursory review of agent literature will reveal a number of definitions for software agent. It can quickly be ascertained that there is no agreed definition of 'software agent; different people define agents based on how they are going to adapt or use the agent.

One commonly quoted definition by Wooldridge and Jennings (1995) refers to software agents as "*... a hardware or (more usually) software-based computer system that enjoys the following properties:*

- *autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;*
- *social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;*
- *reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;*
- *pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative."*

Therefore, agents help to perform on behalf of other programs or people with some degree of self-determination and employ goals or rules in order to achieve some behaviour. From this it is easy to project how they could help overcome some of the barriers relating to human effort in risk management. The agents and how this can be applied in this study will be discussed further in the next chapter.

For this work, the agent development environment, JADE (Java Agent Development Environment) has been chosen as a suitable framework for development. JADE is written in Java and suitable for developing Java agents. From a software engineering point of view, JADE is valued very highly because it drives the following principles; (Fabio Bellifemine et. al., 2007)

- Interoperability – the ability of communicating with other agents within the same standards that is the FIPA[3] specifications.
- Uniformity and portability – agents are built using Java and therefore they provide a greater flexibility in term of which Java run-time environment to use.
- Easy to use – the platform contains built in APIs which made the platform easy to use and simple.

[3] FIPA (The Foundation for Intelligent Physical Agents, http://www.fipa.org/)

- Pay-as-you-go philosophy – features can be selected and used according to the programmer's needs as well as added and removed easily.

This chapter provides a historical overview of different approaches in software risk management, agile methods and also a basic overview of software agents. Evidence has been provided from the literature to show the multifaceted and complex processes in managing risk where most of the approaches present similar process models and involve heavyweight processes. On the other hand, agile methods value a more lightweight process with the notion that their methods by nature reduce risk without the need for explicit risk management. Although risks are implicitly reduced, not all risks are managed. Ignoring some risks is at best uncomfortable, but worse just one of those unnoticed risks could be project-threatening. Therefore, it is essential to further investigate these two areas and in particular how they can complement each other.

The following chapter will discuss the problem being addressed, a review of previous related work and the issues arising formulated as Research Questions.

Chapter 3 - Initial Study and Problem Statement

3.1 Introduction

Even though risk management has become one of the recognized project management practices in software engineering, it seems that not all organizations are systematically applying risk management methods (Ropponen et. al., 2000; Ribeiro et. al., 2009). Paulk (2002) confirmed that there were no specific policies or phases in agile processes stated on how to manage risk. On the other hand, there are some recent projects adopting agile methods that claim to be risk driven, typically claiming to address risk implicitly (Nyfjord and Kajko-Mattsson, 2007; Ramesh et. al., 2010).

Therefore, this chapter will emphasise the investigation into what hinders risk management implementation and what risk related issues exist with respect to agile methods in particular. Finally, there is a discussion on work related to the thesis. The investigation will then be formulated as research questions.

3.2 The Problem in Software Risk Management

Boehm (1989) has described the software risk management as consisting of two primary steps: *Risk Assessment* and *Risk Control*. Risk Assessment contains three subsidiary steps: *Risk Identification*, *Risk Analysis* and *Risk Prioritization*. Risk Control involves

risk-management planning, risk resolution and risk monitoring. Boehm's six steps are now well known and can be incorporated into the software development process. However, this is not what has been happening in many software organizations, especially when they argue that this risk management work involves extra effort, time and cost (Kontio et. al. 1998; Odzaly et. al., 2009; Kajko-Mattsson, 2009). Consequently, this work has sometimes been ignored and, and in the worst case, abandoned; some of the project managers rather rely on intuition and luck instead of managing risk (Kontio et. al. 1998).

Nonetheless, there have been many studies on risk management aimed at improving this situation and addressing the need for effective risk management. These include evolving new methods and approaches and proposing new frameworks in order to increase the project managers' awareness of the needs and importance of risk management. Padayachee (2002) addresses the view that risk management is inappropriate to individual contexts and possibly unnecessary for smaller companies. This illustrates the fact that the preceding studies on risk management established generalized results, perhaps geared to larger organizations. Hence, whenever a new risk management approach or method has successfully being tested in a certain organization or project, it is assumed that it can work well for others too. In reality, a more specific result should be established where it should categorize the risk management approach according to size, cost of software project and its organization. Kontio et. al. (1998) provides a case study for their RISKIT approach within two organizations. RISKIT is a risk management method which focuses first on a qualitative understanding of risks before their possible quantification. It then provides a defined process for conducting risk management. The study provided positive findings from the case studies carried out but also demonstrated that risk management is complex, and that people should be trained in order to use risk management methods. The provided case study was done with two large-sized organizations with no evidence given that it can be applied in small or medium-sized organizations. Gotterbarn and Rogerson (2005) introduced a risk management approach called SoDIS. Their study discussed the issue of how to address risk analysis, identification of stakeholders and the need for qualitative approaches to risk. The authors also claimed that this process was applied to software development projects in different domains on several continents, but did not specify the continents nor include the evidence to support the statement. Freimut et. al. (2001) presented results from a case study using the RISKIT Method in a large German

Telecommunication company. The results of this case study are then compared with the previous case studies and lessons learned and used to improve the method. They claim that this is a systematic approach and can be applied to other projects as well.

As more and more studies are done, it is evident that these have similar scope. Most researchers are building on the traditional risk management methods of Boehm (1989) and Charette (1989). However, some claim that risk management should be looked at from a different perspective and new methods should be introduced. Riehle (2007) describes a new approach to risk management that implements a relevant metrics program, where historical metrics can be used to anticipate future risks. This is important as in implementing risk management practices normally will depend on the memory of previous projects by the people involved, this being confounded by the fact that people can often come and go in a project. Riehle does provide a risk management approach outside the usual scope, but does not provide evidence of its effectiveness, since it is not supported by a case study. Lorenz et. al. (2005) proposed an approach to enable robust decision making in a highly distributed, multi-agent environment where agents need to act in autonomous fashion. It integrates risk management, knowledge management and an agent consideration cycle that can help in evaluating the action to be taken. The author provides a conceptual framework in presenting the idea but no further discussion on how this can work.

Dey et. al. (2007) presents a single case study for a proposed integrated framework for managing risks in software development from developers' perspective. However, the study provides no clear definition on the relationship between the proposed frameworks with the developers' point of view. Furthermore, this framework integrates with the software development cycle but the question that arises here is whether it fits and works with all types of software development processes? Kwak and Stoddard (2004) address lessons learned from implementing project risk management practices in software development environment. They state that effective risk management is achieved by changing the organizational culture to motivate the individual. The justification given for this is that cultural changes in fact require time, cost and even need repetition before they can firmly embedded into the organization. The questions arising are about how much more time is needed and how much the organization is willing to spend?

Bandyopadhyay et. al. (1999) provides a framework that can be used to guide organizations in reducing the losses resulting from the realization of threats to IT use.

However, according to the authors, managerial involvement is needed to implement risk management. Managers must be able to conceive risk from a decision theory perspective and must be encouraged to take three major steps: recognize risk with the level of application, organizational and inter-organizational; undergo training; and provide active participation.

Nienaber et. al. (2008) have reported on using software agent technology to support risk management in the software process and have argued that traditional project management methods cannot, and do not, address the added complexities inherent in current emergent environments; thus there is a need for new methods and measures to support the software project management process. Their paper provides a partial prototype, SPMSA (Software Project Management supported by Software Agents), focusing on the risk management function area but they omit a discussion of the entire implementation of their Software Project Management model.

Craft et. al. (1998) provides an Open Framework for Risk Management which defines a framework that captures various activities that can occur during the risk management life cycle. They concluded that automation support will be a necessity, given the workload involved in managing risk.

A study carried out by Gemmer (1997) has stated that successful risk management involves three elements:

- A repeatable process of risk management activities;
- Widespread access to adequate knowledge and an understanding of functional behaviour which basically deals with human perceptions; and
- Communication and interaction.

This adds to the evidence presented earlier and leads to a conclusion that in order to apply risk management in software development projects there are two major problem areas:

1. The extent of human effort associated with risk management

Referring to the Boehm's tutorial (1989) on risk management activities as depicted in Figure 3.1, there is potentially a lot of complex and monotonous work to be done by the relevant stakeholders in the software project. Risk assessment can be subdivided into

risk identification, risk analysis, and risk prioritization each of which require the identification of one or more suitable techniques. The same comment can be made for the risk control activities of risk management planning, risk resolution and risk monitoring. Thus, risk management has multifaceted steps, continuously repeated over and over again in different projects at inevitably high cost. Moreover, often this effort is wasted due to the often frequently changing nature of software.

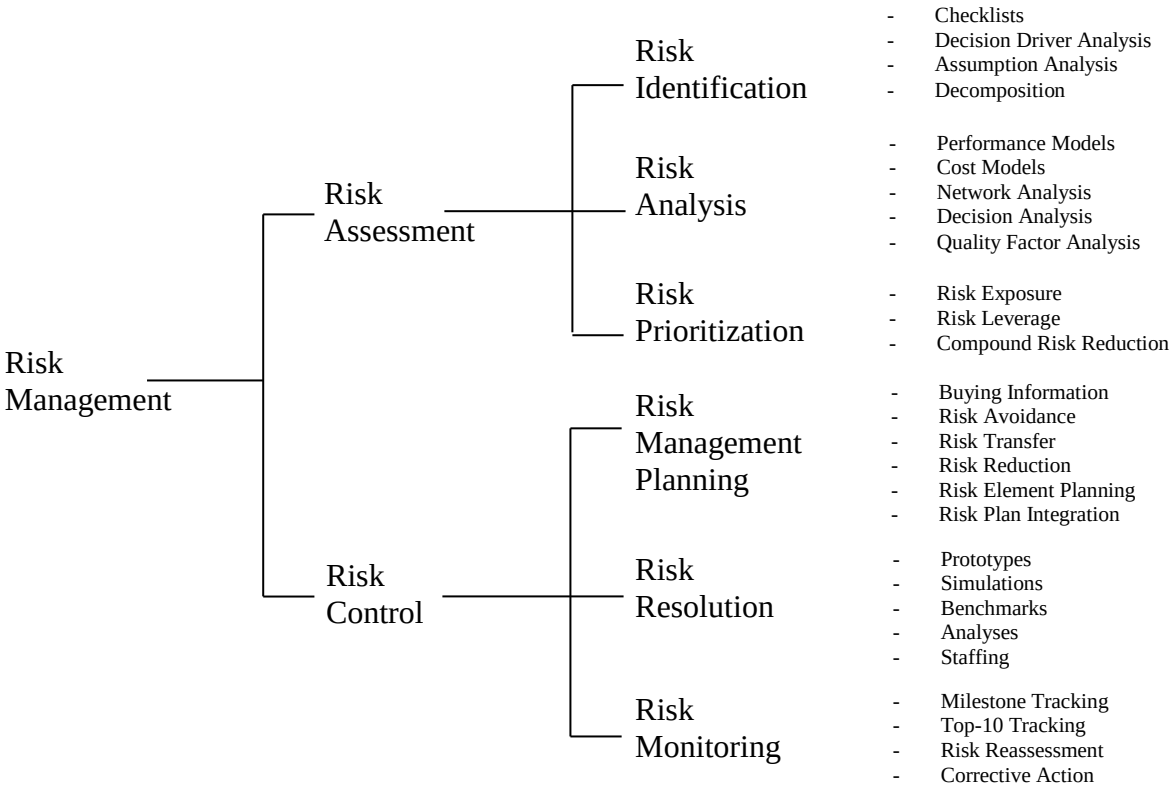


Figure 3-1: Software Risk Management Steps (Boehm, 1989)

2. Organizational culture and human behaviour factors

There have been many studies outlining the problems with organizational culture and human behaviour when it comes to the implementation of the risk management activities in their software projects (e.g. Bandyopadhyay et. al., 1999; Kwak and Stoddard, 2004). Managing risk activities is not easy. Most managers found that despite understanding the risk and the importance of risk management to software projects, being knowledgeable in the area and having to invest effort in the project partly contributes to optimistic behaviour and ignorance of risk. This can be summed up through the well-known remark by Fred Brooks (1986) that *"all programmers are optimists"*.

Proper implementation and use of the risk management activities in developing software projects requires resources, good knowledge, considerable effort and time management. Because of this there is an associated cost, a cost which many are not willing to incur but at the same time those organisations may not foresee the danger of ignoring risk Therefore, a means to manage risk continuously and effectively, but with reduced human effort, is desirable. The proposed solution for this thesis is to employ dynamic autonomous software agents to help to manage and ease the risk management work. This will help to save more time and allow developers to concentrate on development activities in which they are expert.

Based on the evidence in the literature, clearly the major obstacles in implementing risk management are related to human factors, specifically the project team or the stakeholders' involvement in implementing risk management. As evidenced from the literature, many studies show that in order to successfully implement current risk management methods, participation and effort from the stakeholder or project team is important as well. This aspect has been neglected to date and there is a need for new methods for risk management that assist in solving this problem in software development projects.

3.3 Initial Study on Risk Management Practices and Barriers

The initial study discussed in this sub section has been published as part of the results published in 2009 (Odzaly et.al, 2009).

Seaman (1999) argued in her paper that many issues arising in software engineering are best investigated using methods which integrate qualitative study and quantitative data. As a first step in any such study leading to development of a new solution approach, there is a need to investigate the current state of the art. Therefore, in order to address this, an initial qualitative study was developed to investigate practices in use and perceived barriers in the application of risk management.

Kontio and Basili (1997) argue that the three main obstacles to wider use of risk management technology are a low awareness of the technology, the limitations of existing risk management approaches, and a lack of empirical evidence for the usefulness of risk management methods. Given that this view is from more than a decade ago, there is a need to investigate current views about Software Risk

Management and supporting technologies. To this end a questionnaire was developed as an instrument to investigate current Software Risk Management methods and tools being used by the software companies (see Appendix A).

Prior to the distribution of the questionnaire, the study was planned so as to meet the ethical requirements of the School of Electronics, Electrical Engineering and Computer Science, Queens University Belfast. Along with the distributed questionnaire, a cover letter was included in order to explain the nature and purpose of the study. It was also explained that the return of the questionnaire indicated the respondent's consent to participate in the study but that their responses would be held in strict confidence, not to be shared with anyone outside. To ensure uninhibited answers and also to remain ethically sound, companies were informed that no individual responses would be attributed to any person or company in any resulting publication.

For convenience the survey was carried out in Northern Ireland. Here a small part of the findings are presented, but enough to indicate the need in the sample for an improved approach to software risk management, and to justify the proposed research.

The sample used was based on a combination of contact lists for ICT companies from InvestNI, the regional economic development agency and from Momentum, the local ICT trade association. To avoid dormant companies and those in start-up mode the list of contacts was screened and companies with a total number of employees less than 10 were excluded. This cut-off value was chosen so as to avoid companies that were very immature and by implication less likely to have fully established or stabilized software processes or management techniques. The original list of companies also included those no longer involved in software development and these were discarded. The total final population consisted of 89 companies and out of these, 18 responded.

86% of respondents agreed that they are aware of the practices of Software Risk Management. This counters the earlier assertion by Kontio and Basili of a low awareness of risk management technology being the main obstacle to its practice. In fact, the survey also showed that 80% of the respondents have or previously had a process or methodology for Software Risk Management. However, the survey also indicated little common ground among companies regarding the types of risk management methodologies or tools being used and that there is no standard process for risk management. This finding has been supported in Rabbi and Mannan (2008) who, using a comparative analysis showed that there is no single tool or technique suitable

for managing risks in every software development situation. In the sample, a few of the respondents were using the Prince2 (PRINCE2, 2002) project management methodology which includes risk management practices and others relied on a perceived implicit risk control such as via adherence to CMMI/ISO9001. Most companies however claimed to have their own internal methodology derived or in some cases using an in-house developed risk management tool employed, as needed per software project. The survey asked about the individual risk management activities. 40% agreed that Risk Identification was the most effort intensive process. This echoes the statement made by Charette (1996) that stated that project managers somehow do "acts of omission" to perform formal risk identification because most of them do not know how to do it, or because it is difficult to do it. Acts of omission are actions that are not performed due to ignorance. However, 30% stated that Risk Monitoring is most difficult and needs the most effort. One opinion returned in the survey stated that as the project approaches its end, requirements have changed and tracking a risk and its dependencies becomes tedious. A similar point made was that as the project deadline is approaching, team members tend to disregard risk monitoring activities in favour of work perceived as more urgent.

Respondents were asked to comment directly about the most complicated step or process in the Software Risk Management. Some of them stated that Risk Monitoring can be difficult where it involves tracking risks across large projects with multiple teams. Others commented that it is rather difficult when it is treated as a separate activity and that it adds management complexity to the project. Besides this having the discipline to continually review and manage risks was quoted in one case as the biggest challenge. Other comments include that project management pressures put the focus on schedule, quality and budgetary control rather than risk management. Some reference was also made to the perception that agile methods appear to be lightweight and ignore risk management. Some of the respondents state that the identification of probability and impact can be difficult as both are highly subjective. A further point raised was that different contributors to the analysis process may have very different views, so that reaching a consensus can prove to be difficult. Some risks also will be generic to most software development. Thus, it is quite tricky and time consuming to specify the risks which are most related to the project.

Other than Risk Identification and Monitoring, a respondent mentioned that there is no point in only putting into practice any elements of Software Risk Management because

it is about doing all or nothing, where partially implementing will not be successful. Their viewpoint was that it is pointless in identifying and analysing risks if monitoring and resolution steps are not going to be implemented. One response mentioned the difficulty in just getting a process in place and persuading people to use it correctly in order to realize the benefits. Once the process is operating and people are happy to do it, then it is not considered complicated anymore. As far as agile applications are concerned, a response from a company applying an agile approach stated that they do not really have a specific tool or methodology of Software Risk Management but instead they do what they feel is right, rather than use a specific methodology. However in their opinion perhaps it would be better to have one. Based on this, it is apparent that there is a need to derive a sound software risk management application and possibly extend it to adapt to operate in an agile environment. Any tool should aim to overcome the situation where software companies tend to focus on development rather than properly applying software risk management. Furthermore, Bannerman (2008) argued that one of the challenges facing by the practice of risk management in organizations are that they are driven by demonstrable outcomes, usually performance. Thus when the project is successful, it is difficult to demonstrate any part of the outcome pertaining to risk management practices.

The survey attempted to establish possible barriers of applying Risk Management. As a basis the literature was searched for already discovered risk management barriers. Dedolph (2003) lists a number of barriers to risk management but offers little or no empirical analysis of their individual relevance or importance. These barriers were used as a basis for asking respondents in the survey about risk. Respondents were asked to rank the barriers from 1 (most agreed) to 9 (least agreed). The final percentage score for each barrier was derived using the calculation $(100 / t)\ \Sigma s_i$, where s_i is the score from each respondent ($s_i = 9 - \text{rank} + 1$), and t is the total of all scores. The results are shown on the graph and table in Figure 3-2.

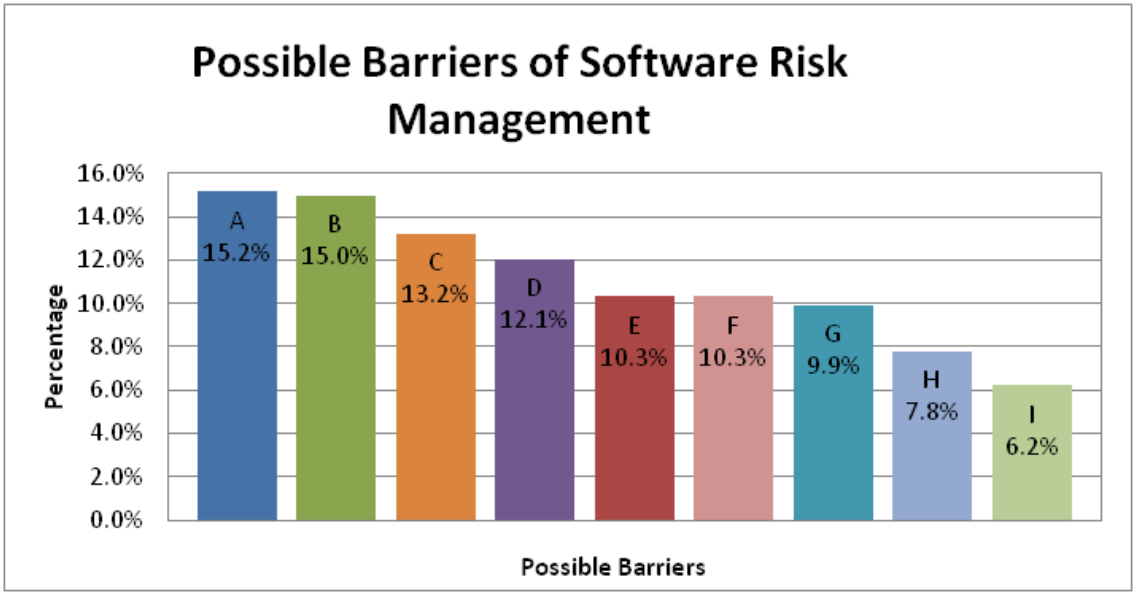


Figure 3-2: Possible Barriers of Software Risk Management

Legend	Possible Barriers	Percentage of Responses
A	Visible (and tangible) development costs get more attention than intangibles like loss of net profit and downstream liability.	15.2%
B	There are no resources available for SRM.	15.0%
C	Mitigation actions may require organizational or process changes.	13.2%
D	Risk management seems difficult or there are too many risks to handle.	12.1%
E	The value of risk management cannot easily be proved.	10.3%
F	Project teams (and managers) see reward for problem-solving, not prevention	10.3%
G	Overconfidence (e.g. risks are already taken care of, implicitly).	9.9%
H	Discussing risks goes against cultural norms (e.g., bringing up potential issues is viewed as negative thinking or as causing conflict within the group.	7.8%
I	Fatalism (e.g., software is always late anyway; there is no way to change that).	6.2%

Out of these nine candidate barriers, 15.2% agreed that the biggest barrier to Software Risk Management being adopted is that the visible and tangible development costs gets more attention than intangibles like loss of net profit and downstream liability, while 15.0% selected the main barrier as being that there are no resources available for Software Risk Management. The third most popular choice was that mitigation actions may require organizational or process change. This may be somehow indirectly related

to the first two barriers; where companies are not willing to invest in activities not directly related to the software development itself. Meanwhile, the lowest barrier as ranked above; contributes only a small portion of 6.2% of respondents believes that fatalism as one of the possible barriers to conducting Software Risk Management. In fact items A to F, the six highest scored barriers can all be related to a consideration of the value of risk management activities compared to the cost of carrying them out. Meanwhile, items G to I are concerned with attitude and human behaviour. Therefore, the biggest problems seem to concern with the value:cost ratio for risk management. Since studies in the literature already demonstrate the value of risk management, it makes sense to try to reduce the cost of risk management, being mainly related to the human effort required to implement risk management activities.

At the end of the survey, views were elicited from the respondents and some have contributed some ideas and comments on future Software Risk Management possibilities. Among the ideas are the need for a risk register and an automated dashboard for monitoring risk and to help minimize the human effort. Others mentioned, based on experience from previous failed projects, the need to train the stakeholders of the project about the awareness, importance and usefulness of practicing Software Risk Management in software development projects.

The survey presented has limitations. As such, the following validity issues established from the conducted survey are discussed:

- External Validity - Questionnaires are known for low response rates and the return of 18/89 means that the evidence is not conclusive, but indicative. The sample is also related to a single geographical territory so that there is the possibility of a cultural bias. The sample was, however, mixed in terms of indigenous and foreign owned companies. The company sizes ranged from 16 to 100,000 employees.

- Construct Validity - In maximizing the time given to answering the questionnaire, the author did not attempt to test respondent knowledge, but based on their stated experience levels assumed that the respondent understood the terms being used. Thus the threat of the questions being the wrong ones to ask was mitigated. Each question left space for elaboration, and when asked for comment on the questionnaire, only favourable responses were received.

- Internal Validity - The author used generic contact lists obtained from the local investment agency and trade association along with emails provided therein. It was specified that the questionnaires should be passed on to an experienced project manager, and assume that this was carried out faithfully. As a check, one of the questions asked specifically about job role and experience and received responses of either “project manager” or “project leader”. The mean number of years of experience they had in this role was 8.8 years, and the range was 2 to 20 years. 12 of the respondents had more than 5 years of experience in a project managing role and 8 had more than 10 years.

- Conclusion Validity - The research provides a mixture of quantitative and qualitative data. Given that it is a small sample and from one region of the United Kingdom, the results cannot be assumed to be generically applicable worldwide. The sample is of experienced project managers; the appropriateness of extending the conclusions to others is left to the reader.

As a conclusion, based on the result from the collected responses from companies in Northern Ireland, it would seem that there is a large gap between the theory and the application of Software Risk Management. Looking at the feedback, it is evident that Software Risk Management is not being fully applied and the major problem seems to be the extent of human effort required and either this not being feasible or not seen as worthwhile. The challenge therefore is not necessarily to add value to risk management but to decrease the effort required to carry out risk management. In this case risk management would be perceived as more worthwhile.

3.4 The Application of Risk Management in Agile Projects

Despite the fact that risk management has been with us for some time, little has been reported about its industrial status, its co-existence with the various development models, and its compliance to standard risk management process models (Kajko-Mattsson and Nyfjord, 2008). In Kajko-Mattsson and Nyfjord’s (2008) case study regarding the state of industrial risk management practice, they have explored the domain of risk management practice within 37 software organizations. The results show that there are some discrepancies between the industrial practice and the standard models studied. They state that industrial organizations have not implemented all the

important activities as prescribed by the standard models. As questioned on the applicability of risk management in the context of agile, sixteen companies reported that risk management is needed in any development model including agile, eleven companies stated that risk management can be partly applicable, four companies claimed that risk management is not useful in agile projects due to its complexity and that iterative agile models include risk management by nature, while six companies did not respond to this question. According to Schwaber and Beedle (2002), agile practices and specifically Scrum advocate a new paradigm of software development that simultaneously provides risk reducing practices. They claim that agile methods indirectly apply risk management to their processes. They also suggest a list of issues that need be addressed in both the standards and the industry, and most importantly propose integrating risk management tools with the organizational tools. The study has concluded that risk management is needed in any development model, whether traditional, agile or other. Descriptions of agile methods provide only very general guidance for managing risks (Nyfjord and Kajko-Mattsson, 2007). Risk management models, on the other hand, provide detailed guidance, but nonetheless their implementation has often been neglected or ignored. If both approaches are considered vital the integration of them is essential and the means to do this has to be found. Integration of agile methods with sufficient risk analysis and control will make sure that risks are handled and not ignored.

Marcal et. al. (2007) has challenged the risk driven concept that is claimed to be implicitly applied in agile projects. Their paper presents a mapping between the Capability Maturity Model Integration (CMMI) model and Scrum. Based on the comparison between the two processes, the paper concludes that Scrum does not cover all specific practices in project management process area. Specifically mapping the process area of risk management, they found that there are some practices relating to this area like risk identification, assessment, monitoring and response that are either being not addressed or are only partially addressed by Scrum. Although claims are made that agile methods have inbuilt risk management, the evidence seems to suggest that they do not when considered against the criteria defined in the CMMI (Bannerman, 2008).

Based on the investigation discussed above, it is evident that there is a gap in the application of risk management in agile projects. Even though the agile process reduces risk to cost and schedule by using short and iterative cycles, this does not mean that all

risks are negligible. The following section will discuss the possible issues in agile projects which, if not treated properly, can manifest as threats to projects.

3.5 Risk Issues in Agile Projects

Due to the fact that agile methods depend a lot on the credibility of the people involved in the projects (Cockburn and Highsmith, 2001; Nerur et. al., 2005) as well as their motivation in applying the agile practices (Layman et. al., 2006; Conboy et. al., 2010), most issues encountered relate to the people and the practices involved. This echoes one of the values in agile manifesto i.e. "*individuals and interactions over processes and tools*" (Agile Manifesto, 2001). This implies that not having the right people doing the right process will be a source of risk.

Cho (2008) developed some research work on issues and challenges of agile software development with Scrum. The research work mainly aims to provide guidelines to assist the companies to avoid and overcome barriers in adopting the method. An in-depth study was conducted between two different companies using various qualitative data collection methods. The study presented various common categories of issues and challenges in agile projects, among which the following points are discussed in Table 3-1.

Chakradhar (2009) has outlined in a white paper the common pitfalls in agile projects in which, he proposed that it is vital for the project manager to understand those pitfalls in order to reduce risks. Among the important aspects discussed are: failure to provide sufficient training in agile methodologies; unfamiliarity of the project manager with agile methods; poor involvement from the Product Owner; the team practicing 'single expert' with no knowledge sharing as well as having passive team members.

Cockburn and Highsmith (2001) highlighted that one of the most important success factors in a project is individual competency. They emphasize the qualities of people involved in the project, where good people will complete the project while if team members do not have sufficient skill, no process can compensate for their deficiency. This is also supported by Boehm and Turner (2005) where people issues are the most critical and it is very important to address them before adopting and integrating agile practices into a project. The paper presents a list of barriers to using agile methods successfully and among the significant issues highlighted with respect to people are

their roles, responsibilities and skills as well as their ability to predict and be knowledgeable.

Table 3-1: Categories of issues and challenges found in agile projects (Cho, 2008)

Categories	Issues Found
Human Resource	• Formation of team, where team was being organized without considering their necessary skills and knowledge • Multiple responsibilities where one team member is responsible for many tasks • Some developers are not aware of the benefits of applying agile methods, and so are reluctant to apply agile practices • Lack of accountability where team members do not take responsibility for any delayed task coupled with a lack of supervision • Collaboration between team members is difficult, especially when they are not located together.
Structured Development Process	• Scrum meetings; Daily Scrum, Sprint Planning and Sprint Review meetings are sometimes inefficient where they are being held too often, or taking up too much time or setting up the meeting is difficult. • Difficulties in estimating project work on legacy code
Environmental	• Poor customer involvement and unclear product requirements • Individual contribution is not recognized and no guidelines exist in determining accurate measurement of individual performance.
Information Systems and Technology	• A lack of communication between team members that are co-located causes duplication in resolving problems. • Newly hired team members tend to create code errors due to unfamiliarity with the software.

Deemer and Benefield (2006), discuss common challenges in Scrum. One of the challenges put forward is the ability of a team to provide estimation of effort in their development work especially when it is done for the first time. Most teams fail to deliver the tasks committed to due to poor task analysis and estimation skills. When this happens, the team tends to extend the duration of the sprint rather than learn to do the correct estimation. This can cause problems in achieving a sustainable pace since the team will not be able to work reasonably due to delay in completing other tasks in the project.

There are also many studies addressing issues with agile skills and personnel turnover as well as a result of job dissatisfaction (Boehm and Turner, 2003; Melnik and Maurer, 2006; Melo et. al., 2011). Individual motivation is important in Scrum as this leads to the team adhering to agile process practices, for example attending Daily Scrum Meetings (Hossain et. al. 2009). Team member behaviour where teams fail to comply with practices can provide early signs of risks e.g. low morale expressed during the daily meeting or avoiding discussing problems when behind schedule (Lindvall et. al., 2002).

Having a team member that is an agile sceptic, meaning, they are opposed to agile methods, can have a huge impact to the team as a whole. This is due to the fact that an agile team relies heavily on trust and sharing of tacit knowledge to support important practices like pair programming and shared ownership (Boehm and Turner, 2005). Melo et. al. (2011) presents an unusual result with regard to the relationship of pair programming to tasks and motivation. Surprisingly, whether the tasks are too easy or too complex, may influence the motivation to work in pairs.

Another important practice in agile process is the collective code ownership. The study results discussed in Layman et. al. (2006) indicates that collective code ownership provides benefits in terms of knowledge sharing within the team. However, the disadvantage of this is that the team will tend to choose the fastest solution ignoring its quality, assuming that they are not the only one who is responsible for the quality or otherwise of the code. By not assigning task ownership for a piece of work, this could demotivate the team in writing code that conforms to the standards or required quality levels. Other problems discovered are working at a sustainable pace and accuracy of estimation. These problems are due to the situations like having aggressive datelines, having to work overtime as well as underestimation of completion of time.

Nelson et. al. (2008) introduced a risk management technique that can be adopted in agile processes. The paper also provides an argument in relation to agile being risk driven in that it implicitly manages risk in the process. One of the implicit techniques used is to prioritize tasks at the start of iteration. However, simply placing higher priority to a high risk task is not considered as managing risk. It does reduce the risk to the project if the associated task is executed earlier, but until the risk is resolved or no longer applicable, the task needs to be monitored for the risk, and action taken if necessary. When identifying the right risk for the task and analysing it is not done properly, presenting an appropriate plan to mitigate the risk is difficult (Williams et. al., 1997; Ahmed et. al., 2007).

3.6 Related Work

The aim of this research specifically is to resolve the research problem by building a novel and reliable model of risk management in agile projects and to support this with the necessary data collection methods and tool. The research work proposed the development of the new approach model and tool support called the Agile Risk Tool (ART). The model provides significant characteristics that are embedded in the ART process model which includes the definition of project or sprint goals and decomposing the goals into problem scenarios that can be threat to the project. The novelty in the ART model and tool support is that together they provide the means to capture data from a project environment using software agents and use this for risk management, thus obviating some of the issues causing non-performance of risk management, chiefly the issue of the excessive effort required in capturing risks.

To date, to the author's knowledge and understanding, no similar work has been done before, especially that aimed at reduction of the human effort in risk management and to provide as much autonomy as possible. This research contributes to three significant areas in software engineering; risk management, agile development and software agents.

Coyle and Conboy (2009) develop a case study investigating to what extent risk management practices are applied in agile projects. The motivation for the study was supported by the increasing failure in projects where the project manager fails to detect risks and where they found little evidence related to risk management in Information

System Development (ISD) projects. The case study was carried out at one firm incorporating the DSDM method (Stapleton, 1997) with composition of risk elements like risk identification, estimation and evaluation. The study concludes that although little evidence was found in the literature relating to risk management in agile methods the application of risk management is, however, still valued.

Machado and Pereira (2012) describe an approach that focuses on the risk identification phase of risk management. They argue that risk identification is a very hard prediction problem and getting a project manager to identify risk is crucial. The main idea is to get the correct identification of the risks, since that all remaining phases in risk management depend on this. Their study proposes the creation of an expert system which is capable of identifying risks in software projects by using lessons inductively learned from similar software projects already developed. They describe an expert system that uses a knowledge base as a decision tree containing a set of risks, an inference engine to select the risk and provide reasoning on whether a risk may or may not occur based on the rules encoded in the knowledge base. The experiments used data on real software projects in order to validate the effectiveness of the implemented system. The results of the study show that the lesson learned obtained from previous projects can be used to automatically identify risks in the new projects and avoid mistakes of the past. Nevertheless, in the study the authors emphasize that the approach is suitable for risk identification tasks but that the human expert is still crucial.

Lamersdorf et. al. (2011) proposes a rule based model for a customized risk identification and evaluation for task allocation of in distributed software development projects. This approach focuses on identifying risk at the start of the project which proposes a model to evaluate individual risks specifically in a project. The model uses a set of logical rules that captures experiences from past projects which includes describing project characteristics that can influence risks in a distributed development project. The study was validated based on the interviews with the 19 software practitioners at a particular company, “Indra Software Labs”. The results indicated that the model was able to make predictions associated with experiences in past projects and that it managed to gain full support by highly experienced managers.

Nyfjord and Kajko-Mattson (2008) propose a model which combines agile process with risk management. The model contributes to two significant elements: an integration model that serves as a guideline to incorporate risk management in agile processes.

They have developed a model of agile risk management that they claim can fit with an agile process. The model was evaluated by ten agile practitioners in Swedish software industry using qualitative methods. As a result from the evaluation, majority of the practitioners provide positive feedback and agree that (i) the integration model provided useful guidelines on how to integrate risk management in agile projects, (ii) the integrated model provided a useful reference model in order to apply explicit risk management in their projects and (iii) the proposed solutions maintain agility where risk management is added while the agile process remain unchanged. The results have led to three main conclusions where (i) the model provides a valid solution for the lack of risk management in agile development which only applies in certain projects, (ii) there is a need of further elaboration for the provided outline and (iii) they admit the need to further investigate the process in terms of its validity in practice.

Masticola (2007) derived the ideas of lightweight risk mitigation for software projects from a complex real time system called "Pocahontas". The approach proposes three important activities in lightweight mitigation which includes data collection through database mining, decision support and remediation support in order to cater for problems with code quality. Using these activities, the author focuses on detecting "bad smells" in the source code using an automated code inspection tool which is embedded in "Pocahontas". Then, the author uses set of scenarios to support decision making in evaluating resources. Lastly, the author proposes development of a generic refactoring tool as remediation support. As a result of achievement from these activities, cost and effort of implementing risk mitigation can be effectively reduced. Nevertheless, the paper does not include any results from validating the approach.

Nienaber and Barnard (2007) designed a generic agent framework to support key areas in software project management (SPM) processes. The paper uses a risk management process to illustrate the example on the overall agent framework. The model was extended into two conceptual models that depicted the core and facilitating functions to support SPM. The core functions include time management, cost management, quality management and scope management. The facilitating functions include communication management, risk management, procurement management and human resource management. The author concludes that based on the core and facilitating models, generic agents framework can be designed in to address common tasks for all key functions. This paper however does not provide any results in validating the approach.

Ramamoorthy et. al. (1993) presents a technique and architecture of a knowledge based support system for risk assessment. The paper mainly discusses the importance of risk assessment for software development and reuse as well as the key steps in the risk assessment process. They also discussed a knowledge based tool called the ATMS - Assumption-based Truth Maintenance System. They claim that ATMS helps in maintaining consistency among software entities in software life cycle. In addition, they claim that using this tool, a more efficient and accurate risk assessment can be achieved.

The following section outlines the Research Questions based on the gaps in existing work discussed above and will provide direction for this research work.

3.7 Research Questions

Kitchenham et. al., (2002) state that the research questions of an investigation are stated if the research involves exploratory studies whilst hypotheses refer to a statement for which tests can be performed to confirm or refute a defined theory. This research study started with the investigation of the problems manifested in two areas; software risk management and agile methods. This is followed by proposal of the solution approach and its validation. Therefore, research questions will be used as a basis for the research rather than specific hypotheses.

Research Question RQ1: Risk Management Barriers and Issues in Agile Projects

RQ1a: Is risk management being performed in industry; if not what are the barriers?

RQ1b: What are the possible risk factors that can be identified and managed throughout the agile projects?

RQ1c: Do the findings of the application of risk management in agile projects give useful insights towards an improved agile process and risk management?

Research Question RQ2: Software Engineering (SE) Environment Data

RQ2a: Is it possible to support risk management using existing SE environment data to overcome barriers in RQ1a? And if it is possible, which data is it feasible to use?

RQ2b: Can data collection be conducted with minimal intrusion and effort?

Research Question RQ3: Software Agents and Rule Engine

RQ3a: Can software agents coupled with a rule engine provide a means to automate risk management in agile projects using data from the Software Development environment?

RQ3b: Is an approach using Agents operating in the Software Development Environment useful?

The following Research questions have been addressed as follows:

RQ1a in Section 3.3, RQ1b in Section 3.5 and RQ1c in Section 7.4;

RQ2a in Section 4.2.2 and RQ2b in Section 5.2;

RQ3a and 3b throughout Chapter 4, particularly in Section 4.1 and 4.3.

The Research Questions will be further discussed and answers discussed in Chapter 8.

Chapter 4 - Solution Approach

4.1 Introduction

Based on the research problems discussed in the previous chapter, there is a strong motivation to improve the management of risk in agile projects. Given the nature of agile methods, there is a need for a lightweight risk management approach which can add the necessary risk management weight without unduly threatening the agility of projects. In reality, contemporary risk management should be looked at from a different perspective where it should be integrated as part of the agile processes and decision making. This includes taking into account human factors such as developer's skills and ability as well as their behaviour in performing their allocated tasks. As demonstrated in Misra et. al. (2009) when identifying important success factors in adopting agile practices, one of the factors that have significant relationship with success was the personal characteristics of the team members. Among the most important are taking responsibility, willingness to work in pairs, having a basis of sound principles, and being passionate about their work. Further, risk management in agile projects needs to be implicitly and explicitly addressed, but should also be dynamically responsive to changes in the product and project environment.

Due to the fact that there is already available data collected in modern software project management tools, the proposal is to take advantage of this in order to inform and improve risk management. Since the initial investigation concluded that the extent of

human effort associated with risk management is a major barrier to its implementation, being able to automatically collect and make use of this data could be used to reduce the effort otherwise needed to manage risk. Since people, processes and tools in the software process represent an environment from which data can be picked up and acted on as it changes, this leads to a consideration of the use of agents in any potential solution. Given that the earlier definition of agent included *reactivity* where agents perceive their environment and *pro-activeness*, where agents exhibit goal-directed behaviour in response to environment changes, one possible solution to the problem of excessive human effort in risk management is one where agents take responsibility for acting on data that indicates a change in risk exposure.

In the following sub sections, the underlying ART model is explained, and in particular its use of agents. The discussion will include the overview of the Agile Risk Tool (ART) model followed by the ART architecture and process model.

4.2 The Development of the ART Model

The development of the ART model started with the establishment of a view of how risk management may apply in an agile environment. Figure 4-1 below depicts an overview of the resulting model.

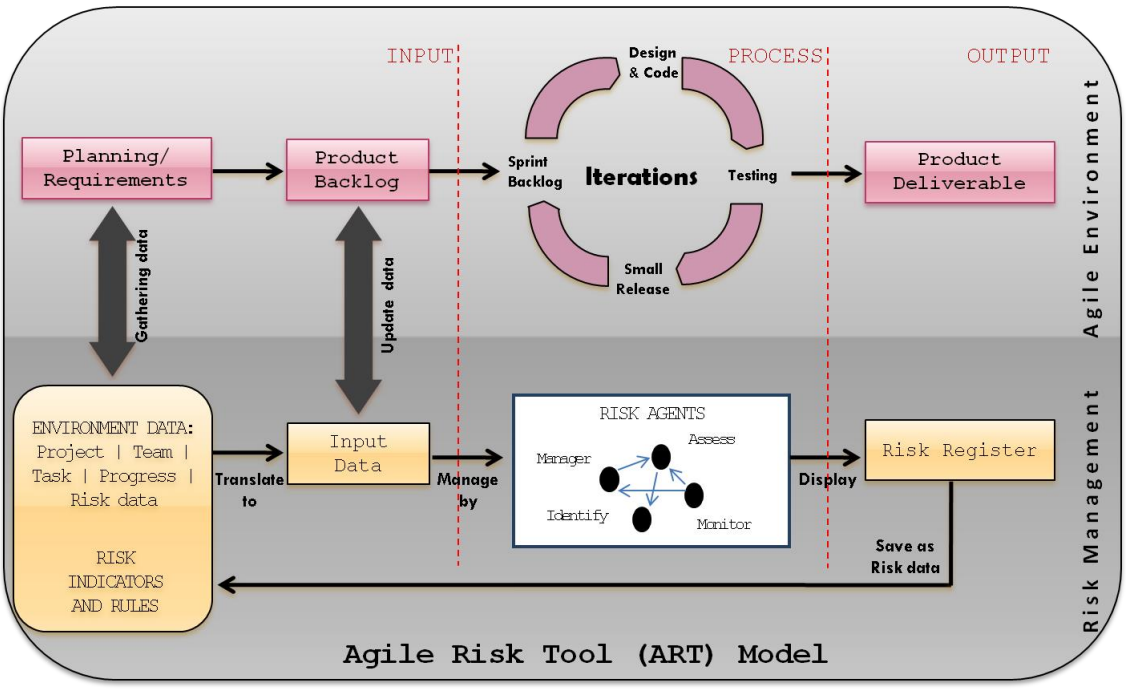


Figure 4-1: Agile Risk Tool (ART) Model describing the application of Risk Management in Agile environment

The model represents how risks are gathered and managed throughout the agile project. During the Input stage, the agile process begins with planning and requirements gathering. At this stage, while preparing the project, at the same time, the gathering of risk data can commence.

Requirements in agile processes are most often represented as user stories. These are textual descriptions that contain the customer's specification of needs for the required system. A *product backlog* is a subset of these requirements that will be selected from based on priority.

The environment data used contains:

- A *project* in this context is a set of user stories, the membership of which is not fixed at any point of its lifetime. Each project relates the unique project name of the project, a set of goals for the project, when it started and when it ended.

- A *team* is a set of persons where each person consists of a set of attributes describing the person. Each team is working to achieve the goals of the project. For each team member there is specific information, for example on the type of skills that the team member possesses and also their levels of expertise in defined skills, stated as an integer;

- User stories are divided into tasks. A *task* refers to a textual description of the task associated with the estimated hours of completion, the name of the person responsible for the task and the progress for the task;

- *Progress* refers to additional information on the progress of a specific task as reported by the person responsible for the task. This includes information on attendance of the team member in the Daily Scrum Meeting and whether progress or an impediment is reported for the task;

- *Risk data* represents information on risk captured by the tool. The information includes the name of the risk, its severity, the owner of the risk, location of the risk as well as the date the risk is triggered and resolved.

The *risk indicators and rules* refer to a set of predefined risk factors brainstormed by the team at the early stage of the project and encoded as rules (discussed later). The risk indicators contain a textual description indicating a threshold or state that will trigger the risk. One example might be where a high priority task is selected in the sprint by a

developer with too low a predefined skill threshold. Rules contain a list of conditions for an event encoded into IF/THEN statements. Later, this information is stored in the rule engine. *Input data* refers to a set of collected data from the environment and translated into a set of templates readable by the tool.

During the Process stage, the project proceeds as iterations which include sprint backlogs, design and code, testing and small releases of the product requirement. *Iterations* contain are time-boxed into fixed length durations of development. A *sprint backlog* refers to a list of tasks that must be performed in order to satisfy a complete requirement in a sprint. *Design and code* is a phase where the team start the development of the selected tasks while *Testing* phase will verifies the criteria of each requirement. *Small releases* refers to a set of functionality that is delivered at the end of a cycle, in other words a subset (or full set) of the sprint backlog.

Risk agents (or ART agents) will manage the risk based on the input data defined earlier. This risk process is autonomous, where software agents; identify, assess and monitor risk based on the input data from the environment. Once any risk is triggered, risk data will be displayed in the Risk register. Any changes or updates to the environment will affect the risk data (whether or not the risk is flagged up).

At the Output stage, the final risk data can be obtained after the *delivery of the product* and during a Sprint review meeting. The *risk register* provides a view of all identified risk data. At the end, the data displayed in the Risk Register can be recorded and saved in the Risk data repository where this information can be used to plan future projects.

Further explanation on the ART prototype architecture and its process will be discussed in Section 4.2.1 and Section 4.3.

The development of the ART model was initiated on the basis of the problems and issues identified in risk management, as well as the desire to adopt this in an agile environment. The chosen solution approach proposes a process model that can collect data from the software engineering environment and manage risk based on the collected data as well as input from the project manager. To develop and illustrate the approach, a prototype tool, ART has been developed. To date while there are plenty of contributions to risk management in software engineering, there is to the author's knowledge no contribution in integrating risk management in agile projects, specifically using Scrum methods. Moreover, existing risk management generally focuses only on goals relating

to schedule, cost and quality (Islam, 2009). Previous approaches tend not to focus on factors like people and process. Nevertheless, project goals are important and failing to reach them is a threat to any project. For example, a project goal specific risk will relate to schedule, quality and cost but will also contain characteristics relating to people factors. A more generalized view as suggested by Hall in her book on software risk management (Hall, 1998) introduced an alternative view of software risk management by categorizing four different factors that influence the success or otherwise of a project. These factors are People, Process, Infrastructure, and Implementation ("P^2I^2"). Furthermore, Misra et. al. (2009) identified 9 out of 14 hypothesized factors to have a significant relationship with "Success", naming People factors as one of the success factors.

Thus, in order to establish the basis of the ART model in relation to risk management, the author reuse the Goal-driven software development risk management model (GSRM) as proposed by Islam (2009). This model used a four layers based concept to manage risk in software development risk where each of the layers is interrelated. The layers include Goal layer, Risk-obstacle layer, Assessment layer and Treatment layer (see Figure 4-2). The Goal layer consists of the objectives that are expected in relation to the project success. This layer includes identifying goals to support refinement of higher or lower level of goals using the AND or OR refinement model which latter was called sub goals. The Risk-obstacle layer refers to the risk factors i.e. events or situations that can obstruct the goals. The Assessment layer defines how the risk factors contribute to the obstruction of the goals and Treatment layer refers to the control actions that can avoid the risks happening. As for this research work, the solution scope is limited to only the first three layers. Firstly, the project goals are formulated into problem scenarios which contain possible risk factors. Then, the risk indicators are used to monitor the risks and observe if the goals are being met.

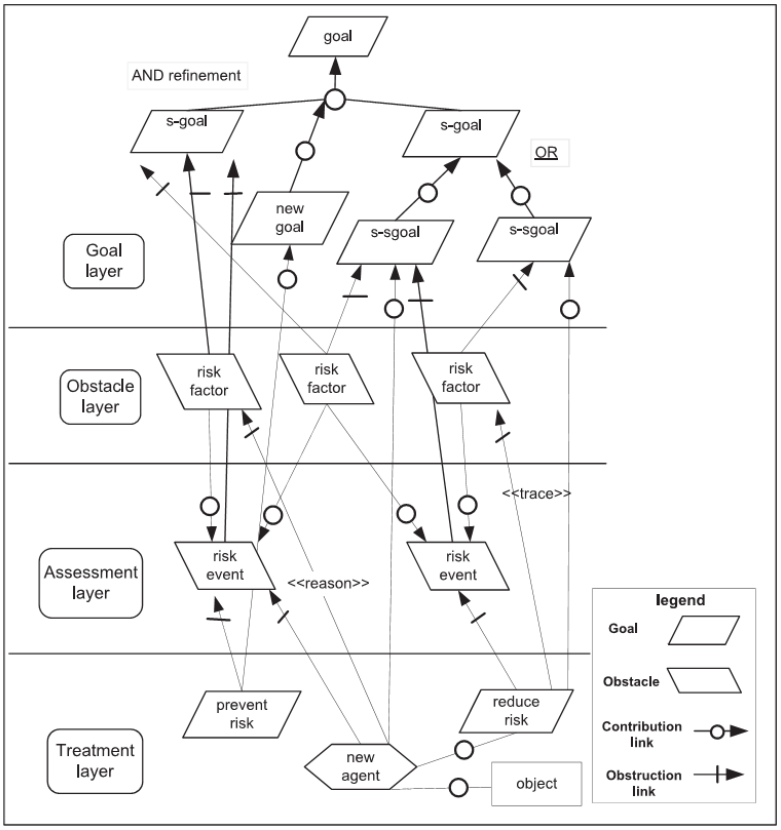


Figure 4-2: Goal-driven Software Risk Management Model (GSRM) (Islam, 2009)

The ART model can be described in terms of two main architectural components:

1) The *Agile Risk Tool* refers to the main engine of the tool which consists of the graphical user interface (GUI) for the Input and Output, the Rule engine and the ART agents. It interacts with the ART template, which is a template that is used to define the environment data. Once the ART template file that contains environment data is uploaded, this data can be modified using a GUI. Further explanation on ART template is discussed later in Section 4.3.1.1.

2) *Environment Data* refers to the data from the project environment. Changes in this data stimulate dynamic reaction from the ART agents. To achieve this, an ART template must be created for the project environment and data from the real project environment must be translated into this template. The categories of data used for this work were ‘Project’, ‘Team’, ‘Task’ and ‘Progress’. Any risks triggered will be stored in a risk data repository so that this data can be used in future to support risk decisions.

Both architectural components are illustrated in the diagram below (Figure 4-2) and each of which will be discussed further in this section.

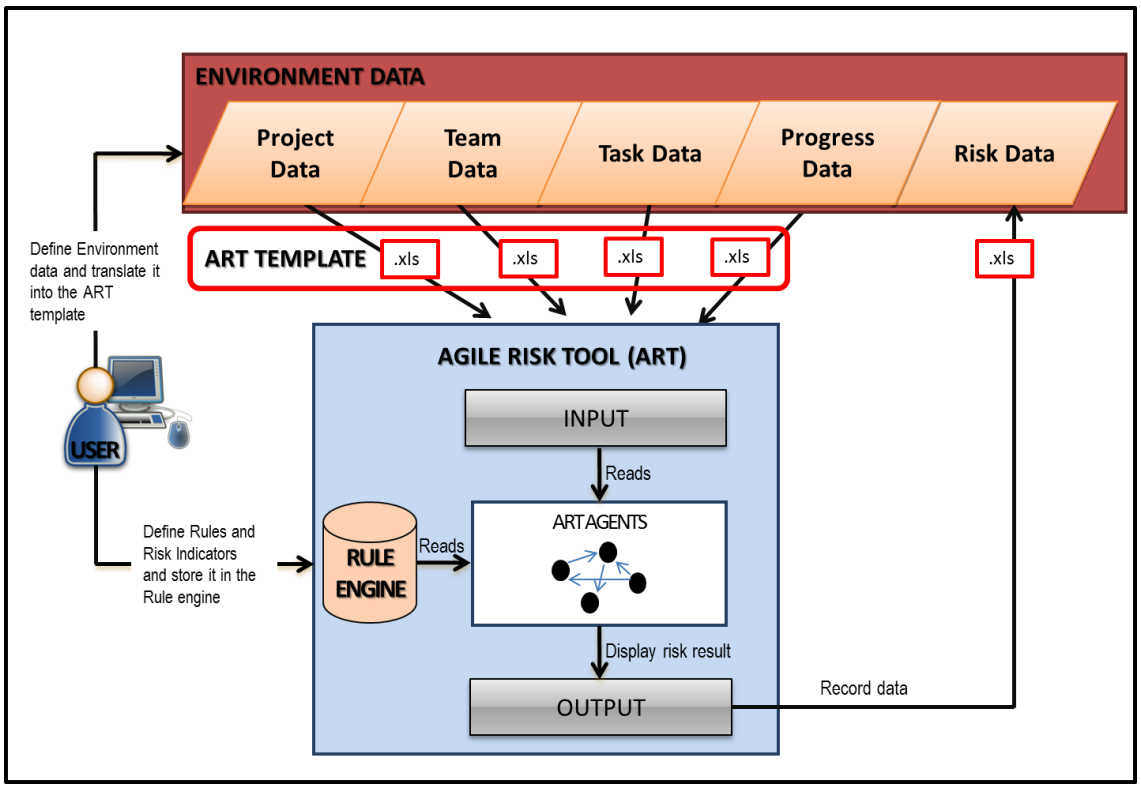


Figure 4-3: The Architecture of the Agile Risk Tool (ART)

4.2.1 The Agile Risk Tool (ART)

The architecture of the main engine of the Agile Risk Tool (ART) prototype tool is depicted in the diagram Figure 4-3. The tool has three main elements; the Input/Output, the ART agents, and the Rule engine.

4.2.1.1 Input/Output

Previously (Section 3.5) issues in agile projects were discussed and it was found that most of the problems related to the people involved and their motivation and skills in software development. Indeed Agile relies heavily on the competency of the people involved. Therefore the issues from Chapter 3 are converted to risk factors i.e. situations or events that may cause a loss to occur and therefore that we need to monitor in a project. Table 4-1 summarises the issues found in the literature and focusing on two agile aspects: the people and the process.

Further in this section, there is a need to specify the input for the project which consists of the type of risks and its risk indicators as well as the environment data which can be used to identify the risks for the project. As indicated earlier, the first three layers of the

GSRM model (Islam, 2009) are reused for translating the project goals into problem scenarios. Thus the issues in Table 4-1 are transformed into a set of sprint goals. These will later be used to define the risks and their indicators thus allowing risks to be monitored continuously.

Table 4-1: Summary of Problems or Issues in Agile Projects from Section 3.5

Agile Environment	**Problems or Issues in Agile Projects**	**Source(s)**
People	• Multiple responsibilities or no accountability or ownership • Developers reluctant to apply Scrum • Difficult to collaborate	(Cho, 2008)
	• Lack of individual competency – roles, responsibilities and skills	(Cockburn and Highsmith, 2001)
	• Personnel turnover, lack of agile skills causing job dissatisfaction	(Boehm and Turner, 2003); (Melnik and Maurer, 2006); (Melo et. al., 2011)
	• Insufficient agile training • Poor Product Owner involvement • Single expert and passive developer	(Chakradhar, 2009)
Process	• Daily meetings in Scrum ceremonies – inefficient meeting, waste of time because sometimes it involves more time than usual. • Not enough people skilled in agile, therefore difficult to form Scrum team with relevant skills	(Cho, 2008)
	• Poor task analysis and estimation causes problem in maintaining sustainable pace	(Deemer and Benefield, 2006)
	• Complex or easy tasks give negative impact on pair programming and shared ownership	(Boehm and Turner, 2003); (Melo et. al., 2011)
	• Collective code ownership – written code not conforming to standards • Underestimate completion time, aggressive dateline and working time causing difficulty in maintaining sustainable pace	(Layman et. al., 2006)

A risk can be a threat to a goal. As such, any generic problems identified in agile projects can be considered as risk categories for agile projects. Intuitively, resolving a problem normally entails investigating its root. Knowing the problems in agile projects generally allows to maps the problems into set of goals. Then, these goals are monitor specifically in a project and identify possible risk events that can threaten the achievement of the goals. The proposed solution approach offers indication, assessment and monitoring of risks from the agile project environment. Therefore, risks are more closely monitored if not avoided in the first place.

In order to identify the sprint goal for this project, the list of issues found earlier are grouped and assigned an appropriate goal for each item. An identifier was assigned for each goal and this identifier is used throughout this work. Sprint goal rather than project goal was used to allow each sprint to have different goals and different risks associated with each goal. Table 4-2 below presents the sprint goals that are mapped to the problems identified earlier in Table 4-1.

Table 4-2: Mapping Problem Identified to the Sprint Goal of the Project

Sprint Goal	**Problems (Table 4-1)**	**Identifier**
In Sprint X, Task Y should be assigned to appropriate number of developers once Sprint X is started	• Pair programming • No accountability or ownership • Collective code ownership	G1: Task ownership
In Sprint X, Task Y should be assigned to a proper skilled team member	• Not enough people skilled in agile / forming Scrum team with relevant skills • Insufficient agile training	G2: Skills and Experience
In Sprint X, developer should focus on one role and one project at a time	• Multiple responsibilities	G3: Resources
In Sprint X, developer should attend Daily Scrum Meeting and provide task Y progress	• Personnel turnover • Daily meetings in Scrum ceremonies – inefficient meeting, waste of time because sometimes involves more time than usual	G4: Progress

The ART GSRM model consists of the goal, risk-obstacle and assessment layers. This model is referred and used throughout this chapter starting from determining the Input

for ART prototype tool. A visual representation on how a GSRM model is adopted to identify goals is presented in Figure 4-4.

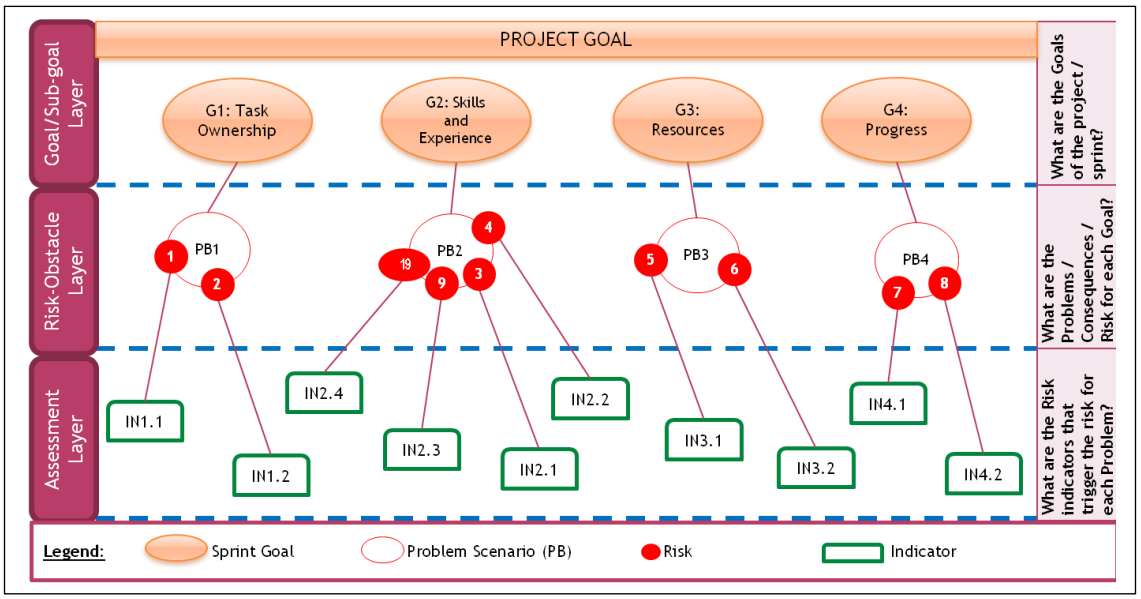


Figure 4-4: The Agile Risk Tool (ART) GSRM Model

At the top layer (Goal/Sub-goal layer) of the model, a set of sprint goals are developed. For this work, four sprint goals are proposed based on the problems identified (Table 4-2) and mapped this to the problem scenario and risks (Risk-Obstacle layer) that could possibly threaten the sprint goal. Further, the risk and possible indicators (Assessment layer) are mapped accordingly between two that could later provide an alert which will trigger a risk. In a real world situation, each project might have different goals and risks associated to the goals. However, the set of goals proposed for this work are limited to the problems identified in the literature. An example of how the ART GSRM model is applied is shown below.

Goal 1 (G1): *In Sprint X, Task Y should be done in pairs*
Problem Scenario (PB1): *During the sprint, the developer does not practice pair programming*
Risk 1: *Pair programming is not applied, single expert risk occurs*
Indicator 1.1: *When the sprint cycle is started, a task being selected in the sprint and the selected task has no pair with another developer*

It is recommended that the project manager apply this model prior to the start of the project so that they will have a clear view of what the project goals are and the risks that they want to monitor. The indication of the goal, problem scenario, risk and indicator used for a project is subjective and depending on the requirements of the project. When

a goal (G) is defined, it will be up to the project manager to expand and identify the activities or actions of other layers in the model. The problem scenario (PB) is the opposition of the goal and it will happen if the defined goal is not met. The risk indicator (IND) is used to measure the risk and at a certain level indicate a condition that will trigger the risk. These form the basis of a better way of monitoring the risk. In addition, the risk data recorded for the particular project can be used to predict future problems and risks.

A template called the Rule Template is proposed, that can be used as a baseline for developing the model. The design of the template is discussed in the following sub section and the application of this Rule template is further elaborated in the following chapter. This includes the extended version of the Rule template distinguishing each goal presented in the ART GSRM model as applied in this work.

The next sub section will describe more about the ART agents and Rule engine.

4.2.1.2 ART Agents

Earlier in Chapter 1, a lightweight risk management approach was proposed to reduce barriers in risk management application. This includes three main steps in risk management; risk identification, risk assessment and risk monitoring. The rationale of doing so was twofold (i) to develop a realistic and acceptable risk management process that can fit into the agile methods (ii) an empirical study (Odzaly et. al., 2009) confirmed the most complicated steps in managing risks were risk identification and risk monitoring. In addition, prior to this chapter evidence is established that contended that risk management was difficult mainly due to the required human effort. Given this, the aim is to substitute some of the human involvement with autonomous software agents with the goal that these could manage risk and minimize the need for manual input. Automated agents can therefore help ease the work load in managing risk, specifically in identifying, assessing and monitoring risk.

Software agents help to perform on behalf of other programs or people with some degree of self-determination and employing goals or rules in order to achieve some behaviour. From this it is easy to see how they may help overcome some of the barriers of human effort to risk management. To demonstrate this work, the agent development environment, JADE (Java Agent Development Environment) was chosen as a suitable framework for development. As stated in Chapter 2 JADE is a powerful environment

that is popular and has been used extensively. It has been valued due to its interoperability, uniformity and portability, ease of use and standalone features (Bellifemine et. al., 2007).

Autonomy implies that these agents have the ability to perform their tasks without direct control, or at least with a minimum of supervision. This work focuses more on the managing domain where the agents representing part of risk management activities will have their own goal and rules as well as needing to interact with each other. A simple scenario for the proposed solution is that the agents will be able to identify, assess and monitor risks as well as react dynamically to changes in the project environment. Hence, their output will help the project manager to generate and execute plans based on the risks output. These plans may initiate risk avoidance or transfer, based on the agent-assessed severity of the risk. The ultimate goal is to extend the agents' goals with the necessary intelligence to make judgements and decisions based on risk analysis.

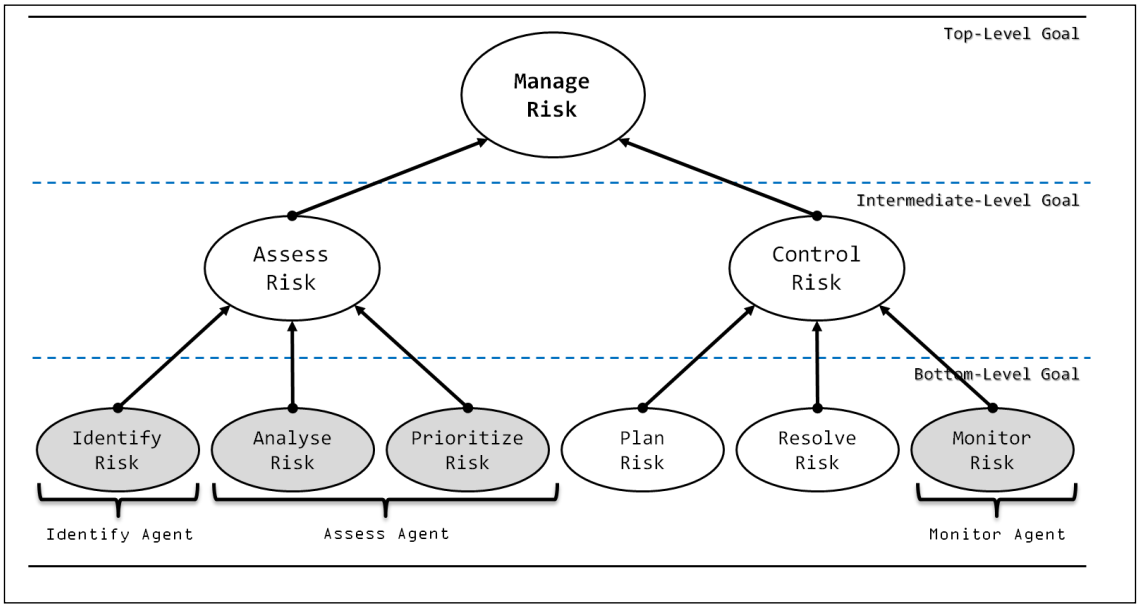


Figure 4-5: Risk decomposition graph for the Agile Risk Tool (ART) agents of four risk management activities (Bresciani et. al., 2002)

Decomposition of risks into activities is commonplace. Kontio (1997) used decomposition of risk into conceptual elements like risk factor, risk event, risk outcome, risk reaction, risk effect and utility loss. More recently a top down goal decomposition technique is described in (Bresciani et. al. 2002) and (Dardenne et. al. 1993). Indeed Boehm's tutorial on risk (1989) decomposes risk management into activities. In this work the category or type of agents used was derived based on initial agent goal decomposition as shown in Figure 4-5, based on Boehm's work.

The generic aim of this research is to find ways of lowering the barriers to application of risk management. One of the objectives is to use the agents since agent behaviour is more adaptable and can act on behalf of the project manager of the agile project. In this case, some of the effort of the project manager is replaced by agent execution such that they will react automatically according to their own goals. In identifying goals for the agents, the top level goal is started in order to apply risk management in software development project, particularly in agile projects. This goal is further decomposed into two intermediate sub goals; assessing risk and controlling risk. These sub goals are then decomposed into six smaller sub goals; identify, analyse, prioritize, plan, resolve and monitor. As a result of the decomposition of the goal, agents were assigned based on the smallest sub goals which supported the top level goal. Since the most effort intensive steps identified earlier were identification and monitoring, for the meantime, both sub

goals were selected in addition to analyse and prioritize goals as highlighted in Figure 4-5. Note that here that only the bottom level goals are engaged; the assumption being that top and intermediate level goals might have largely a controlling function but nonetheless have their own goals on how lower level agents should interact.

Further ART agents were developed for this work as four agents; Manager Agent, Identify Agent, Assess Agent (combines analyse and prioritize goals) and Monitor Agent.

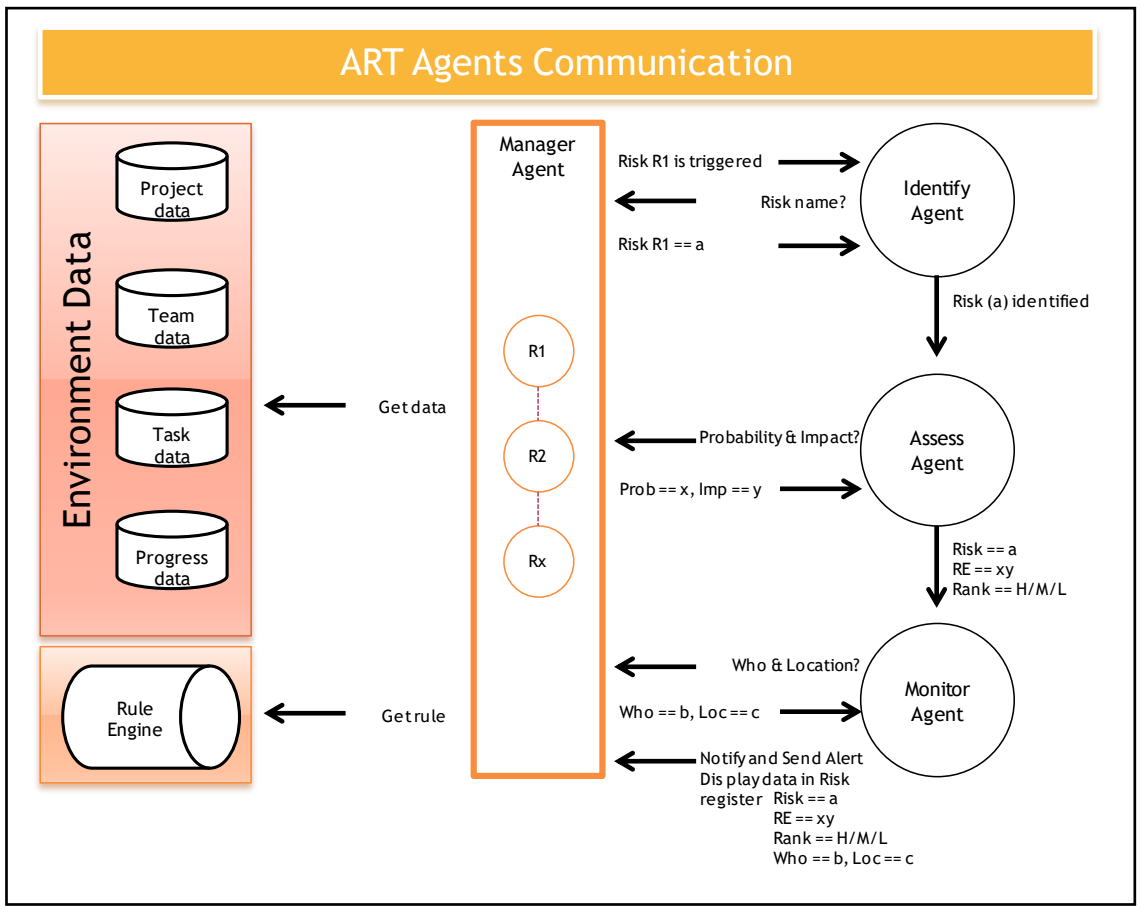


Figure 4-6: The communication between the ART agents and how they interact within the environment data and rule engine

The ART agents model is depicted as in Figure 4-6 and shows the interactions between Manager agent and the Identify, Assess and Monitor agents. Figure 4-6 show how these communicate via message passing between the agents. Depending on the data from the environment, the agents react to detect risk dynamically through rules execution, where rules are invoked from the rule engine. The ART agents' communication is described further as below.

- *Manager Agent* acts as an intermediate between the other three agents. It manages and executes rules (to be discussed in Section 4.2.1.3) gets data from the project environment (to be discussed in Section 4.2.2) and notifies the Identify Agent if any risk is triggered.

- *Identify Agent* is notified if any risk is triggered. It is controlled by the Manager Agent and notifies the Assess Agent for the next action.

- *Assess agent* is invoked by the Identify agent and its goal is to calculate the Risk Exposure (RE) (see Appendix B) of the identified risk where RE = Probability x Impact. The identified risk is ranked as Extreme, High, Moderate or Low and the Monitor agent is notified for subsequent action.

- *Monitor Agent* is invoked by the Assess Agent receiving data from it: Risk Exposure (RE) and the rank of the identified risk. The Monitor agent will establish the location of the identified risk along with the owner of the risk. The data is then displayed in the Risk Register. The Risk Register is represented as a display screen for all identified risk data. Data displayed in the Risk Register can be recorded and saved in the Risk data repository. The documented risk data can be used in future to plan and mitigate risks for future projects.

While implementing the agent reaction towards the environment it needs the support of the rule engine. The rule-based engine and framework used in this work is discussed in the next sub section.

4.2.1.3 Rule engine

In order to allow ART agents to use information in the environment data, agents need to be able to reason and have access to a set of inference rules and logic statements in order to be able to execute the rules. Moreover, due to the fact that the defined problems are on events basis, rules are proposed to be used in order to capture sets of actions to be taken by the agents. These are stored in the rule engine, as shown in Figure 4-6.

A rule engine refers to an engine which complements a set of facts against a set of rules, the working memory and executes an action for matching rules.

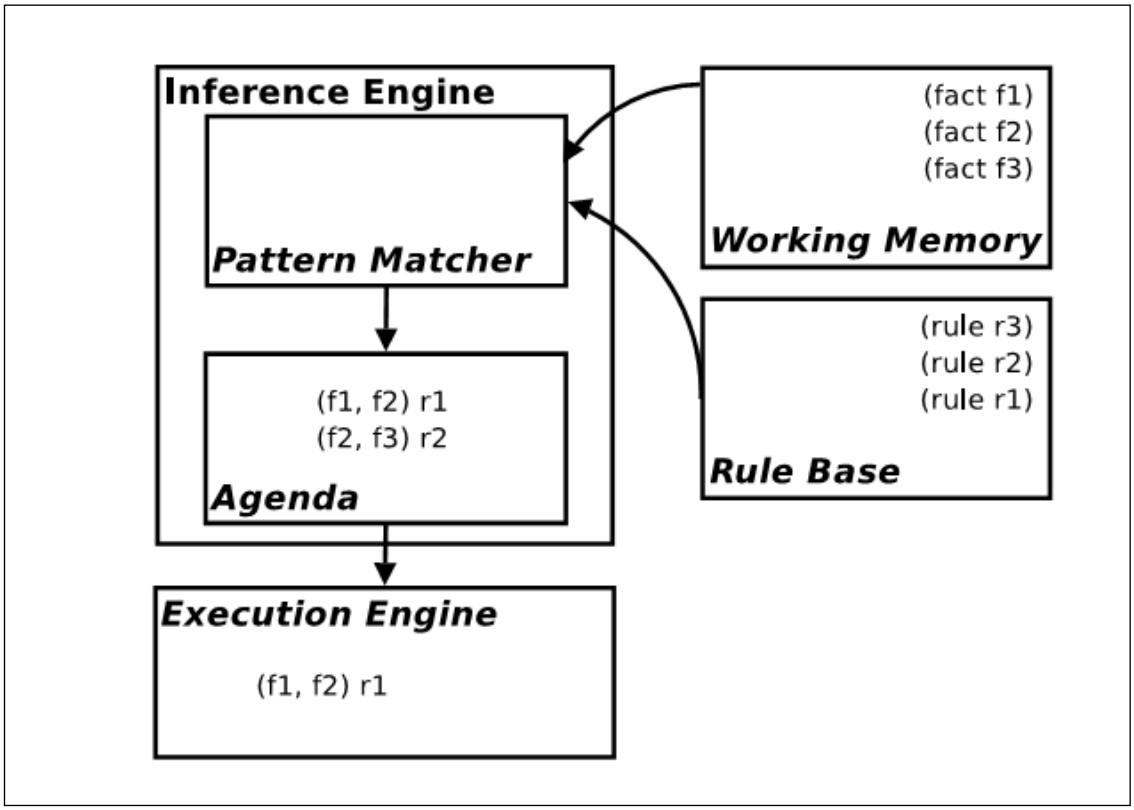


Figure 4-7: The general rule-based architecture (Friedman-Hill, 2003)

The main architecture is shown in Figure 4-7. It consists of the working memory (fact base), the rule base (knowledge base) and the inference engine (rule engine) (Friedman-Hill, 2003). The working memory contains facts which are the most basic construct in the architecture. Rules are in the form of if-then statements and represent the knowledge of the user and/or problem domain expert and are held in a rule base. On running, facts are matched by the inference engine using working memory with rules in the rule base. The inference engine determines the rule(s) are that apply using whatever reasoning is built in. A conflict set is then created consisting of the list of applicable rules that have been determined by the inference engine. The system will have a conflict strategy and this is used to select and fire a rule from the conflict set thus executing its associated actions. The inference engine continues to select the next rule and repeat the process until all rules are executed.

In the context of applying the rule engine in the ART model the model used was influenced by that proposed by Van Raalte (2009).

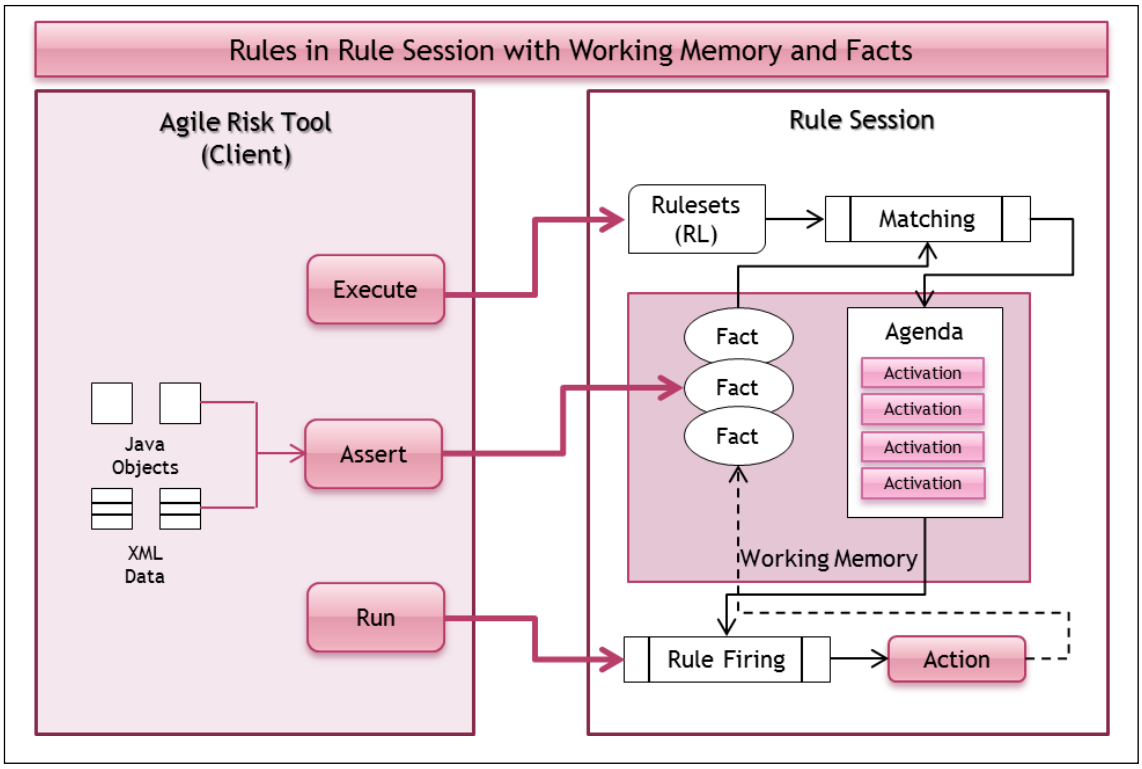


Figure 4-8: Rules in Rule Session with Working Memory and Facts (Van Raalte 2009)

The diagram (Figure 4-8) above shows the interaction between the ART prototype as a client and the rules in the rule session. A rule session consists of set of rules, facts and an agenda. Facts works like a database where the records can include objects and classes. Facts are structured in advance using a designated template (the ART template), which consists of a table of properties. During execution, rules will match the objects which refer to the properties and values as defined in the template and fire the rule if activated. Facts also can be asserted or retracted from the working memory. Rules on the other hand contain a list of conditions encoded into IF/THEN rules. A ruleset contains a set of rules to be executed. When a set of rules is executed, the engine will find matches from the facts and store this in the agenda for activation. Agenda acts as a scheduler which controls the execution and prepares rules for activation. Once the rule engine runs, it executes action and fires the activated rule.

In this work, the rule engine was developed using Jess (Java Expert System Shell) which supports the development of rule-based systems that can be tightly coupled to code written in Java (Friedman-Hill, 2003).

A program using Jess may consists of facts, rules and objects and can be defined for matching fact patterns that contain Java objects. It uses the RETE algorithm (Friedman-Hill, 2003) to process rules. Jess can be incorporated with many agent frameworks, including JADE.

Earlier in this chapter problems and issues were summarized and mapped these to a set of sprint goals. In the ART GSRM model presented earlier, a set of indicators were derived for the identified problems and risks. As such, the indicators were used to generate rules to then develop inputs for the rule engine. The indicators are determined beforehand using the Rule template in Table 4-3 below.

Table 4-3: Rule template

Goal	**The sprint goal that needs to be defined before the project starts**
Problem scenario	A possible risk event that associated with the sprint goal
Consequences	The penalty if the risk is occurred
Indicators	The events or measures that forewarn of the risk event and their values
Repository/Data	The list of repository of environment data involve in this risk event
Rule(s)	The list of rule(s) that trigger the risk event
Risk name	The unique name for the risk

Table 4-3 represents a standard template that allows the manager to define the Sprint Goal, Problem Scenario, Consequences, Indicators, Repository, Rules and Risk Name before the project starts. Each problem scenario proposes a possible risk event that is associated with a sprint goal for the project. Defining a sprint goal is important in that it allows us to identify how data from the development environment can serve as an indicator to a risk event in the project. Hence, the rules were generated using the indicators to identify risk based on the objects defined in the environment data. This work is started by employing details from the ART GSRM model which consists of sprint goal, problem scenario, risks and indicators and further defined the consequence of each problem if the risk where to occur. Subsequently, the rule is constructed to produce an alert to the risk.

There remains the possibility that there is not sufficient indicators that can be implicitly collected from the environment; in this case it is up to the project manager to decide on

additional data and metric collection activities. These can be goal and priority driven for example using the goal decomposition graph by first listing the goal that one would be most interested in, and secondly defining the data and measures that have to be collected to detect the threat to this goal, that is a risk.

In some cases, the project manager will have to consider (or brainstorm with others) what risks to monitor and possible indicators that contribute to setting up the rules. An example of an indicator is: "*When a project is started, there is more than 10% of code defects found in Sprint X"*. At first, an educated guess of parameter values for the indicator is acceptable and thereafter they can be iteratively refined, if necessary. A second strategy is to let the practice run for a short time and derive the parameters from the observed data. This can be understood as the derivation of the process from its execution. However, this only works under the assumption that the process is performed appropriately.

The next section will explain further about the environment data used in this study.

4.2.2 The Environment Data

As far as the agile development process is concerned, XP and Scrum are the most widely used agile methodologies. This is evidenced from the various surveys stating that a majority of the respondents use Scrum or XP in their projects (e.g. Ambler, 2006; VersionOne, 2013). This research work therefore will concentrate on those methods. Further an agile development survey (VersionOne, 2013), stated that agile companies use a wide variety of agile tools and 60% of them are currently using an agile project management tool. The tool choice might be different ranging from a simple spreadsheet to commercial tools such as VersionOne. Tools help the project manager to have better visualization in delivering the project and in more interactive manner. Therefore, two agile project management tools are studied, Rally software[4] (RS) and Extreme manager[5] (EM), considering what is accessible in terms of configuring their product features and data design. In both cases information was gathered on the environment data model to include their process model, the objects used, and the attributes and values for those objects. The outcome from this was a more generalized definition of the available data that could be available for this work.

[4] http://www.rallydev.com
[5] http://www.hindsa.com

4.2.2.1 Rally Software

Rally Software is an interactive web-based tool that supports agile project management practices and claims to provide a complete integration of the software development lifecycle. Rally software depicts a model that includes data on Project, Backlog, User stories, and Release and Iteration Planning. Rally is one of the most widely used agile development tools as it has been a tool of choice for almost 30% of development organizations (Behrens, 2006). However, the data model is proprietary and difficult to access so that using it for research is difficult.

4.2.2.2 eXtreme Manager (EM)

eXtreme Manager has a similar basic structure as Rally software. They claimed to be more effective in managing risk for projects, even when requirements are frequently changing. However, eXtreme Manager is less popular but appears to be more generalized and specifically designed for companies adopting agile development methodologies such as eXtreme Programming, Scrum, DSDM, RUP, Crystal, and Feature Driven Development. eXtreme Manager captures data that includes Product, Requirements/User stories, Task, Acceptance test, Release and Iteration Planning.

A comparison between the two tools was made in order to establish a generalised dataset to represent the required environment model for ART, with a view to eventually making ART generic across agile lifecycle tools. The findings are summarized in Table 4-4 which shows a list of attributes for which values are available from both eXtreme Manager (EM) and Rally Software with an indication of which attributes can be possibly used in this work, focusing only on the phases of Product backlog and Sprint backlog. An extended version of this table including all phases of process is included in Appendix C.

Generally, both tools use the same process model although the naming convention might be different. For example, in Product level, EM uses *Product* while Rally uses *Project* in naming process level. There were also similarities in term of the attributes used for example, *User stories* or *Product backlog id*, *name* and *priority*. Both tools were methodically studied and compared in order to obtain a standard list of objects and attributes for the environment data model. These were then selected based on whether they were suitable to be used in this work. However, some of the attributes were eliminated to avoid complexity; for example parent and release attributes, which refer to

static data or information that cannot possibly be the indicator to risk. While doing so, some important attributes found were not provided in either tool; for example information on the team skills and confirmation of attendance in the Daily Scrum meeting. Therefore, these attributes were added in the model with the assumption that these can be collected manually. The standard attributes are then will be used as a baseline dataset for the ART prototype. It is expected, however, that these can be modified or changed to suit individual organizational needs.

Table 4-4: Comparison of the objects and attributes in two agile project management tools; Extreme manager and Rally software

Scrum Process	**Extreme Manager**			**Rally Software**		
	Objects	Attributes	Values	Objects	Attributes	Values
Product	Product	• Product Market Name		Project	• Name	
Product Backlog	Requirements/ User stories	• Requirement/User Story Id • Title • Priority • Requirements Type • Status • Created By • Developed by • Paired by • Development Started on • Development Completed On • Iteration	 Highest, High, Normal, Low, Lowest Bug Report, New Feature, Product Integration, Scalability/Operational, Technical Refinement Draft, Confirmed, Estimated, Pending/Scheduled, In Progress, Completed, Deleted, All	Backlog	• Rank • Id • Name • Plan Estimate • Priority • Owner • Accepted Date • Task Status • Iteration • Project	 None, Resolve Immediately, High attention, Normal, Low
Sprint Backlog	Task	• Task Id • Title		User stories	• Rank • Id	

		• Priority • Task Type • Status • Developed By • Paired By • Development Started On • Development Completed On • Iteration	New Development, Refactoring, Integration, Investigation, Quality Assurance, Documentation		• Name • Iteration • State • Plan Estimate • Task Estimate • Owner	Defined, In Progress, Completed, Accepted

The environment data model used for this research project is shown in Figure 4-9 as derived from the environment data gathered from the studied project management tools in Table 4-4. Some simplifications were made to create a standard generalised model. For example, instead of using *Product market name* from the XM project management tool the *Project name* were used since 'Project' is more general in that not all development efforts are for a 'Product'. The derived model serves as a logical data model of the ART template, where the objects represent a corresponding dataset pattern, later defined in an XML schema and used in the ART template. Further details regarding the ART template will be discussed in Section 4.3.1.1.

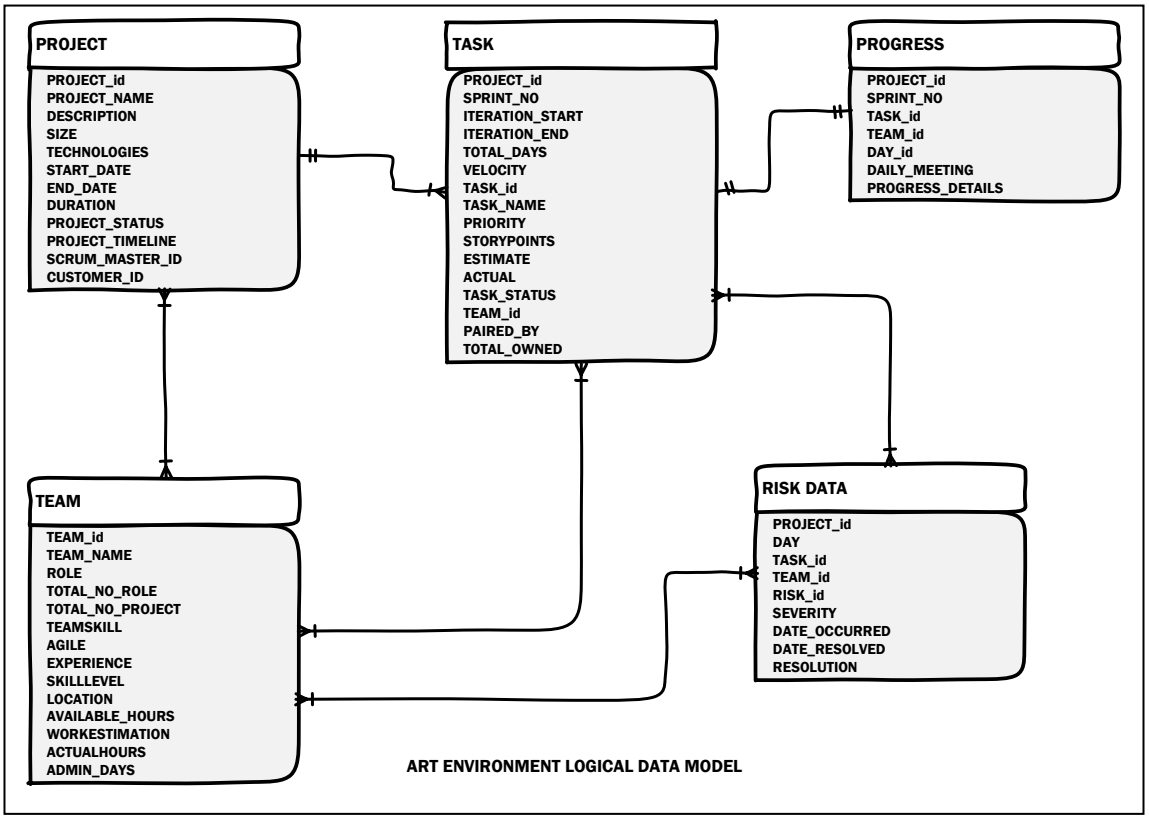


Figure 4-9: The ART environment logical data model

The next step taken was to configure this data model as an XML schema and so to specify the data type of each of the attributes. XML schema language as defined by Van der Vlist (2011) is useful for validation of the constraints that consist of elements and attributes and to define a document content type and structure. XML is 'human-readable' but allows the transformation of formal descriptions into graphical views that are easier to understand. XML schema benefit also from its universal and broadly applicable document format, allowing reading and writing of XML data from different types of application (Harold, 2004). An excerpt of the ART XML schema used in this

work is depicted in Figure 4-10. This figure shows the elements that belong to the object ‘Task’ and the data types of each element of Task data.

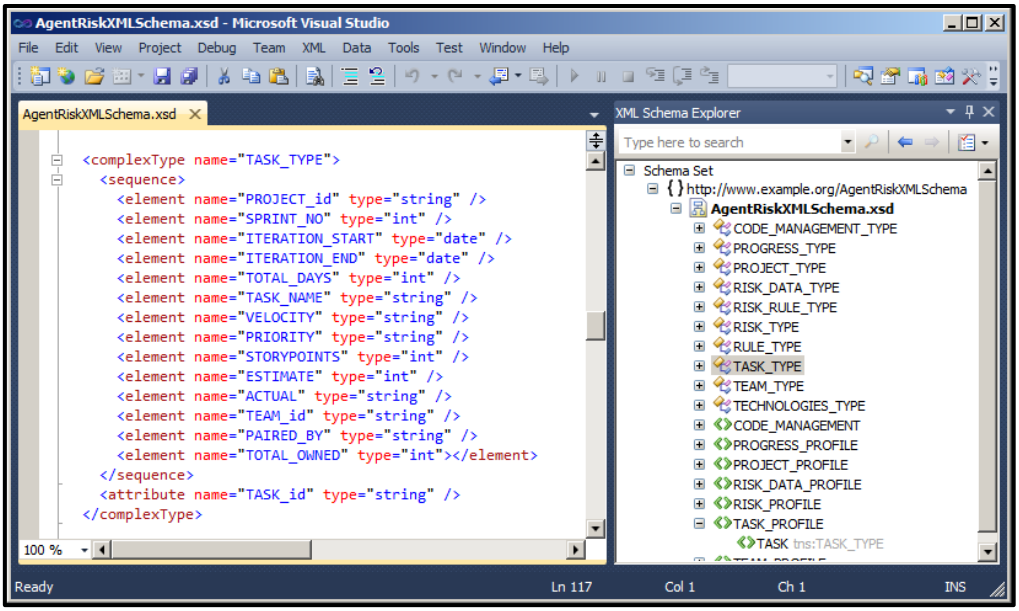


Figure 4-10: An excerpt of AgentRiskXMLSchema.xsd for Task data

Once the XML schema is defined, the next step is to structure the data into an XML document. Due to the objective of collecting the environment data from the project management tool or archived data, the ART template developed is designed to be reusable, based on the object and attributes illustrated in Figure 4-9. This template is implemented as a spreadsheet, this format being chosen as it is widely used in project management.

As can be seen from the list in Table 4-4, there is a substantial list of attributes that can possibly help in managing risk. In this work the desire is to validate the overall idea but also at the same time to maintain agility in the process. The eventual aim is to support risk management by embedding this work in the agile environment coupled with existing agile project management tools. Hence the most appropriate attributes were selected i.e. those related to (i) identifying problems in agile projects as discussed in the previous chapter (ii) the most likely available attributes in common project environments which can also be easily collected. The data that are used specifically for this work is shown in Table 4-5. In the longer term, however, this data can be expanded on according to the needs of a specific agile project.

Table 4-5: List of selected environment data used in this work

Data/ Objects	List of Attributes
Product / Project*	Project ID / Name Project Start date / End date User Story ID / Name User Story Estimation User Story Priority User Story Owner ID / Name Task ID / Name Task Estimation Task Priority Task Owner ID / Name Task Status Task Paired By (Pair Programming)
People**	Team Member ID / Name Total no. of Role in a Project Total no. of Project Involved Programming Skill Level Agile Experience Level
Progress**	Attendance in Daily Standup Meeting Daily Progress Report on Assigned Task

*Refers to data available in both EM and RS

**Additional data collection that is required in this work

These data were then translated into the ART template in form of objects and attributes, where the value of each attribute can be collected from the real project. The ART template represents as data within the project environment in which changes in the project environment are captured dynamically by the ART agents.

The next section will explain the process in detail in applying this proposed approach.

4.3 ART Process

In order to support the ART process the ART prototype tool was developed and used to demonstrate the reactions of the ART agents towards the changes in the environment data following the execution of the set of rules built from consideration of goals. The ART process flow is depicted in Figure 4-11.

For any product it is important to consider its usability (Dumas, 1999). Since the demonstration of this tool represents the rationality of this work demonstrating its usability is critical. Under those circumstances an informal usability evaluation method were employed that suits the purpose which is a heuristic evaluation. Heuristic evaluation is a common informal method in which evaluators examine a user interface and look for problems that violate recognised principles (heuristics) (Nielsen, 1994). The process involves the steps: pre-evaluation training session; individual evaluations; debriefing session, if needed; and combination of problems and estimation of severity of the problems. However, the main contribution of this work is not in proving how good the tool is but to demonstrate the functionality of the tool in fulfilling the research aims.

The next sub section will demonstrate the ART prototype tool using a set of dummy data to demonstrate the process and associated tool functionality.

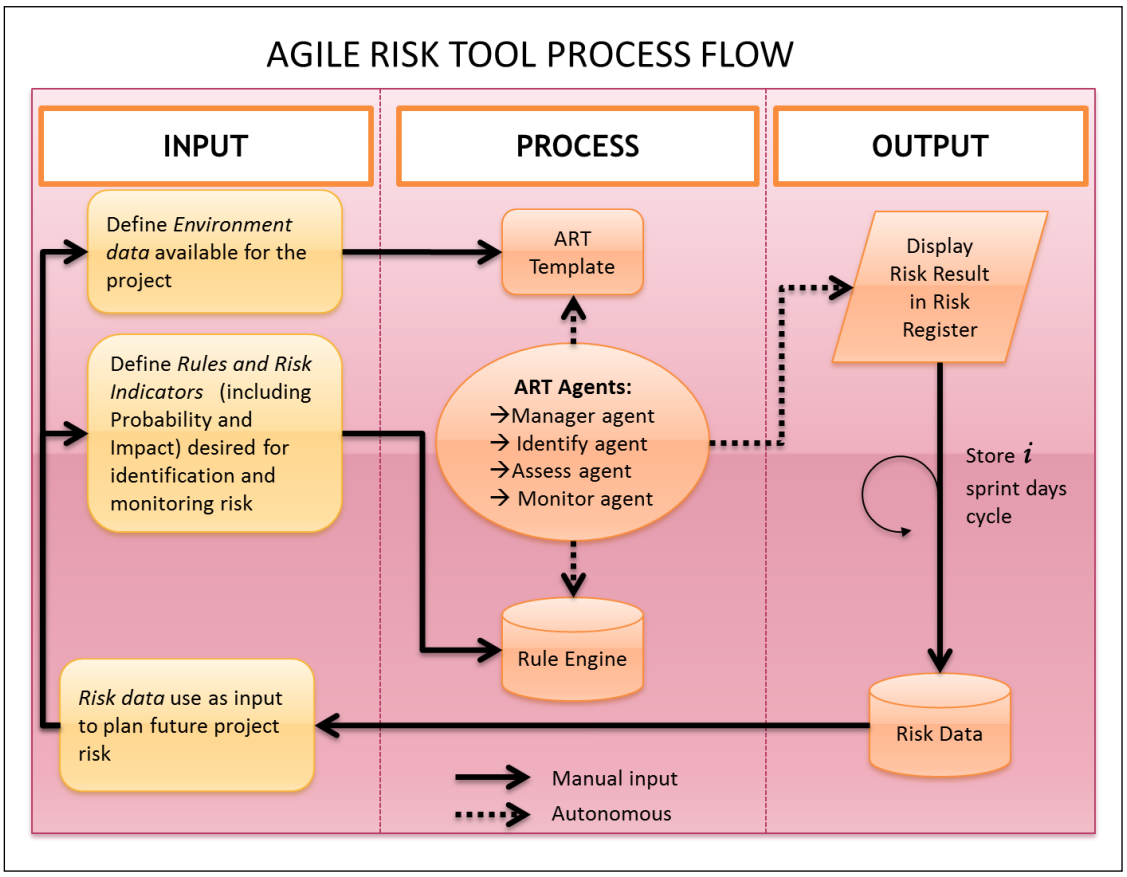


Figure 4-11: ART Process describing the flow or steps starting from defining the Input and storing the Output in the Risk data file

4.3.1 Input

This process is started at the Input stage. At this stage, there are two main inputs needed: defining the environment data available for the project and defining the rules and risk indicators to monitor risks.

4.3.1.1 Defining the environment data available for the project

Firstly, one should define what environment data is available in the project. Since that this was constrained by the modern Software Engineering environment and the need for agility, the data was limited to the easily collected data. An example of the form of environment data in a project includes user stories name and id, task name and id, task priority, task owner and so on. The collected data are then translated into the ART template. The ART template is referred to the structure of XML schema that consists of objects and attributes as shown previously in Figure 4-9. It contains data and values collected from the project environment. Figure 4-12 below shows an excerpt of transformation from the Environment data model into the ART template which contains

some data from a project environment called Task data. This step however should be implemented prior to the start of the project.

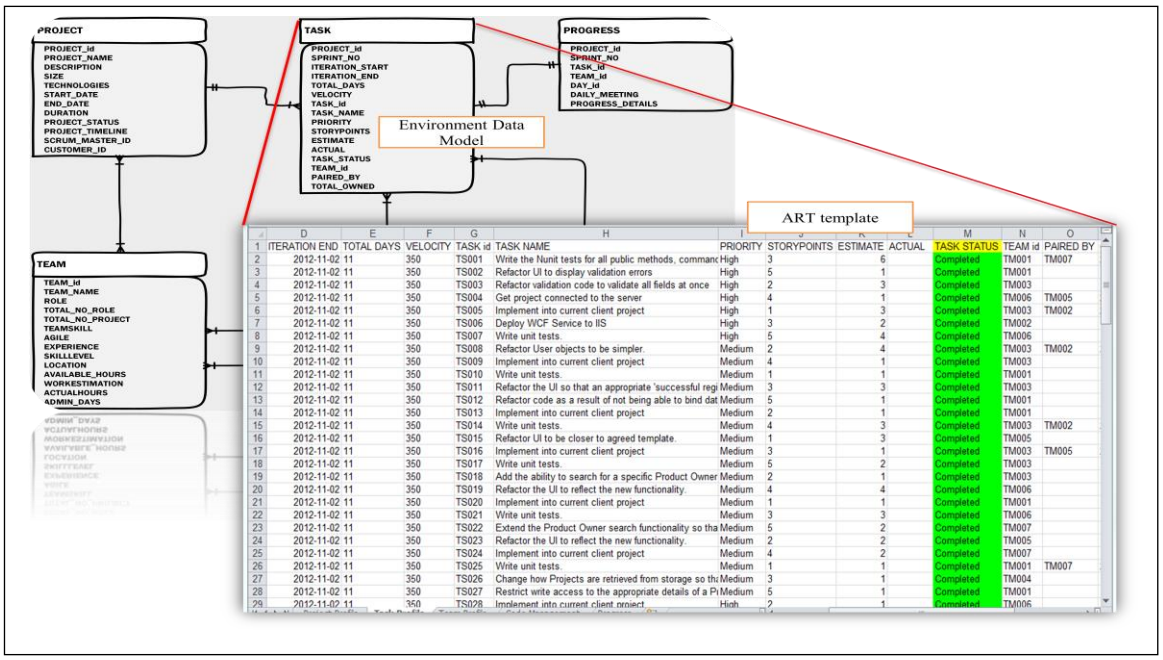


Figure 4-12: An excerpt taken from the Environment Data model into ART template for collecting Task data

Once the environment data is translated into the ART template, the ART prototype tool (which from now on is referred to as the tool) is used to access this data. On the main screen, the [Open Project] button is selected. Upon selecting this button, a new window is prompted to allow selection of projects in spreadsheet format. This project presents the ART template that been translated earlier. If there is no template is available, it is will be necessary to create one before starting the project.

Once a project is selected, the tool will return the window to set the profile of the selected project (Figure 4-13).

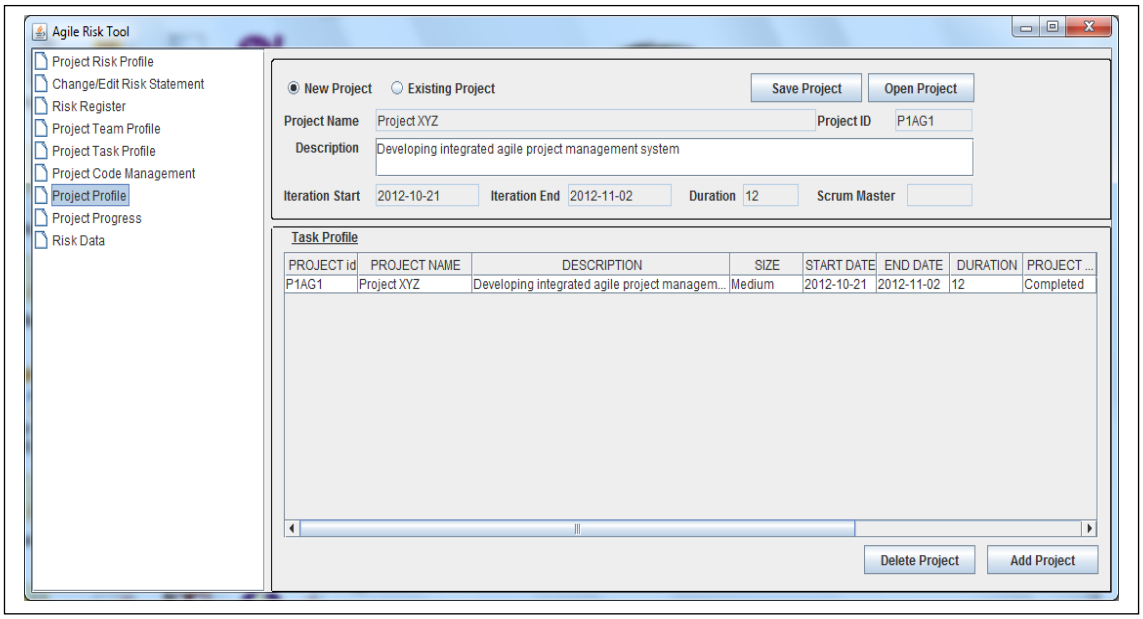


Figure 4-13: Agile Risk Tool - Project Profile Screen

Once the project is loaded, one can navigate to team data, task data and so on as well as edit the data using the main screen rather than having to go back to the ART template to make changes and load the project back into the tool.

The following figure, 4-14 depicts the input windows to define the rules and risk indicators.

4.3.1.2 Defining the rules and risk indicators

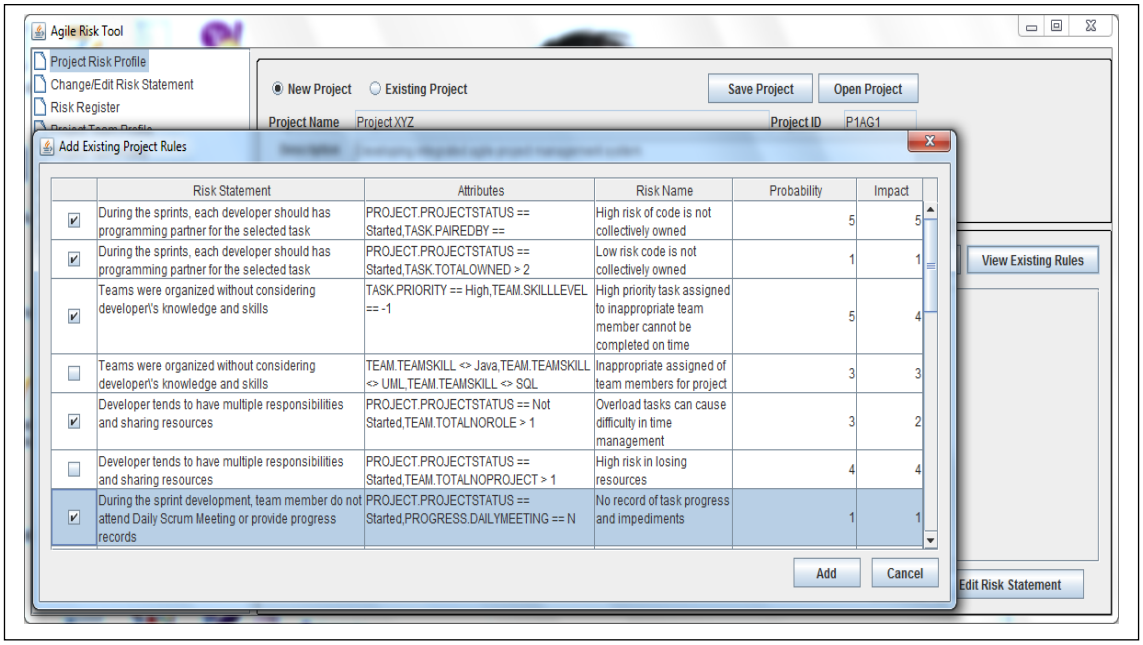


Figure 4-14: Agile Risk Tool - Adding Rules into the Project Screen

The definition of rules and risk indicators for the project allows one to identify what risks to monitor for this project and from which data or indicator that the risks could possibly be triggered. At this stage, one can either add new rule or add an existing rule where the rule was created from a previous project. Example of the list of rules initially embedded in the tool is shown in Figure 4-14. Designated rules can be selected (Figure 4-15). Note that, the embedded rules in the tool was designed based on the problem scenarios discussed earlier in this chapter and will be used throughout this project.

The logical concept of risk management activity in this work refers to the following:

- The collection (e.g. by brainstorming) of possible risks in the project;
- Identification of the environment data which can be used to identify risks;
- The establishment of the risk indicators and risk rules;
- Management of identified risk based on the rule which flags up the risk;
- Action advised or taken in order to reduce the risk.

Therefore, risk is embodied in the entity which encapsulates the risk data and once the rule flags up the risk then it is considered an identified risk.

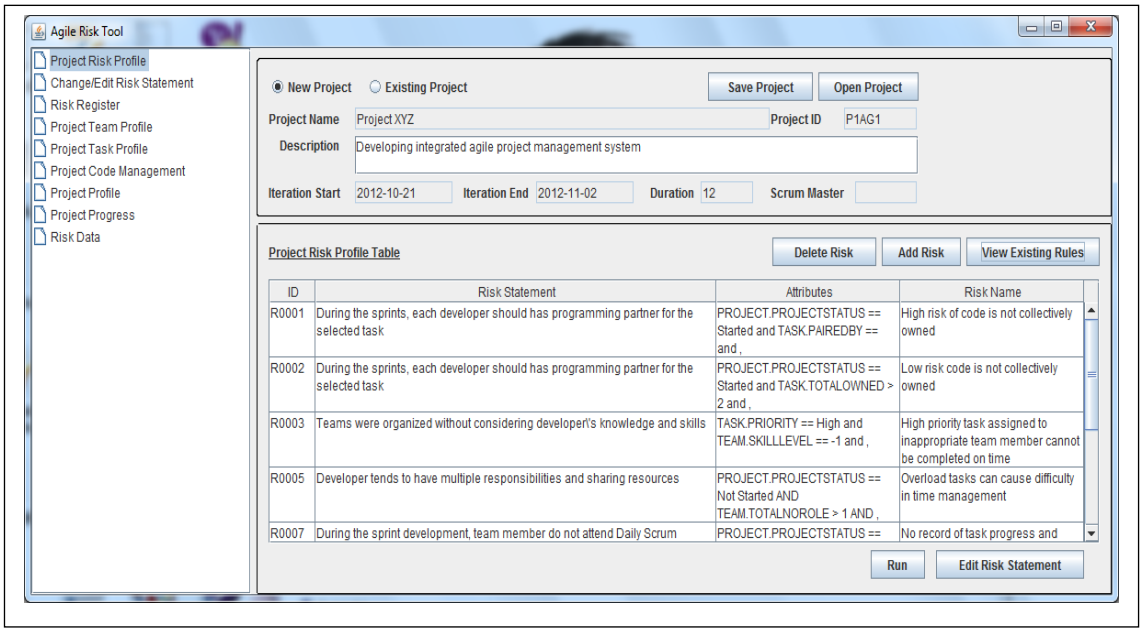


Figure 4-15: Agile Risk Tool - Rules Added on the Main Screen

Besides using the embedded rules to monitor risk in the project, one can also add or modify existing rules from the main screen. It is advisable, before adding or modifying

the existing rules, to be familiar with the tool and have detailed knowledge on the process of generating rules which includes the following:

1. Defining the risk statement and the conditions of the rule associated to the risk; and
2. Assessment of the probability and impact of the risk.

Figure 4-16 below shows an example of adding new rule in the screen.

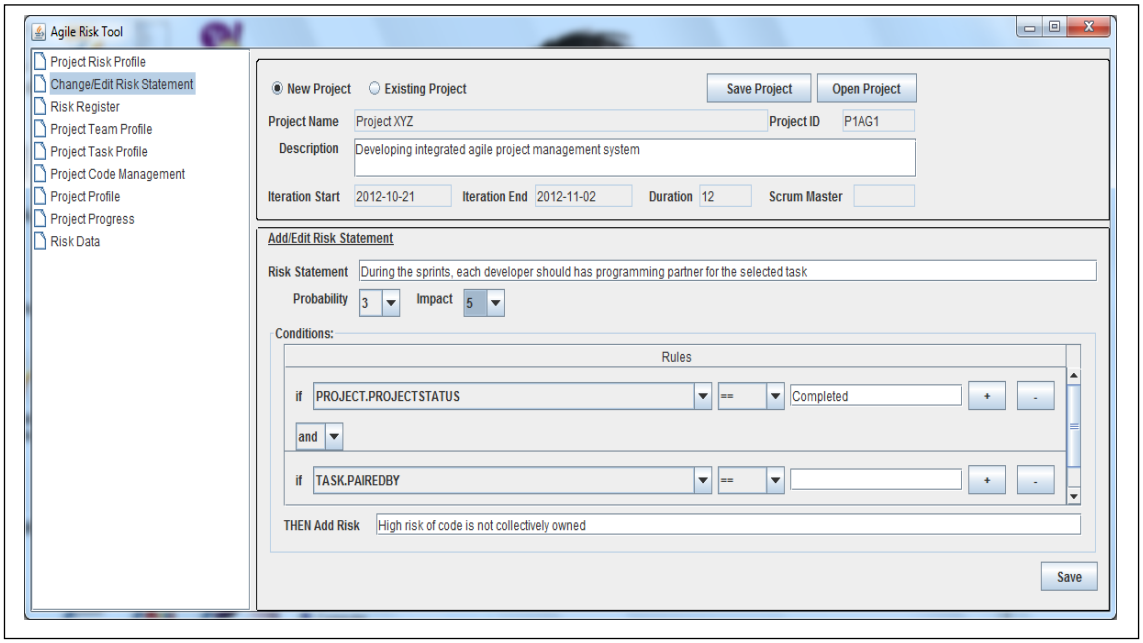


Figure 4-16: Agile Risk Tool - Adding New Rule Screen

1) Risk statement and the conditions of rule

Identified risks are described and communicated to management in the form of risk statements. A risk statement provides the clarity and descriptive information required for a reasoned and defensible assessment of the risk's occurrence probability and areas of impact. There are two essential components:

i. Description of the *condition* that exists and the circumstance or situation that is raises a concern; and

ii. Description of the *consequence* that may result from the current condition.

There are a number of constructs that may be used in developing the risk statement, but the preferred construct is as follows: *"Given that a condition exists, there is a possibility that a consequence will occur"*. Figure 4-17 illustrates the components for the risk statement as cited in SRE method (Williams, 1999).

Components of Risk Statement:

Given a **Condition** → there is a probability → of **Consequence**

Condition is defined as an event that identifies possibility of problems to occur in future; *Consequence* describes the result of undesirable or negative outcome of the current condition.

Figure 4-17: The Components of Risk Statement as described by Williams (1999)

For this work, the derivation of a risk statement is important due to the fact that one can use it to identify the situation of the risk and the result of which the risk could happen. Once a risk statement is developed, one can select appropriate environment data attributes that are related to the risk. The environment data refers to the data that was defined earlier in the ART model which consists of Project, Team, Task, and Progress. The attributes are then used as a condition in the rule engine to detect risk. The Table 4-6 below shows an example of a risk statement, attributes and its associate risk. Note that the attributes for the given rule is rather specific and relates to the conditions given in the risk statement i.e. they are tightly coupled. This is also means that the developed risk statement is under observance and once this has been input into the tool, the risk cannot be missed, unlike a more traditional risk management approach where an identified risk might be overlooked or ignored in the middle of the project. In the real project, prior to the start of the project one can include a brainstorming meeting in the planning phase in order to come out with a list of risk statements for the project.

Table 4-6: A Risk statement example that contains attributes for the rules, probability and impact and risk name

Risk Statements (Condition - Consequence Statement)	**Attributes for rules**	**Probability***	**Impact***	**Risk Name**
Given that a high priority task is assigned to a low level skill team member, it will cause the task not to be completed on time	Task.Priority = = High Team.SkillLevel = = 1	Scale from (1-5)**	Scale from (1-5)**	Incomplete task
*Standards Australia and New Zealand (1999) ** 5 = Very High, 4 = High, 3= Moderate, 2= Low, and 1= Very Low				

2) Assessment of probability and impact of a risk

In the case of defining the risk severity of the risk identified, a matrix can be set up with the axes of probability and impact in order to calculate the risk severity. There were many selections of risk matrix available in the literature ranging from scale as low as 3x3 (e.g. High/Medium/Low) and as high as 5x5 (e.g. Very high / High / Medium / Low / Very low). In this work however, the risk matrix from the Standards Australia and New Zealand (1999) was adopted due its international use (Raz and Hillson, 2005) (see Appendix B).

Further in this step, one may need to define the Probability (P) and Impact (I) in order to assess risk severity for each risk. As stated earlier, the P and I scales can use terms appropriate to the organisation in question e.g. "Very Low", "Negligible", "Very High", "Extreme" etc. Selecting the appropriate P and I might be difficult in the beginning because one might not understand the specific metric values for indicating the probability of the event to happen and the impact if this is to happen. In this case, one needs to formulate the parameters of the P and I for the first instance and use the overall approach to iterate and improve the selection over time. The ART agents will react dynamically to any changes of these parameters.

Once both inputs are entered into the tool, the tool can be run in monitoring mode. The ART agents will then react towards the environment data using the rules to provide risks results when any risk is triggered.

4.3.2 Process

At the Process level, the ART agents will monitor the risk by acknowledging any rules or risk indicators triggered as informed by the ART template. The ART agents will initiate communication between them. Messages are passed according to request and each agent will notify another agent in prompting any further action to be taken. An example of the ART agents' communication was introduced earlier in this chapter (Figure 4-6).

Figure 4-18 below show an interaction between the Art agents starting when a risk is triggered. The figure shows the agents passing message using Sniffer agent in the JADE platform. True to its name, sniffer agent is a purely java application that tracks messages in the JADE environment. It is useful when debugging the agent behaviours and for analysing message passing using in the sniffer GUI (Bellifemine et. al., 2007).

Rules and the environment data are dynamically editable. In the event where changes need to be made, one can modify the environment data (which has been translated into the ART template earlier) as well as the risk rules and indicators using the provided main screen area. On the other hand, when developing possible risks associated with rules and risk indicators, one might find the environment data used to be insufficient to detect certain risks. In some cases, a small change in collection of the environment data would allow defining or detecting more risks. For example, adding the information on developer's skill will allow monitoring the developer's programming capability especially in completing high priority task. An example of a rule syntax that can be used is, *"IF the developer skill level is 'Low' AND the developer involved with a 'High' priority task, THEN there is probability a risk of the task cannot be completed on time because of the developer's poor programming skill"*.

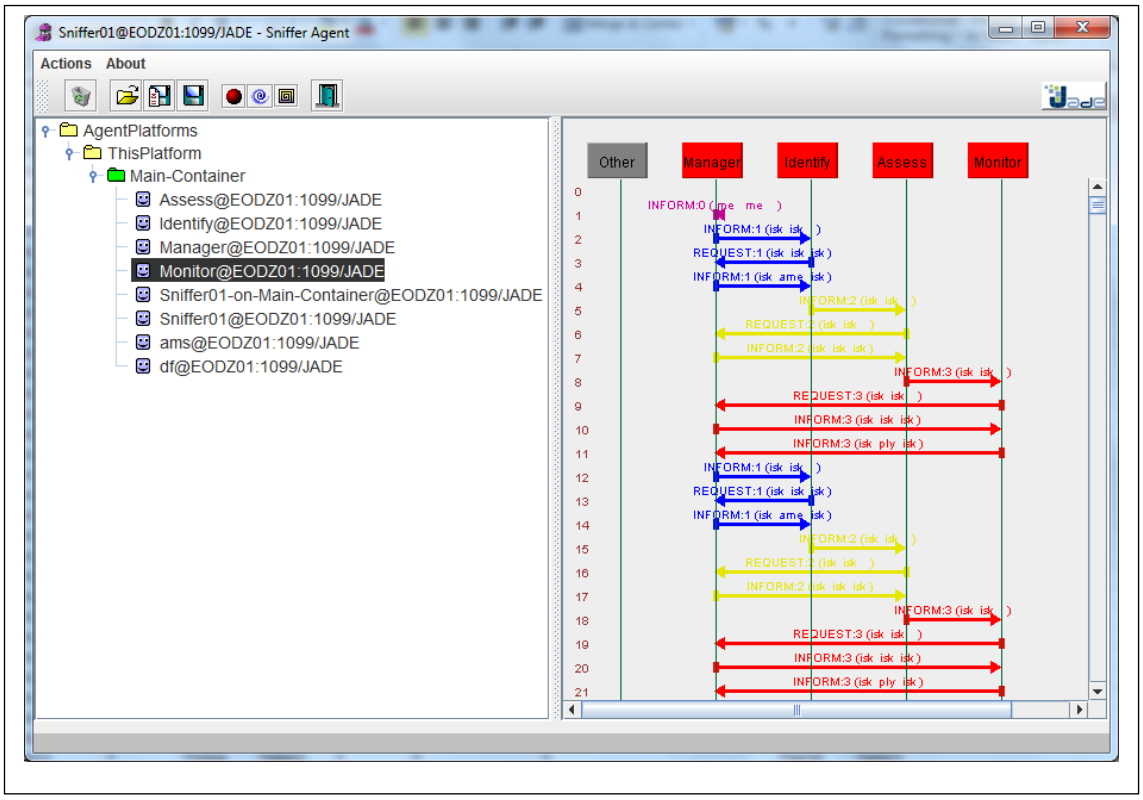


Figure 4-18: Sniffer Agent

ART agents will react dynamically to input data, process the input by assessing any risk triggered and produce a risk result in the Risk Register.

4.3.3 Output

The idea of a Risk register has been defined by Williams (1994) who states that *"the risk register has two main roles. The first is that of a repository of a corpus of knowledge… The second role of the risk register is to initiate the analysis and the plans that flow from it"*. While Patterson and Neailey (2002) reported that very few development and construction of risk registers although it is commonly used in Risk Management. As such, risk register developed in this work can represent as a risk dashboard in which one can see a list of risks triggered by the ART agents. The Figure 4-19 shows an example of risk register in this tool.

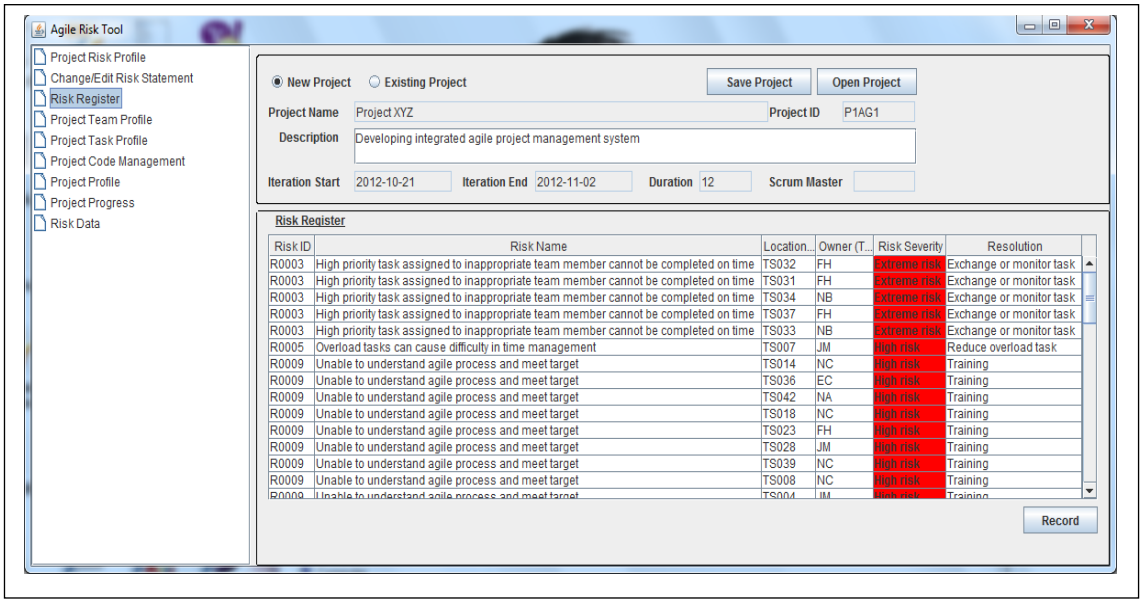


Figure 4-19: Agile Risk Tool - Risk Register

Finally, the risk data can be stored in the Risk data repository by selecting the [Record] button. The risk data also can be captured daily up to the *i* no. of days in a sprint and the data will be saved in spreadsheet format. Later, one can use this risk data, analyse the risk according to project and use the analysed data as an input for identifying future project risk as shown earlier in Figure 4-11.

As discussed earlier in Section 1.6, this tries to support Continuous Risk Management (CRM). Applying the ART process accompanied with the designated tool will help the project manager to manage risk continuously. This is where, when changes take place in the environment data, these are captured by the ART agents who constantly run updates on the risk data and display the results in the risk register. As far as the CRM is concerned, manual implementation of this technique can be minimized and the monitoring is moved towards being autonomous.

In conclusion, this chapter presents the basis of the main contribution of this thesis, the ART development model and its process and tool support. The ART prototype has been walked through and the conclusion can be made that there is initially an extra effort needed for initiating the tool for the first time especially. However, once this process is completed, future efforts will require less effort. Risk is now can easily be managed via the Process and Output stages with a move away from carrying these out manually, helping to reduce the human effort required. Generally, the implementation of this solution approach can be seen as monitoring specific risks related to adherence to the Scrum methodology. The risks however, need not be limited to Scrum compliance and so risks arising from other sources could be added in the future.

The following chapter describes how the ART prototype tool and process have been validated using a case study.

Chapter 5 - Methodology and Case Studies

5.1 Introduction

The goal of this thesis is to build a novel and reliable model supported by a tool to help manage risk in agile projects. As part of this, two case studies were conducted to validate the feasibility and applicability of the approach. The first case study is named as Case Study Alpha (CSA) and the second case study is named as Case Study Beta (CSB). The data for both case studies was collected a posteriori and each case study design is discussed in the following section. The case studies presented four interesting sets of results associated with the research questions developed in Chapter 3 and as repeated in Figure 5-1. The outcome of CSA has yielded a result that refer to as Total Risk Score (TRS) as well as some additional data that has been gathered. TRS refers to a metric that is used as an estimate of the total risk exposure in a project. The detail of TRS is discussed further in Section 6.2 and the results can be found in Section 6.3.2 and 7.3.2. The additional data gathered refers to a sub section on qualitative study done that covers the work to detect common oversights in practicing agile methods, as discussed in Chapter 3, using the collected artefacts. This result is presented in both case studies specifically in Section 6.3.3 and Section 7.3.4. CSB has also yielded results for Total Risk Score (TRS) and the additional data gathered, but also Risk Factor Points (RFP). RFP is the calculation of risk based on the weighted risk factor and the size of the task

affected by the risk. This is discussed in Section 7.2. These results collectively contribute to identifying useful insights on agile practices.

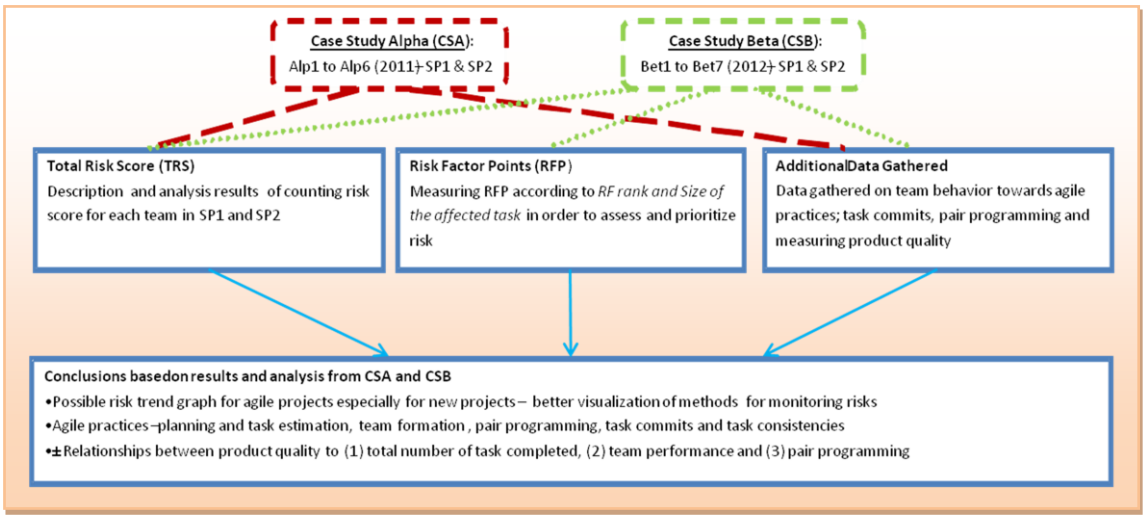


Figure 5-1: The summary of outcomes from Case Study Alpha (CSA) and Case Study Beta (CSB)

5.2 Selection of Investigative Methods

In order to evaluate the solution approach, there are five basic approaches that are relevant to software engineering; controlled experiments, case studies, survey research, ethnographies and action research (Easterbrook et. al., 2008).

A *controlled experiment* refers an investigation where hypotheses are developed and tested; where changes in one or more independent variables have an effect on dependent variables. Yin (2002) introduces the *case study* as empirical investigation that offers an in-depth study of a phenomenon. This includes exploratory study that is used as an initial investigation to derive new theories of the phenomena while confirmatory study is used to test the built theories. *Survey research* is suitable where the purpose is to recognize the characteristics of a broad population of individuals which usually related to the use of questionnaires for data collection. *Ethnography* is a form of research focusing on studying a community of people regarding their social interactions. For example, how a group of programmers build a culture of practices and effective communication between them. *Action Research* is where an attempt is made to get involved and influence a real-world environment, concurrently studying the existing environment and making improvements based on the studied situation.

This research work has developed in a context where the nature of the study is exploratory and has involved an evaluation of a tool after it has been used in a project. It focuses on investigation of problems in the studied areas and then develops a solution approach. Later, an initial study is developed in order to build some theories and further study is developed in order to support or refute the theories. In such situations, a case study is the preferred means.

5.3 Case Study Methodology

As addressed by Kitchenham et al. (1995), choosing appropriate methods for a study is an important aspect and each method yields different environmental designs, analysis techniques and conclusions. Case studies are an empirical method aimed at investigating contemporary phenomena in their context (Runeson and Höst, 2009). Yin (2009) states that *"..case study should be used when a how or why question is being asked about a contemporary set of events over which the investigator has little or no control*". They can employ either quantitative methods, where numeric data regarding a specific entity or situation is collected, or qualitative techniques that involve words, pictures or diagrams. Case studies are normally used in Software Engineering to monitor projects, or particular activities within the software development lifecycle (Wohlin et. al., 2003) and can be used on an exploratory basis, or seek to investigate relationships between variables of interest.

Due to the fact that the nature of the study was rather exploratory and involved a large number of variables, a flexible and natural method is needed. Easterbrook et. al. (2008) declared exploratory case studies to be suitable for preliminary investigations of some phenomena to develop new hypotheses and build theories, and confirmatory case studies to assess the existing theories. Case studies can provide a deeper understanding on the subjects under study and the results obtained are valuable as well as contributing to the body of knowledge (Kitchenham et al., 1995). Besides, the underlying analysis and results of the case study is valuable and can be used to build upon for a future project.

Easterbrook et. al. (2008) promotes case study research where it uses the purposive sampling, thus depending on the nature of the research objectives. A case study is suitable when research is required in order to gain deep insights into chains of cause and effect. Furthermore, exploratory investigations are appropriate where there is little

control over variables in the study. It started with the investigation into issues and problems in risk management in agile projects in order to gain deeper understanding of the phenomena followed by the construction of set of Research Questions and a proposed solution approach. Later, the solution approach is validated using a developed prototype tool using the case studies CSA and CSB. The first case study, Case Study Alpha (CSA) was considered necessary to explore the problem domain and was rather preliminary in nature, mainly intended to develop the Research Questions set up earlier. The second case study, Case Study Beta (CSB) was conducted as a confirmatory case study and was used to assess the existing theories and results developed from CSA. However, both case studies aimed to provide validation of the solution approach and the tool support with improvements being made in CSB based on the lessons learned from CSA. Mixed methods were used including (i) an informal interview with the Product Owner to validate results from the prototype tool and (ii) artefact or archive analysis was done to understand compliance with agile practices in the team as well as to demonstrate the outcomes generated from the prototype tool. The artefact or archive analysis refers to the investigation of project data which includes the Agile Project Management tool spreadsheet they used to collect data, minutes of the meeting and SVN repositories in order to identify patterns in the behaviour of the development team. This analysis can be found specifically in Section 6.3.1 and Section 7.3.4 as part of the study results and analysis chapters.

Darke et. al. (1998) not only emphasized the issues of biases that come from the researcher's beliefs but as well as the behaviour of participants in the study, setting out what can influence the analysis of the case study evidence. Further to that, Walsham (1995) strongly argued that by sharing the concepts and interpretations of the study with the participants on site has indirectly encouraged bias. Participants in that case study were trying to produce high quality software and trying to adhere to agile principles. Measurements made in the study did not affect their marks. Additionally, these issues can be counteracted by collecting data from different sources (Miles and Huberman, 1984; Yin, 2009). Easterbrook et. al. (2008) describe a strategy called 'Concurrent triangulation strategy' due to the fact that often 'what people say' could be different from 'what people do'.

The proposed approach was expected to be used as an initial step to manage risk in a project that uses an Agile Project Management tool. The nature of the study involves identifying threats to a project and assumes that any violation of best practice can

trigger a risk. The nature of a case study is to build it around the studied environment, leading to indirect bias (i.e. in terms of selection of practices and changes to developers' behaviour) towards the desired practices, thus forming a threat to the internal and external validity of the study. To counter this, one can refer to the fact that the developers had no idea that their project data was to be evaluated for risk management. If they had they would have been strongly biased towards following the desired agile practices. In reality however, the aim is to see how the proposed approach fits into the project environment and at the same time to validate the outcome of generated risk rules derived from the project goals. In addition, it is claim that the proposed approach, when used correctly and in the right project, can produce valuable insights to agile projects in managing risks. This is also to answer Research Question RQ2b: Can data collection be conducted with minimal intrusion and effort?

The case studies were carried out on groups of students who were tasked with a software project and used their data as an input to the proposed approach. Many practitioners in agile projects claim that agile methods inherently reduce risk (e.g. Cohn, 2005; Boehm and Turner, 2005) but to the author's knowledge very little research has been done to confirm this or in relating these two areas. However introducing something new in an organization is difficult and likely to be costly. Menzies et. al., (2009) use the term 'data drought' to describe the situation where there is an unwillingness from organizations to share their operational data due to, among other factors, business sensitivity associated with the data. Given the difficulties of obtaining industrial data coupled with the ready availability of student project data in a university setting, student group projects were used in order to demonstrate and validate the approach. On the other hand, the use of student project data has strengthened the approach due to the access to a unique data set which is not be available in any other setting. This includes the access to data for a set of parallel agile teams all carrying out identical sets of user stories. This scenario would be very difficult to find or engineer in a similar study elsewhere, and virtually impossible in an industry setting.

Prior to conducting the case studies, the study was vetted to ensure ethical requirements were met. Ethical approval was obtained from the ethics committee of the School of Electronics, Electrical Engineering and Computer Science, Queens University Belfast. Since the study involved an interaction with one respondent, the Product Owner / Educator, the nature of the study was explained beforehand and assurance given that any sensitive data would be held in the strictest confidence.

There were two goals of the case study aimed to address the Research Questions developed earlier:

- To validate the approach and tool support. This involves assessing the possibility of using collected environment data from agile support tools like Rally Software and Extreme Manager. Likewise, the rules and risk drivers were generated based on the generic list of problems and issues identified in agile projects earlier. This is to answer Research Question RQ3a: Can software agents coupled with a rule engine provide a means to automate risk management in agile projects using data from the Software Development environment?

- To provide useful insights that contain valuable information on the current agile project practices that correlate to the perceived Total Risk Score. The results were analysed quantitatively since existing data was used and the collection of data for study was conducted a posteriori. This goal aimed at answering Research Question RQ1c: Do the findings of the application of risk management in agile projects give useful insights towards an improved agile process and risk management?

5.3.1 Case Study Design

The study used data from a final year undergraduate honours course in Agile Methods, taught at Queens University Belfast in the years 2011 and 2012. In the theoretical part of the course, students received lectures on general agile development practices with an emphasis on Scrum. During the course, students were required to build a large software artefact using Microsoft.NET technologies using an industrial strength environment adopting both agile project and software engineering practices. This includes applying important Scrum project management practices such as Pair Programming, Test Driven Development, Release and Iteration Planning and Refactoring in their software project. The first case study was developed in 2011 involving 38 undergraduate students, assigned into 6 groups with 6 or 7 developers each. All groups were required to develop the same product requirements. The first case study developed in 2011 was named as **Case Study Alpha (CSA)** with groups called Alpha1 to Alpha6 (Alp1-6). The second case study was developed in 2012 which involved a total of 56 undergraduate students with eight groups and each group consisting of 5 to 8 developers. Likewise, all groups were given the same product requirements as in the first case study. The second case study was named as **Case Study Beta (CSB)** with groups of Beta1 to Beta8 (Bet1-8).

However, due to some missing data from group Bet8 this group was dropped from the study, leaving only 48 undergraduate students involved in this study. The projects in both case studies involved two sprints, SP1 and SP2 which respectively had at least 10 to 15 working days. Before the start of the project, the Product Owner, a role played by a member of academic staff, introduced the Product backlog items which consist of a list of prioritized user stories. Thereafter, it was each group's responsibility to deliver these to the satisfaction of the Product Owner. The Product Owner and students were able to communicate regarding any arising issues in both sprints; therefore the groups have been supervised throughout the agile process.

As described in Chapter 4, the case studies were used to demonstrate the ART Process Flow (Figure 4-2) which contains essential stages in Input, Process and Output stage. The most vital part of the process is to determine its environment data and risk rules for the project. This is further elaborated on in the following sub sections.

5.3.2 The Environment Data

At the beginning of this work, two agile project management tools were studied; Extreme manager and Rally software in order to define possible environment data that may be available in a real-world scenario and could be used in this work. The environment data were categorized as follows:

- Project data – contains information about the project i.e. project name, start date and end date.
- Team data – contains information about the team members i.e. skills and experience.
- Task data – contains information on the user stories and breakdown of the tasks associated with the estimated hours.
- Progress data – contains information about the team reports on task progress.

After this information has been obtained from the studied tools, four categories of data described above were focused. Upon starting up the first case study, the ART template was set up for collecting data from these categories. The collected data from the student projects was as far as possible, screened and matched in order to meet the general cases from the studied tools. Even though the student projects did not use any of the studied tools the same environment data was available in their environment by other means.

The summary of the possible environment data that can be collected is simplified in the following Table 5-1. Note that there are two available data items that were not used in this study; Total number of projects involved and also Team skills since the students in the case studies were only involved with one project at a time and all students fulfilled the required skill in programming, which in this case was a need for C# experience. It is envisaged that in an industrial project this data would also be used to identify risks.

Table 5-1: The summary of collected environment data from the student projects in Case Study Alpha (CSA) and Case Study Beta (CSB)

No.	Environment data Attributes	Value	Used	Repository
1	Project ID	Project unique number	✓	Project data
2	Project Status	Not Started, In Progress, Completed	✓	Project data
3	Team name/ ID	Team member name or unique number	✓	Team data
4	Role	ScrumMaster / Developer	✓	Team data
5	Total No of Role Assigned	1 or 2 roles	✓	Team data
6	Total No of Project Involved	1 or more projects	×	Team data
7	Team Skills	Programming (C#)	×	Team data
8	Agile Experience	True / False	✓	Team data
9	Agile Level	Very Good, Good, Average, Poor, Very Poor	✓	Team data
10	Skill Level	5 (highest skill) to 1 (lowest skill)	✓	Team data
11	Task Name/ID	Task name or ID	✓	Task data
12	Task Priority	High, Medium, Low	✓	Task data
13	Paired By	Paired or Not Paired [“ ”]	✓	Task data
14	Total Owned	1, 2 or more developers	✓	Task data
15	Estimated Hrs	No. of hours	✓	Task data
16	Daily Meeting Attended	Yes/No	✓	Progress data
17	Progress Details	Yes/No	✓	Progress data

5.3.3 The Risk Rules

In Chapter 3, the research problems and issues in agile projects were discussed and transformed into a set of problem scenarios which was then presented in Chapter 4 (Table 4-1). Each problem scenario represents a possible risk event that is associated with a Sprint goal for the project. The Sprint goal is important since it can be used to consider how environment data values could be used as indicators of threats to those goals i.e. triggers for the risks. Therefore, it is proposed that risk rules can be formulated using the risk indicators to identify events that cause loss (delay/extra cost/loss of value) i.e. risks, leading to a situation where risk identification can be automated.

Tables 5-2 to 5-5 show the sets of risk rules and risk indicators for each problem scenario as inferred from the issues investigated in Chapter 3 and using the Rule template presented in Chapter 4 (Table 4-3). Earlier, the ART GSRM model was developed that shows the relationship between goal, risk-obstacle and assessment. Using this model, the following Rule template tables show the sets of goals, problem scenarios, rules and risk indicators used in this work. (Refer also to Figure 4-3 and Table 5-1).

Table 5-2 indicates the set of rules and risk indicators for goal G1 where when a sprint is started, an appropriate number of developers should be assigned to the particular tasks. The usage of indicators generally depends on how the project manager wants to signify that the condition appears to be at risk. Two indicators were selected for the goal based on the risk issues highlighted earlier in Chapter 3. Indicator IN1.1 states that once the Project Status was [In Progress] and the Task selected has no pair [“ ”], i.e. null or empty string, then the Risk 0001 – “Pair programming” is triggered. IN1.2 states that once the Project Status is [In Progress] and the Task is owned by more than two developers [>2] then the Risk 0002, “Task ownership”, is triggered. These indicators are then translated into rules RL1.1 and RL1.2 that contain object, attributes and value of the attributes that will be picked up by the agents. The repository section shows where the environment data is involved for the particular risks. As mentioned earlier, the set of indicators and rules can be updated from time to time depending how the project manager decides to identify risk. One such case is the modification of the rule for pair programming which took place between CSA and CSB.

Table 5-2: Rule template for Task Ownership

Goal	**G1: In Sprint X, Task Y should be assigned to appropriate number of developers once Sprint X is started**
Problem scenario	PB1: During the sprints, the developer does not have any pair or has too many programming partners for the selected task
Consequences	Avoiding 'single expert' or too many developers sharing code
Indicators	IN1.1: Project is started and when a task is selected in the sprint , 'task paired by' is empty – indicates high risk IN1.2: Project is started and the selected task owned by more than 2 developers – indicate low risk
Repository/ Data	Project data Task data
Rule(s)	RL1.1: If Project.Project_Status = [In Progress] AND If Task.Paired_By = [""] RL1.2: If Project.Project_Status = [In Progress] AND If Task.Total_Owned > 2
Risk ID & Name	RN1.1: **R0001 Pair Programming** RN1.2: **R0002 Task Ownership**

Table 5-3 below indicates the set of rules and risk indicators for G2 where for assigning team member for the project, a task should be assigned to an appropriately skilled team member. For this work, four indicators associated to the goal were used. Indicator IN2.1 stated that if the selected Task priority is [High] and the task is taken by a developer with Low skill level [1], then the Risk 0003 – "High priority task assigned to low skilled team member so that the task may not be completed on time" is triggered. The assignment of team skill level for this project will be discussed further in Chapter 6. IN2.2 stated that the Project technologies needed is [C#, VB] and if the Team member skills does not match [C#, VB] then the Risk 0004 – "Inappropriate assignment of team members for the project" is triggered. Indicator IN2.3 stated that if the selected Team member has no Agile experience then the Risk 0009 – "Unable to understand agile process and meet the target" is triggered. IN2.4 stated that if the Team member has agile experience and agile experience level is low [Very poor] then Risk 00019 – "Unable to comply with the agile process" is triggered. These indicators are then translated into rules RL2.1, RL2.2, RL2.3 and RL2.4 that will be picked up by the agents. The repository section shows where the environment data is involved for the particular risks. Note that for this work, IN2.2 was not used due to the fact that students were acting as developers and all had the required C# skills. However, in a real world project this indicator would be useful especially when dealing with new projects or hiring new team members where particular skills are needed to accomplish the project. This would enable the project manager to better monitor the risk R0004.

Table 5-3: Rule template for Developer Skills and Experience

Goal	**G2: In Sprint X, Task Y should be assigned to a properly skilled team member**
Problem scenario	PB2: Teams were organized without considering developer's knowledge and skills
Consequences	Inappropriate composition of development teams can result in development taking more time and being more costly
Indicators	IN2.1: Selected high priority task in the sprint shows task taken by a developer with low skill level IN2.2: Selected team member skills for the project do not match with the technology or skills required to develop the project IN2.3: The team member involved in the project has no agile experience IN2.4: The team member involved in the project has very poor agile experience
Repository/ Data	Project data Task data Team data
Rule(s)	RL2.1: If Task.Priority = [High] AND Team.SkillLevel = [1] RL2.2: If Team.TeamSkill ⊂ Project.Technologies RL2.3: If Team.Agile = [False] RL2.4: If Team.Agile = [True] AND Team.Experience = [Very Poor]
Risk ID & Name	RN2.1: **R0003 High priority task assigned to low skilled team member and so cannot be completed on time** RN2.2: R0004 Inappropriate assignment of team members for the project RN2.3: **R0009 Unable to understand agile process and meet the target** RN2.4: **R0019 Unable to comply with the agile process**

Table 5-4 below indicates the set of rules and risk indicators for G3, where when a sprint is started, the developer should focus on one role and one project at one time. There are two indicators associated with this goal. Indicator IN3.1 stated that once the Project Status is started [In Progress] and the Team member involved has more than one role [>1] then Risk 0005 – "Overload tasks can cause difficulty in time management" is triggered. IN3.2 stated that once the Project Status is started [In Progress] and the Team member involved has more than one project [>1] then the Risk 0006 – "High risk in losing resources" is triggered. These indicators are then translated into rules RL3.1 and RL3.2 that contains object, attributes and value of the attributes that will be picked up by the agents. The repository section shows where the environment data is involved for the particular risks. Note that for this work, IN3.2 was not used due to only one project

being present at a time. However, in a real world project this indicator is useful for situations where a developer is committed to more than one project.

Table 5-4: Rule template for Developer Resources

Goal	**G3: In Sprint X, developer should focus on one role and one project at a time**
Problem scenario	PB3: In a project, a developer tends to have multiple responsibilities and shares resources
Consequences	Developer will be overloaded and there is a high possibility of losing resources when one project meets a problem
Indicators	IN3.1: Project is started and developer has more than one role IN3.2: Project is started and developer is involved with multiple projects at the same time
Repository/ Data	Project data Team data
Rule(s)	RL3.1: Project.Project_Status = [In Progress] AND Team.Total_No_Role > [1] RL3.2: Project.Project_Status = [In Progress] AND Team.Total_No_Project > [1]
Risk ID & Name	RN3.1: **R0005 Overload tasks can cause difficulty in time management** RN3.2: R0006 High risk in losing resources

Table 5-5 below indicates the set of rules and risk indicators for goal G4 where, when a sprint is started, the developer should attend the Daily Scrum Meeting and provide updates on work progress. There are two indicators associated with this goal. Indicator IN4.1 states that once the Project Status is started [In Progress] and the Team member involved is absent from the meeting [No] then the Risk 0007 – "Developer absent in meeting meaning a possible lack of engagement in the task" is triggered. IN4.2 stated that once the Project Status is started [In Progress] and the Team member does not provide record of task progress [No] then the Risk 0008 – "No progress report" is triggered. These indicators are then translated into rules RL4.1 and RL4.2 that contain the object, attributes and value of the attributes that will be picked up by the agents. The repository section shows the environment data involved for the particular risks.

Table 5-5: Rule template for Developer Progress Meeting

Goal	**G4: In Sprint X, developer should attend Daily Scrum Meeting and provide Task Y progress**
Problem scenario	PB4: During the sprint development, a team member was absent from the Daily Scrum Meeting and provides no progress on the committed task
Consequences	No record of progress for each task and possibility of employee turnover when team member consecutively did not attend meeting
Indicators	IN4.1: Project is started and developer involved did not attend meeting IN4.2: Project is started and developer involved provide no record of task progress
Repository/ Data	Project data Progress data (Enter data manually)
Rule(s)	RL4.1: Project.Project_Status = [In Progress] AND Progress.Daily_Meeting = [No] RL4.2: Project.Project_Status = [In progress] AND Progress.Progress_Details = [No]
Risk ID & Name	RN4.1: **R0007 Developer absent from meeting possible lack of engagement** RN4.2: **R0008 No progress report**

The generated risk rules associated with goals described above were mapped to goals as follows (Table 5-6). These shorter terms will be used throughout the rest of the thesis. In each case the risk represents a threat to the goal success.

Table 5-6: Mapping goals to associate Risk ID used in Case Study Alpha (CSA) and Case Study Beta CSB)

Goals	**Risk ID**
G1 : Task Ownership	R0001, R0002
G2 : Skills and Experience	R0003, R0004, R0009, R0019
G3 : Resources	R0005, R0006
G4 : Progress	R0007, R0008

5.4 The Case Studies

Given that the approach is novel and still at a research stage, the chosen study was rather exploratory in nature, where it relies on the collection of existing data used in the project as opposed to ongoing data collection in an ideal situation. Where possible, multiple sources of evidence (triangulation) were used, meaning that archival artefacts and informal interviews with the Product Owner were used to confirm findings. For example, after each collection of data an informal interview with the Product Owner

was held in order to verify the interpretation based on the collected data. As mentioned earlier, students had no knowledge that their project data was being be assessed for risk, removing any possible bias in this respect. Rather they were motivated in demonstrating that they had followed agile project management practices e.g. pair programming as taught in classes and producing high quality working software. Based on the data collected in CSA, some issues were found in adopting the approach. These were noted along with conclusions of further discussions with the Product Owner. The outcome from this investigation is discussed in Section 5.3.1.1. Further, one modification was made to one rule, R0001, to be used in the next case study. The rational of doing this modification is discussed in Section 7.2.

The following sub section discusses in detail the ART process flow (Figure 4-10) as conducted in the two case studies. Both case studies consisted of two sprints so that the process flow was repeated iteratively in four instances.

5.4.1 Case Study Alpha (CSA)

As described earlier in Chapter 4, the ART Process Flow consists of three main stages; *Input*, *Process* and *Output*. In this section this process is described in detail.

The *Input* stage starts by gathering all necessary data from the student project artefacts and translating the data to match the ART Template. During the *Input* stage, there were two inputs needed - definition of the list of environment data and definition of the risk rules used in this study. In order to define the list of environment data, two steps were performed.

1) Gathering data

For the purpose of defining the list of environment data, archived artefacts derived from the student projects were used. There were five main artefacts used in this study, summarized as below.

- *Hartmann-Orona Spreadsheet*[6] – this spreadsheet mostly contains important information on the sprint backlog. This includes the breakdown of each user story into tasks, estimated hours and the owner of each task.

[6] www.bryanavery.co.uk/file.axd?file=2009%2F5%2FSprint+Backlog.xls

- *Sprint Backlog Target* – contains a list of user stories and associated points and dependencies.
- *Scrum Minutes of Meeting* – contains information on team attendance and updates on tasks.
- *TortoiseSVN Repositories* – is a subversion[7] source control system that was used by students to manage their project. Each group was required to log their activities and check in all documents related to the project and the source code of the programs.
- *Source Code* – the students were required to use the C# and VB.Net programming languages

Table 5-7 shows the list of extracted data from the archived artefacts of the environment data. A sample of this data is included in Appendix D.

Given the novelty of this study, it was expected that there would be some issues raised while performing it. The first issue that was found at this stage was the difficulty in matching the data designed from the studied tools with the data archived in the student project artefacts. From the studied tools, a large amount of environment data was found that could be used to identify risks, as discussed in Chapter 4. The archived artefacts available however, did not provide as much data as the studied tools. Nevertheless, the archived artefacts contained enough useful information, particularly related to the sprint backlog and the user stories, breakdown of the tasks, details of the developer responsible for a task and so on. In addition, the goal of the study was to demonstrate the approach and tool support, not applicability to every data item collected in mainstream tools.

One issue that had to be overcome was that of missing data in the SVN repository. Although all student groups were required to log their work into the repository, some groups had not done this. For example, all groups were required to record daily minutes of meetings yet some of the minutes were missing from the repositories. Consequently, the Product Owner had to trace this record through other methods such as email archives to obtain the missing data. Similarly, there were some inconsistencies in the format of the minutes of meetings. For example each group should have specified the name of the Scrum Master for each meeting. However, this information was missing in

[7] http://subversion.tigris.org/

some of the groups. Again, the Product Owner had to retrieve through email archives and provide this information. All issues found were recorded and written in the investigation notes so that the process could be improved in the future.

Table 5-7: The environment data extracted from the student project artifacts

No.	File Name	Data Available / Collected
1	Hartmann-Orona Spreadsheet	Sprint Backlog Information: Major Task Area / User stories Task Name Task Owner Task Status (Completed/In Progress/ Not Started) Estimated Hours Commits days and hours for each task Team Member Information: Team Member Name and Initials Working Days for this Sprint Working Hours for this Sprint Start Date End Date No. of Calendar Days Sprint Team Member Daily Activity Team Burndown Chart Team Member Burndown Chart
2	Sprint Backlog Target	List of User Stories Points for each User Stories Dependencies
3	Scrum Minutes of Meeting (Daily)	Name of the Scrum Master for the day Work Progress for each Team Member: Work Done since Yesterday Work Planned Today Problems
4	TortoiseSVN Repositories	Directory and Files Versioning Commit Files/Code Track Changes
5	Source Code (C#) integrates with Resharper	Group Project Source code Code Quality Analysis

2) ART Data Translation

Once all data had been defined and organized into its categories, the next step was to translate this data manually into the ART template. Figure 5-2 shows an example of the translation of the data from the archived artefacts to the ART template. This includes transforming the raw data obtained from the artefacts in the form of object oriented concepts. This is essential so that the ART agents will be able to pick up the data and match this with the rules embedded in the tool. Since this study was done after the project had ended, the data obtained was comprehensive starting from day one in sprint SP1 until day 15 in sprint SP2 and ready to be translated into the template. In the event where the project is new or 'fresh', data can be keyed or added directly through the user interface. Similarly, there were also issues highlighted while doing this step. The main issue identified was associated with the process of translating the raw data into the template. Since this was done manually, it involved tedious and highly effort intensive tasks. In this case study, there were six student groups with six different sets of archived data. Even though they might have the same format or almost the same format of the document, the interpretation and explanation of the data was different. To overcome this it was often necessary to carefully go through all the documents in the repositories in order to understand how they implemented their project. Additionally, the problem is compounded in the case of tracking a task assigned to a team member. For example, in the Hartmann-Orona spreadsheet, the team defined the user stories and the breakdown of the tasks. The spreadsheet itself did not provide a unique id for each of the tasks although it did include the name of the person responsible for the task. Whilst in the team minutes for each meeting each team member provided updates on the task they were assigned, this was only briefly described. In the event where which team member committed to a specific task had to be identified and were tallied with the spreadsheet, there is a need to go through the repositories and look at the code commits by the team member. The time consumed was two to three hours for translation of one group's data, excluding time spent retrieving missing data.

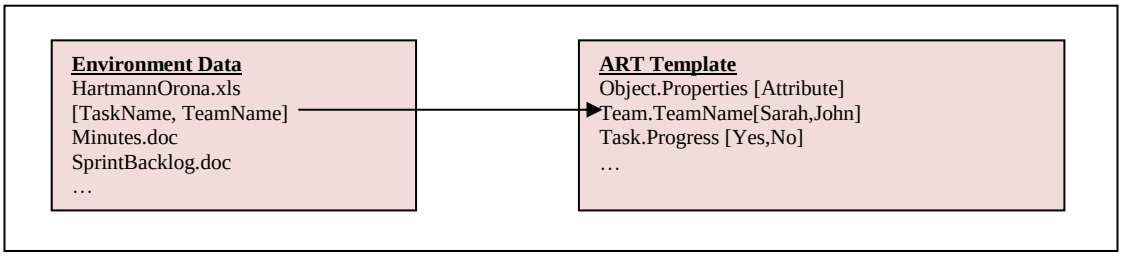


Figure 5-2: Translating environment data to ART template

Next, the risk rules for this case study were defined. Table 5-8 shows the list of risk name and rules as discussed at the beginning of this chapter as well as its probability and impact score as defined for each risk. Note that the rules were embedded in the Rule engine during the development of the ART prototype tool and at this stage existing rules from the Rule engine database had only to be selected. However, when needed, new rules were added into the Rule engine or existing rules edited using the provided user interface. Similarly, when entering the probability and impact the parameters could be adjusted later on. Note that the probability and impact matrix used for this work is based on the Standards Australia and New Zealand (1999) as discussed in Section 4.3.1. For this case, the value of the probability and impact score for each risk was cross checked with the Product Owner. Since this was a student project, there was no actual impact on cost involved for this project. Therefore, the impact factor was based on the consequences of the student not completing the project and not producing a quality end product as required by the Product Owner. In the real world project, the risk identified will be more project-specific, in other words risks are assessed individually for a specific project situation or environment. As discussed earlier in the literature review, a significant project risk can be the result of certain characteristics in the project environment. For example, a developer who is considered to have very low skills but is assigned to a high priority task could lead to a higher risk exposure compared to where that developer is assigned to a lower priority task. In brief, the project manager determines what risk is significant and how severe the risk is.

The *Process* stage is the stage where the risk assessment automatically took place. Based on the defined inputs described previously, the ART agents communicate between the ART template and the Rule engine. At this stage, once the project is loaded into the ART prototype, changes can be made using the provided user interface. Once the tool is 'run', the ART agents will react if any of the rules are triggered and then notify an identified risk. This is further illustrated in Figure 5-3. Any changes in the inputs will result in changes in the outcome of the triggered risk as well. This is because the identified risk can be observed in the *Output* stage and the problem of ignoring a risk is avoided.

Table 5-8: List of risk name along with its associated rules and probability and impact score

Risk ID	Risk Name	Rules [Object.Attribute] == [Value]	Prob Score	Imp Score
R0001	Pair programming	PROJECT.PROJECT_STATUS = Completed, TASK.PAIRED_BY = “”	3	5
R0002	Task ownership	PROJECT.PROJECT_STATUS = Completed, TASK.TOTAL_OWNED > 2	1	1
R0003	High priority task assigned to inappropriate team member cannot be completed on time	TASK.PRIORITY = High, TEAM.SKILLLEVEL = 1	5	4
R0005	Overload tasks can cause difficulty in time management	PROJECT.PROJECT_STATUS = Completed, TEAM.TOTAL_NO_ROLE > 1	5	1
R0007	Developer absent in meeting possible of employee turnover	PROJECT.PROJECT_STATUS = Completed, PROGRESS.DAILY_MEETING = N	1	3
R0008	No progress report	PROJECT.PROJECT_STATUS = Completed, PROGRESS.PROGRESS_DETAILS = N	1	3
R0009	Unable to understand agile process and meet the target	PROJECT.PROJECT_STATUS = Completed, TEAM.AGILE = false	3	3
R0019	Unable to comply with the agile process	TEAM.AGILE = true TEAM.EXPERIENCE=Very Poor	1	1

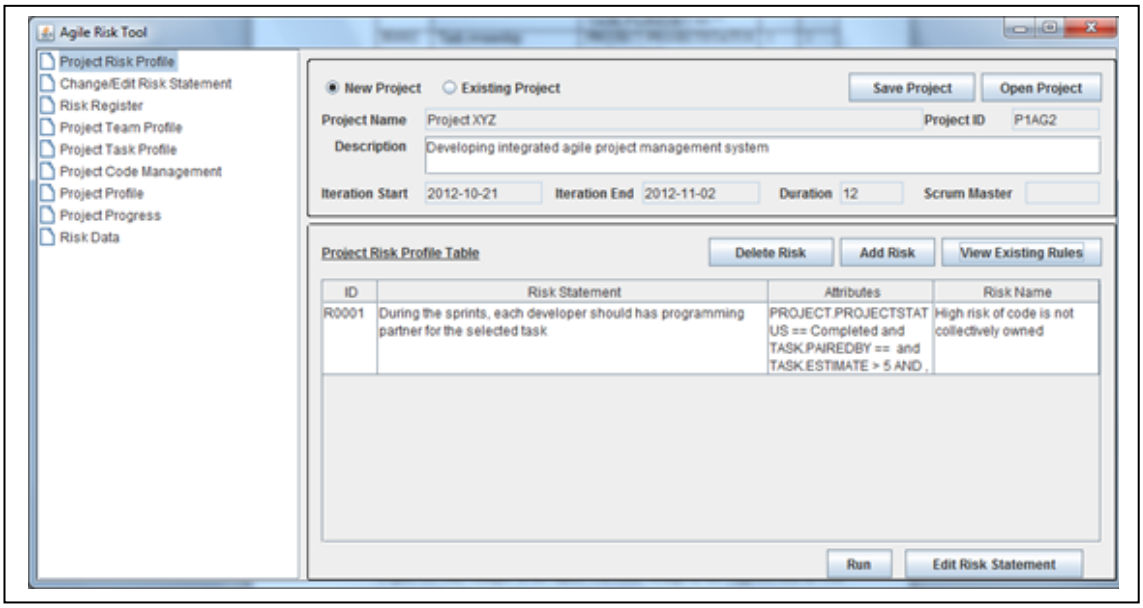


Figure 5-3: Agile Risk Tool – Modifying Existing Rules Screen

In the *Output* stage, once a risk is identified, the risk result is displayed in the Risk Register. This should also show an overview of all risks triggered. This includes the risk name, location of the risk associated with the affected task and the owner of the risk, defaulted to the owner of the task. The risk register also displayed the risk result according to priority, starting from the risk with high to low severity. After one sprint is completed, the risk result is stored into the Risk data repository.

After each sprint in CSA was completed, reports were created. The presented reports provide useful information on the total of risk score each day and in one sprint. This includes information on the breakdown of risk identified each day. Further, the reports provide useful insights in correlating identified risks to agile practices, discussed in detail in Chapter 6 and 7.

5.4.1.1 Outcome for improvements

Using the investigation notes gathered in CSA sprints the following issues were identified.

1) Design of the project

   Since the project was designed for students as part of a university course, the real goal in practice was for students to apply what they had learnt during the course. Hence changing the structure of the project was not possible. Further, due to the limitation of time in completing their project it is not possible to add more management tasks. Nonetheless, how the data can be collected more easily is established based on the investigation notes and the available data in the repository. This supports Research Question RQ2b: Can data collection be conducted with minimal intrusion and effort?

2) Format of the document

   In order to avoid missing data, standard formats were established for documents used in the project; i.e. the minutes of meetings. The structure of the meeting minutes can be found in Appendix F.

3) Naming conventions used in the project

   It is proposed that in order to easily track the data between a task and its owner a unique id for each task and each team member is required. For example, all tasks could start with a unique id beginning with “TS” e.g. TS001 and all team members

could have a unique id starting with "TM" e.g. TM001. This is useful especially when they are presenting reports or updates on project progress.

4) Task allocation and estimation

   Based on the summary of data collected on each group project background, it is found that some groups failed to allocate tasks evenly to each team member. There were some of the team members who carried out too many tasks which resulted in some of the tasks not being completed. Further, it was noticeable that some of the estimated task sizes were too big and should have been chunked into smaller tasks where each task has almost the same size. It is also emphasized that team members should practice pair programming whenever a bigger task is assigned.

5) Specifying student skill level and agile experience

   During sprint SP1 it was proposed that student skill and agile experience should be taken into account. At this stage, identifying student skill was relatively easy, as discussed in Chapter 4. Specifying agile experience is more problematic. In SP1 it is assumed all students did not possess any agile experience but their agile experience was then measured in SP2 based on the assessment by the educator from the first sprint.

6) Risk rule for pair programming

   Risk results were presented to the Product Owner and one of the most common risks found was related to absence of pair programming. It was obvious, from the findings that most of the students did not adhere to this practice. However, the argument was that some of the tasks were too small and were not suitable for work in pairs. As such, it is essential to propose to modify the rule, where the modification being described further in the following case study.

5.4.2 Case Study Beta (CSB)

Based on the lessons learnt from the previous case study, some improvements to data collection were provided and to performance of the students in applying the agile practices learnt in their course. Students were informed of these. At this stage, it is assumed that the Product Owner's awareness of the students' performance has increased based on the lessons learnt from CSA. In addition, it is expected to change the performance of the student group in this case study.

During the *Input* stage, the same steps included in the previous case study were adhered. As described previously, due to the nature of the project we were not able to change the structure of the project and therefore the same artefacts were used and data capture methods from these artefacts.

At the data gathering step the process was found to be much easier than in the previous case study. For example, some of the student groups used the new naming convention in naming their reports and one group used the format of the proposed minutes of meeting. As reported earlier regarding some missing data from the previous student projects and for this case study there was one group who had not logged their main document which was the Hartmann-Orona spreadsheet in the repositories. This document was untraceable therefore there is a need to drop this group from this study. Although the acceptance in changing the working method is rather poor in this instance this can be improved gradually. Similarly, in a real world project it is normal that some organizations might refuse or feel challenged when asked to change their methods. However, due to the limitation discussed earlier it is not possible to compel the students to adhere to standards due to the constraints of this also being an assessment exercise.

As discussed earlier the difficulty in carrying out the process of transforming the raw data into the template during the ART data translation step. Although the effort of performing this step is still quite intensive, the time to translate one group's data was reduced to less than an hour. This was especially the case where the standard naming convention was used in reporting for some groups. In addition, it is agreed that to implement this approach the first time was rather challenging. This was greatly improved and the process would be more effortlessly managed if adopted repeatedly.

Next, the risk rules for this case study were applied. In the previous case study, the list of the risks and their associated rules (Table 5-8) were summarized. Based on the outcomes and lessons learnt from the previous case study, the pair programming rule was modified in order to provide a more realistic risk rule. The ability to modify the rules as needed demonstrate that the solution approach and tool support is dynamically responsive to changes thus, as is required in agile projects. This is described in Table 5-9 which shows the highlighted risk rule as modified. The remaining risk rules were unchanged. This has also supported an answer to the RQ3b question on whether an approach using Agents operating in the Software Development environment useful?

Table 5-9: List of risk name along with its associated rules and probability and impact score (modified)

Risk ID	Risk Name	Rules [Object.Attribute] = [Value]	Prob Score	Imp Score
R0001	**Pair programming**	**PROJECT.PROJECT_STATUS = Completed, TASK.PAIRED_BY = "" TASK.ESTIMATE > 5**	**3**	**5**
R0002	Task ownership	PROJECT.PROJECT_STATUS = Completed, TASK.TOTAL_OWNED > 2	1	1
R0003	High priority task assigned to inappropriate team member cannot be completed on time	TASK.PRIORITY = High, TEAM.SKILLLEVEL = 1	5	4
R0005	Overload tasks can cause difficulty in time management	PROJECT.PROJECT_STATUS = Completed, TEAM.TOTAL_NO_ROLE > 1	5	1
R0007	Developer absent in meeting possible of employee turnover	PROJECT.PROJECT_STATUS = Completed, PROGRESS.DAILY_MEETING = N	1	3
R0008	No progress report	PROJECT.PROJECT_STATUS = Completed, PROGRESS.PROGRESS_DETAILS = N	1	3
R0009	Unable to understand agile process and meet the target	PROJECT.PROJECT_STATUS = Completed, TEAM.AGILE = false	3	3
R0019	Unable to comply with the agile process	TEAM.AGILE = true TEAM.EXPERIENCE=Very Poor	1	1

Once the *Process* and *Output* stage have been completed for the two sprints, again, the reports on the information gathered on the Risk data were created and presented in the form of diagrams. The outcome of both case studies has yielded results of Total Risk Score (TRS) and Additional Data Gathered. However, in CSB, a new result was presented called Risk Factor Point (RFP) that was initiated based on lessons learned in CSA. These results collectively contribute to identifying Useful Insights on Agile practices that will be discussed in detail in Chapter 7.

Chapter 6 - Study Results and Analysis 1: Case Study Alpha (CSA)

6.1 Introduction

This chapter will describe the findings from the first case study (CSA) starting with introducing the Total Risk Score (TRS), the metric used to count risks in this project. This is followed by a presentation of findings and analysis of the risk results produced by the tool as well as the results collected from the additional collected data obtained from the CSA student project artefacts. The aim is not only to use available data to assess risk using the proposed approach but also to define violations of agile practices that could indicate risks to the project. Therefore, in the final analysis, the risk results are used to provide useful insights into how agile practices are being applied as well as to help ensure the success of the project.

The goals of performing both case studies were to validate the model and the tool as well as to provide knowledge on how risk data can be collected without hurting agile principles. Due to the limited scope of the data and small size of the projects, no claim is made that the empirical findings can be generalized to all agile projects, but the study gives valuable insights on agile practices as well as information for the further development of the model. This is also to answer the RQ1c: Do the findings of the application of risk management in agile projects give useful insights towards an improved agile process and risk management?

Using the ART prototype tool for both case studies, risk data was collected for each team involved in the case study. In what follows, these results are elaborated on further.

6.2 Total Risk Score (TRS)

For continuous risk management purposes a metric, Total Risk Score (TRS) is introduced. This is intended to be an approximation of the level of risk for each team in both sprints SP1 and SP2. The TRS was calculated as follows:

Consider that there is a set of tasks *T* in the project on a given day *d*:

$T_d = \{t_1 .. t_n\}$ where n=total number of tasks on day *d*

There is also a set of predefined generic risks *R* that can potentially be identified in the project:

$R = \{r_1 .. r_m\}$ where m = total number predefined risks

Thus, the risks associated with each task, *t* on a given day *d* is $R_{td} \subset R$.
The TRS for a task *t* on day d is therefore

TRS_{td} = card (R_{td}), where card = cardinality of the set

These risks can be associated with any of the tasks *t* in *T*. These risks are present while the task is being carried out in the sprint.

Therefore, for a given day *d* the TRS is

$$TRS_d = \sum_{t=1}^{t=n} TRS_{td}$$

6.2.1 Application of TRS in a current or on-going project

The application of TRS will be different when used in a current or on-going project compared to a past project. In a current project, TRS can be calculated daily. The project manager can see the risk triggers from day one and if it is resolved, the risk no longer appears. Otherwise, the risk can be monitored throughout the sprint cycle at any point in time, including at the end to review how risk was handled. It also allows the

project manager to compare the same risk associated to the same task trigger consecutively. Thus, immediate action can be taken if the risk is not being resolved.

A simple scenario using the calculation described previously is illustrated as follows:

In SP1, Day1, suppose it is identified that a task TS023 (owned by John) was associated with two types of risk R0001 (Pair programming) and R0009 (No agile experience). Therefore, the TRS_1 calculated for Day1 is 2. However, if the risk R0001 (Pair programming) is resolved at the end of Day1, then in Day2, TRS_2 is 1 provided that no other risk is triggered on that day. The risk R0009 (No agile experience) remained as an existing risk until action is taken. In the case studies, the calculation of TRS applied to completed projects being analysed with all of the data in place. The way it was calculated is explained in the next sub section.

6.2.2 Application of TRS in a past project

Due to the fact that the project used in this work had already been completed and that risk reduction was not actively considered in the project, there was no evidence confirming that any risk identified in the sprint had been resolved. Therefore, the calculation of TRS_{td} (on the particular task and day) is accumulated according to the number of days the task taken to complete in the sprint. The aim of the case study is therefore to demonstrate how the approach could have worked with the ART process and tool, retrospectively.

Retrospectively, the TRS for a sprint *s* is then

$$TRS_{sd} = \sum_{d=1}^{d=s} TRS_d$$

where s=total number of sprint days.

Given the same example as described in the previous sub section, the two risks identified in Day1 will be accumulated on the number of days the task is being carried out in the sprint. As in the example below:

Let's say task TS023 was ongoing for five days in SP1 starting from Day1 to Day5; therefore,

$$TRS_1 = 2*5 \text{ days} = 10$$

The rationale for the multiplication is that past project data is used and the project analysis is done a posteriori, therefore the number of days the risk is present is assumed equivalent to the duration of the task. It is also assumed that the risk scored in this work is considered as being present but not being realized by the team member during the implementation of the project until the task has been completed. This is understandable since students were not actively managing risk in their project. This is also due to the fact that the collected data does not include any information about whether the risk has been resolved earlier than the completion time. In a real project, the calculation of TRS can be applied continuously as described in Section 6.2.1. This is where the environment data will be collected as required (or continuously); thus there will be a possibility where the affected task no longer has a risk or an action to resolve the risk has been taken before the task completion time. Therefore, this will eliminate the risk triggered and so the number of days that the risk was present is less than the duration of the task.

The data analysis and results presented in both case studies used the calculation of TRS as in this section. This type of calculation can provide benefits to organizations who would like to use the proposed approach to identify risk from a past project, where the exact information on the risk and whether it is has been resolved or when was resolved is unknown. The risk data obtained can be used as an input using the proposed approach or to predict risk for a future project.

6.3 Case Study Alpha (CSA) Results and Data Analysis

During the planning phase, each team was given Product backlog items that consisted of a list of user stories. They were asked to estimate the effort needed for each user story in hours and break this down into a set of tasks. Hence, each team had a range of tasks in their sprint. At this stage, the students were challenged to plan and estimate the work effort required for each sprint based on the lectures taught in class. Whilst using the Hartmann-Orona[8] spreadsheet to document the sprint planning, each team was required to update their task status as ‘not started’, ‘in progress’ or ‘completed’.

[8] www.bryanavery.co.uk/file.axd?file=2009%2F5%2FSprint+Backlog.xls

6.3.1 Summary of CSA Project Data

Table 6-1 below shows the summary of tasks for each team in the CSA. Based on the data gathered from the student project, the total number of tasks for each team differs, ranging from as minimum as 33 tasks to as maximum as 123 tasks. There was only one team (Alp1) who had a percentage of total completed tasks that was between 40% to 50% in both sprints, while the remaining teams had more than 80% to 100% completed tasks. Two teams (Alp3, Alp4) had an improvement in the percentage of total completed tasks from SP1 to SP2 and one team (Alp5) has had a slight decrease of percentage completed tasks in SP2. On the other hand, two teams (Alp2, Alp6) managed to complete all tasks in both sprints.

Table 6-1: Summary of Tasks for CSA in Sprint 1 (SP1) and Sprint 2 (SP2)

Team ID	**Team size**	**SP1 Tasks**		**SP2 Tasks**	
		Total	**% Comp**	**Total**	**% Comp**
Alp1	6	53	52.8%	123	45.5%
Alp2	6	59	**100.0%**	70	**100.0%**
Alp3	7	56	87.5%	33	**100.0%**
Alp4	6	68	91.2%	72	97.2%
Alp5	7	61	**100.0%**	62	96.8%
Alp6	6	33	**100.0%**	39	**100.0%**

From the table shown above, there is evidence that shows that decomposing user stories into smaller task and task estimation is rather challenging, especially when it is done for the first time. There was a large gap between the total tasks among the team ranging from as low as 33 tasks to 123 tasks despite teams having been given the same set of user stories. This provides evidence for the issue of task estimation identified by Deemer and Benefield (2006), as discussed in Section 3.5. Additionally, among other issues found in agile projects there are problems relating relation to the ability of team members completing their tasks within the estimated time. Therefore, one question arises here is whether the team with 100% completed tasks produced a better quality product? Following from that, a variable was noted, **(V1) Task completion** to be further

investigated for its relationship to product quality. This will be discussed later in Section 6.3.3.3.

6.3.2 Total Risk Score (TRS) in Case Study Alpha

Figure 6-1 and 6-2 below show a plot of the TRS for both sprints. In SP1, the number of risks score ranged from as low as zero (either no risk being present or no ongoing task) up to the highest of 53 risks found in team Alp1, Day9 of the sprint. In SP2, risk score ranged from zero to the highest of 103, also found in team Alp1, Day15. Both graphs show the increasing and decreasing pattern of risk detected for each team with some peaks at certain days which give a better visualization of risk.

The already-established Risk Burndown Chart (Cohn, 2010) discussed earlier in Chapter 2 generally results in a downward risk exposure graph, computed from the changing probability and size of potential losses in every sprint. However, that technique does not show or visualize specifically which risk is being identified, assessed and monitored throughout the project. On the other hand, the graph below may be used to develop a risk trend for a type or a category for agile team. For example, a team who are new to agile projects, one might see risks might increase gradually to a peak and start decreasing towards the end of the sprint as risks are resolved or tasks completed. On the other hand, a more established team might not have any risk occurring daily in their sprint; instead showing a peak at a certain times with regards to the type of risk that they want to monitor.

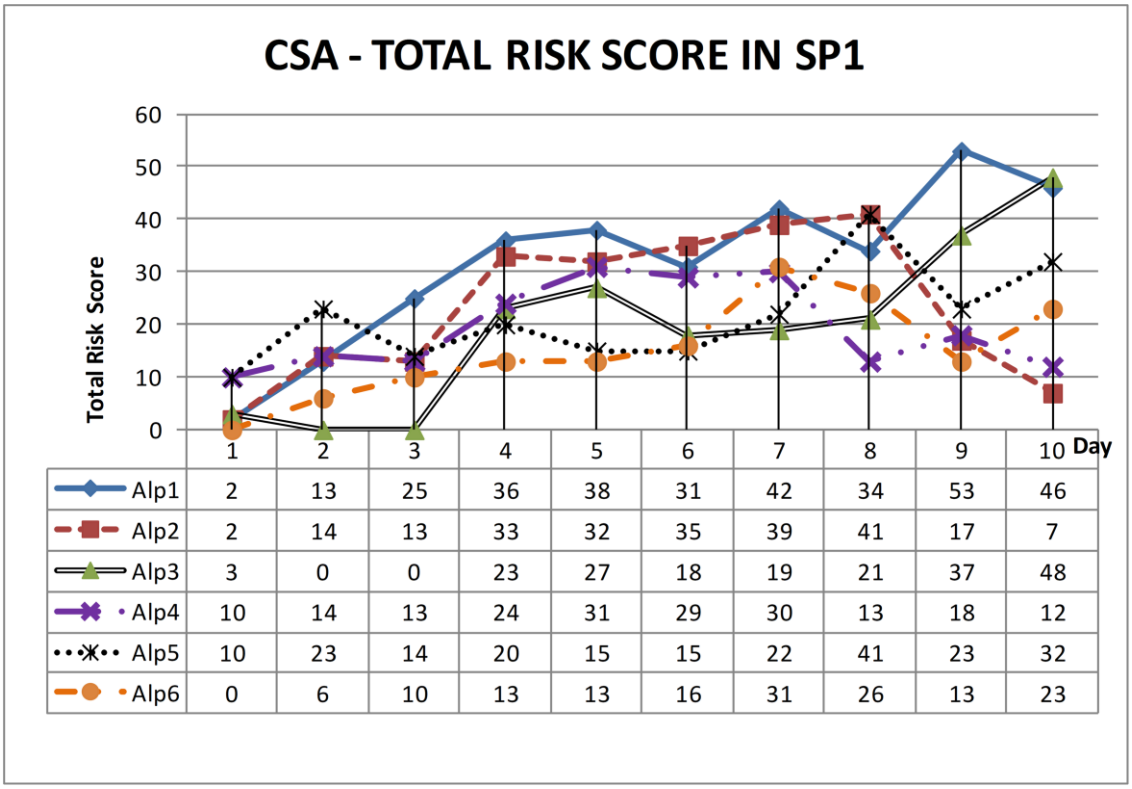


Figure 6-1: Graph of Total Risk Score for CSA in Sprint 1

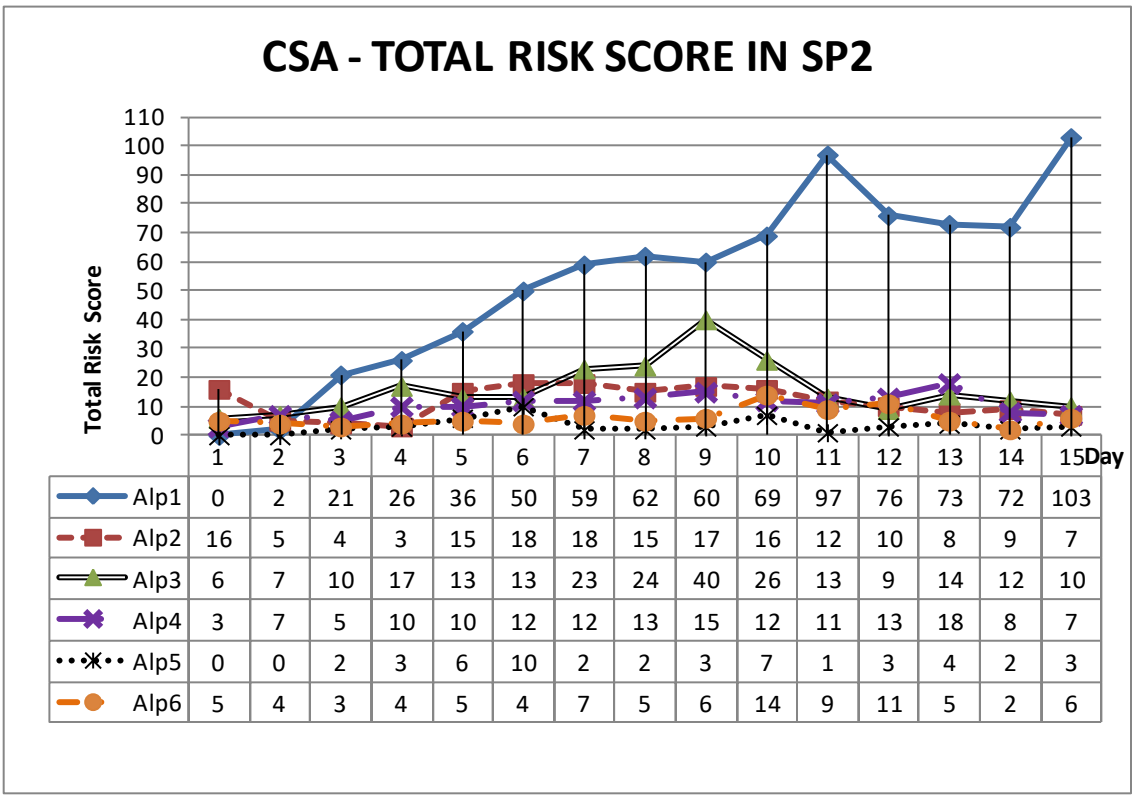


Figure 6-2: Graph of Total Risk Score for CSA in Sprint 2

Schatz and Abdelshafi (2005) emphasized the lessons learnt in which, by consistently focusing and adhering to the agile processes the team will continuously make improvement each day. Figure 6-3 below shows the mean of TRS in SP1 and SP2 where the graph indicates that all teams produced less risk in their second sprint except for Alp1. This result provides evidence that the students gained better knowledge and experience following completion of the first sprint, thus reducing risk over time. Further, above all of the teams, Alp5 was the highest in mean difference of 19. This appears to suggest that team Alp5 performed best in their project in terms of reducing risk compared to team Alp1. However, there is a need to further investigate if the team that performs best (in term of producing less risk) provides a quality end product too? In order to later investigate this relationship, a second variable was identified, **V2 - Team performance** for further investigation its relationship to product quality. This is presented later in Section 6.3.3.3.

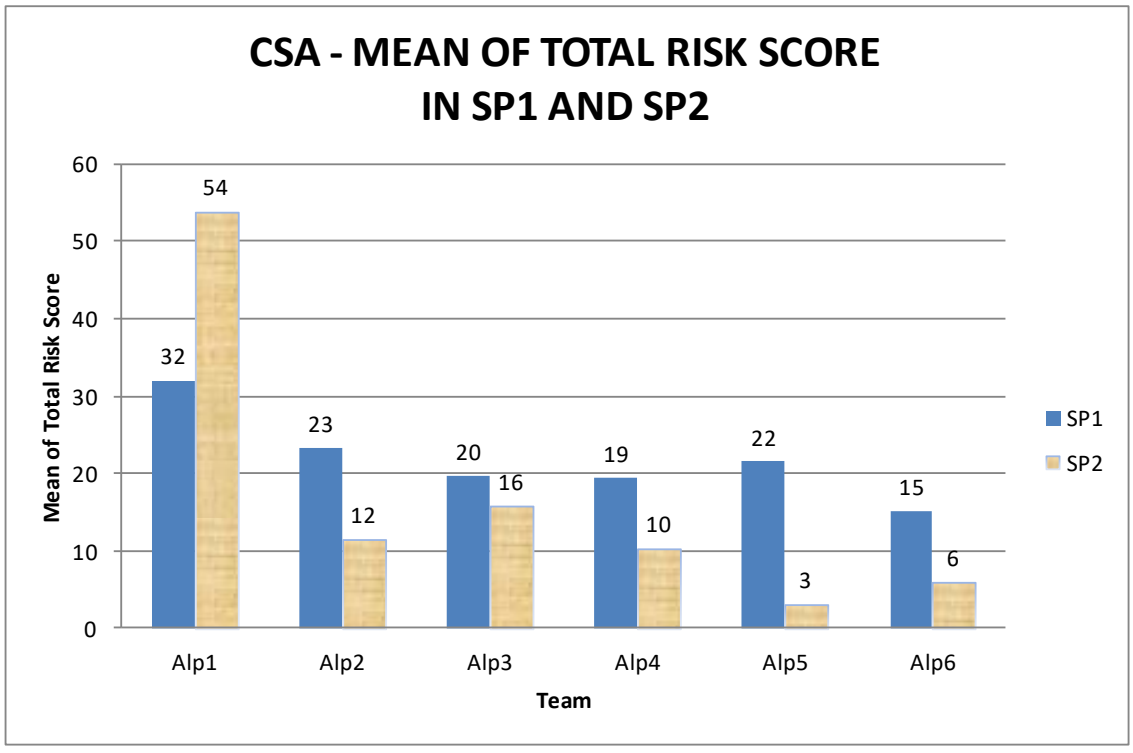
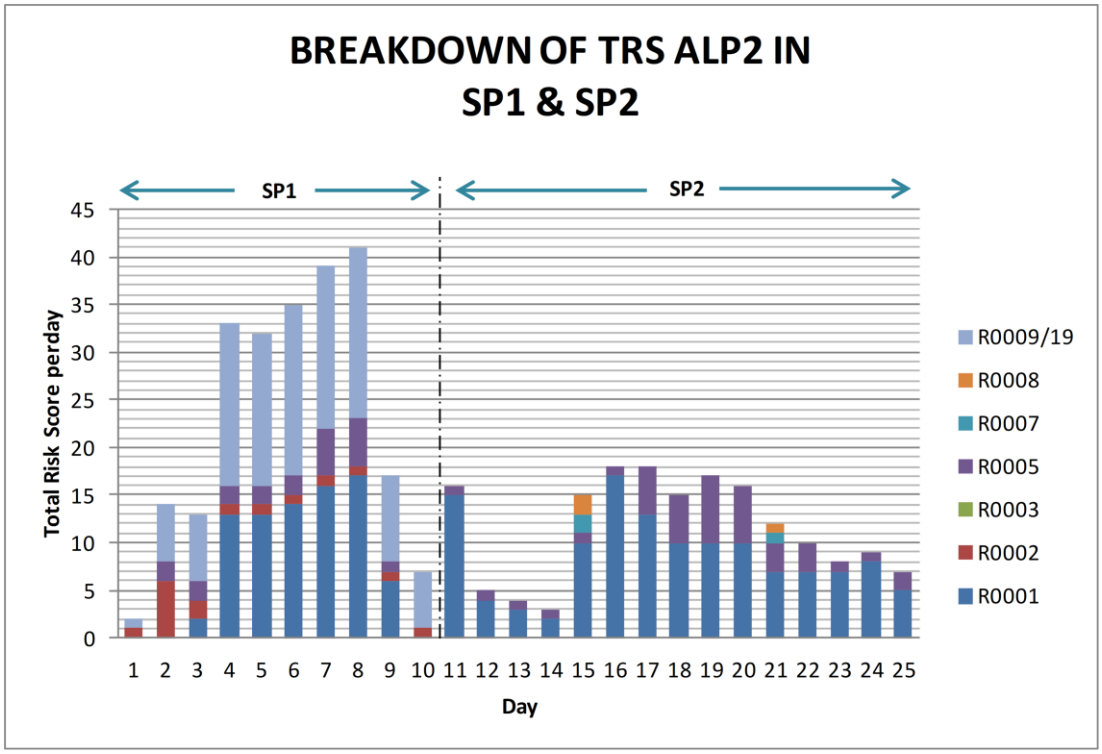


Figure 6-3: Mean of Total Risk Score for each team of Case Study Alpha (CSA) in SP1 and SP2



Risk ID and Risk Name			
R0001 Pair programming	R0002 Task ownership	R0003 High priority task assigned to low skilled team member cannot be completed on time	R0005 Overload tasks can cause difficulty in time management
R0007 Developer absent in meeting possible of employee turnover	R0008 No progress report	R0009 Unable to understand agile process and meet target	R0019 Unable to comply with agile process

Figure 6-4: Graph showing the breakdown of Total Risk Score for team Alp2 in SP1 and SP2

The data shown in Figure 6-4 provides a better visualization of the type of risk triggered each day in team Alp2, for both sprints SP1 and SP2. The advantage of this method was twofold; i) one would be able to view the category of risk triggered for the particular day as well as ii) one would be able to view which risk being triggered the most and needs attention from the project manager in order to help in making decisions e.g. whether prompt action should be taken towards the risk that occurred consecutively. For example, in SP1, team Alp2 had generated risk mainly through not performing pair programming and from having team members without agile experience. However in SP2, since the students had already developed using agile in SP1, the risk of no agile experience no longer existed, but the risk from not performing pair programming could still occur. In the student projects these risks if realized might have very little impact

towards project success but in the real-world project and these risks are triggered without being addressed a threat to the project could arise. On the other hand, patterns of risk occurrence for a particular team member can be inferred and used for estimating future project risks in similar projects with the same team members.

6.3.3 Additional Data Gathered in Case Study Alpha

In the light of the risk issues highlighted in Chapter 3, it is found that it is essential to study the behaviour of the team member in applying agile practices. In addition, a successful agile project depends heavily on the people in the project. Therefore, in this sub section there is a need to investigate team member behaviour focusing on assessing few practices such as how they commit to the given tasks and pair programming. This is to address Research Question RQ1c: Do the findings of the application of risk management in agile projects give useful insights towards an improved agile process and risk management?

To try to answer this, the artefacts obtained in the CSA were used with the ART process and tool support to collect the necessary data as well as to detect common oversights in practicing agile methods in projects.

6.3.3.1 Total Number of Task Commits per day

Since the case studies used in this work employed the Subversion[9] source code control system, the term commits will be used here in the same sense as in the Subversion (svn) glossary[10]. Commits are therefore the communication of changes of files in the directory resulting in a revised version of the file. In the student project, the students were required to update the task commits (task being carried out) on particular days indicating the progress of the task including the day when the task is started until it is completed using the Hartmann-Orona spreadsheet. During meetings with the Product Owner, the students were advised to carry out only one task at a time and they maintained the "task consecutive rule". This is the condition where the student has to complete one task before starting a new task. Figure 6-5 and Figure 6-6 below shows interesting information on each team's task commits in SP1 and SP2. In SP1, tasks were ongoing with as many as 21 tasks per day (Alp3, Day10) while in SP2, the total tasks were slightly higher at 35 tasks per day (Alp1, Day15).

[9] http://subversion.tigris.org/
[10] http://tortoisesvn.net/docs/nightly/TortoiseSVN_en/tsvn-glossary.html

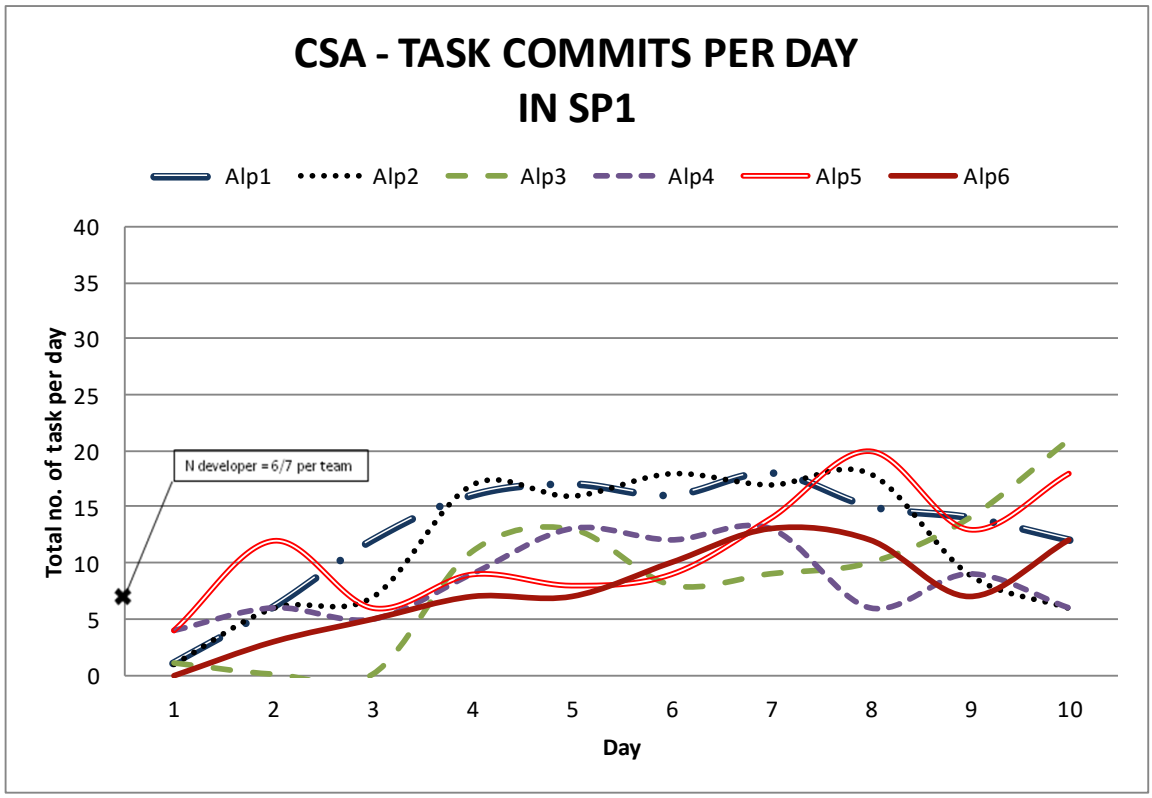


Figure 6-5: Graph showing task commits for all teams of CSA in Sprint 1

Note that the marker drawn on both graphs denotes that the limit of the total number of tasks that should be carried out per day that was either 6 or 7 according to the number of team members in each team. Higher number of tasks commits in other words, has a relation to the high total number of tasks carried out at the same time. In this case, the students were pressured with a high number of tasks that needed to be completed within a short period of time. As a consequence, they carried out too many tasks at one time. This form of practice in a real project would be a threat to the project since too many task commits at the same time could affect the quality of the code produced. This is due to the fact that it is assumed the reason for multiple commits was caused by a rush to perform tasks at the last minute. Moreover, usually the task commits graph should be decreasing towards the end of the sprint however, this is not the case here. The graph in SP1 (Figure 6-5) shows that three teams (Alp3, Alp5 and Alp6) tended to increase commits on the last day of the sprint while in SP2 (Figure 6-6) team Alp1 and Alp6 also show increased commits toward the end. From the analysed graphs for all teams in SP1 and SP2 it is evident that they have not established consistency in the number of tasks being carried out each day. A stand out example can be seen in SP1 team Alp3 whereby this team did not commit any tasks in Day2 and Day3 which has presumably in turn led

to the team committing more tasks on the last day of the sprint. This sort of analysis in an industrial setting would assist the project manager in predicting risk to maintaining a sustainable pace in the project, a requirement from the agile principles[11]. It will also enable the case of a high percentage of incomplete tasks to be flagged up.

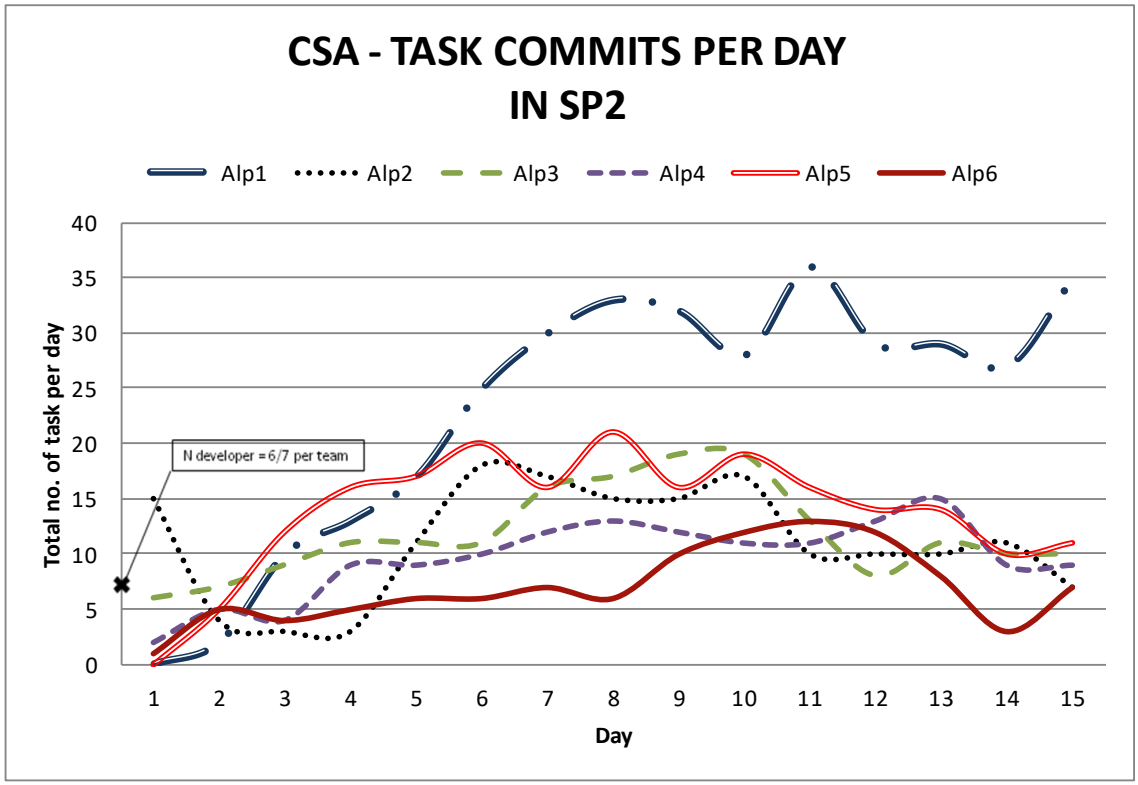
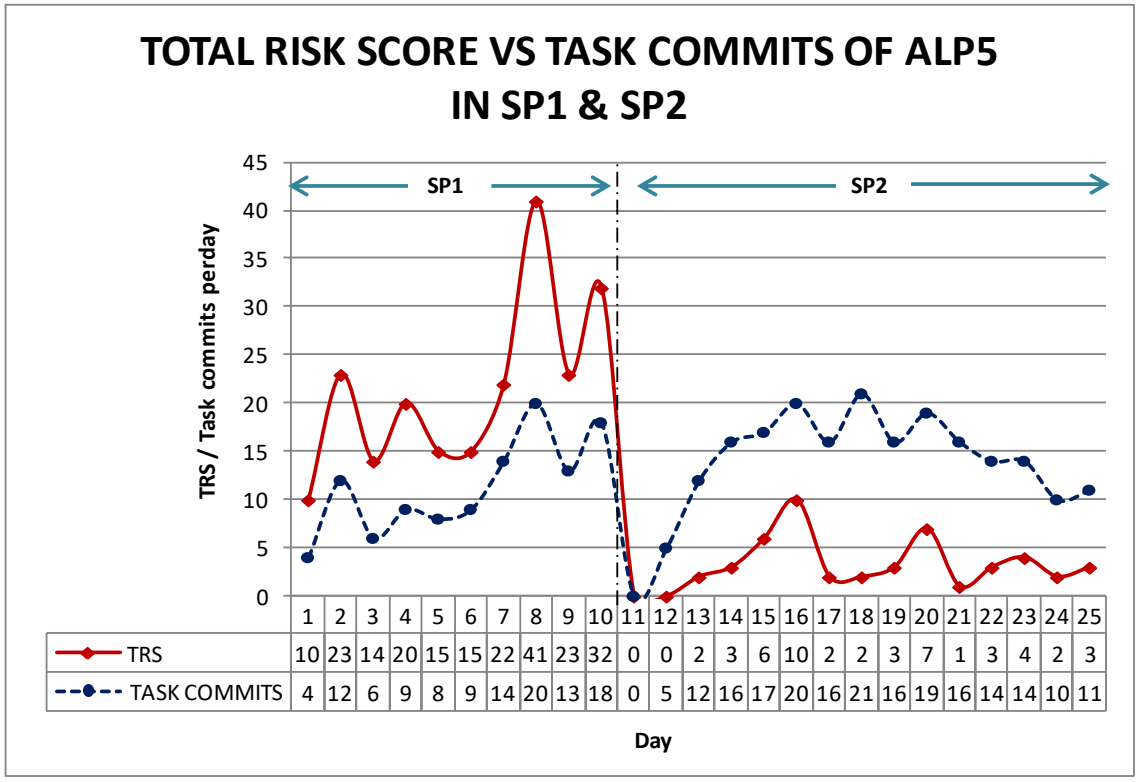


Figure 6-6: Graph showing task commits for all teams of CSA in Sprint 2

From the archived documents of the student projects, the total number of risks counted and the task commits were compared in a sprint as shown in Figure 6-7. The reason for doing so was to investigate the relationship between the risk scored and the task commits and then to compare the difference between the sprints. Furthermore, one of the risk issues discussed in Chapter 3 was related to poor task analysis and the problem of achieving a sustainable pace. In this case, team Alp5 was taken as an example since they managed to make the biggest reduction in the number of risks in SP2. Intuitively, before the start of the project, one could predict that there will be a positive relationship between these two variables, as the more tasks committed or ongoing at the same time will increase risk. This is prone to happen especially to a new project or a team whose members are new to agile practices. Therefore, by using this graph in such a situation will provide additional support to a project manager in monitoring the project risk.

[11] http:://agilemanifesto.org/principles.html

Generally, a new agile project or new agile team or even a team with new team members increases the possibility of uncertainties in agile processes.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
TRS	10	23	14	20	15	15	22	41	23	32	0	0	2	3	6	10	2	2	3	7	1	3	4	2	3
TASK COMMITS	4	12	6	9	8	9	14	20	13	18	0	5	12	16	17	20	16	21	16	19	16	14	14	10	11

Figure 6-7: Graph comparing between risk score and task commits of team Alp5 in SP1 and SP2

The information captured in the graph above (Figure 6-7) gives an interesting insight; during the first sprint of the project the risk increases as the number of tasks increases. However, as the team moved to the next sprint it was the opposite. Evidently, it supports the perception that a new agile team may generate many risks in the first instance, for example, due to poor task analysis when selecting appropriate tasks. However, throughout the process as the team feels more confident and focused on what they are doing, the risks to the project will gradually decrease. On the other hand, if the relationship between these variables works positively throughout the project, it can be taken as a sign to alert the project manager that there is something going wrong with the team. Therefore, early precautions can be taken in order to mitigate the risks.

6.3.3.2 Pair Programming

In the risk issues highlighted in Chapter 3, the problem of team members not complying with agile practices was addressed. Initially, this study focused on the practice of pair programming. Figure 6-8 below shows the results of pair programming application in

CSA for both sprints. In SP1, 68% of the team members fail to adhere to this practice while in SP2 the percentage is only slightly smaller at 66%. However, looking from the perspective of tasks completed, more than half of the team members managed to fully complete their tasks even without practicing pair programming in both sprints. Initially, it is assumed that the student might not practice this in the first sprint due to the fact that they were inexperienced in applying the practice. However, the percentage has not changed in SP2. With this in mind, the investigation on the students' programming skill level was extended in order to get a clearer picture of this issue.

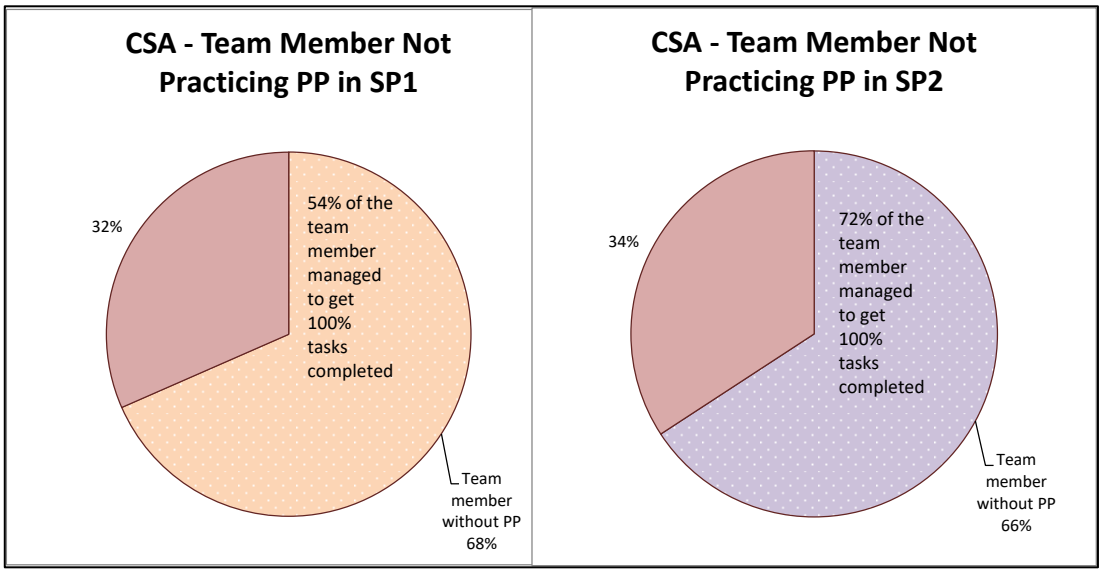


Figure 6-8: CSA – Percentage of team member not applying pair programming in SP1 and SP2

In determining student ability in programming, one of the students' prerequisite courses: "Data structures and Algorithms" was examined, which was almost entirely based on acquiring programming knowledge and skills. The students' final marks were converted from this course into the 'skill level' similar to those proposed by Boehm and Turner (2003). In this study, the level used was revised, ranking from 5 (very good skill) to 1 (very low skill). Table 6-2 below shows the construction of the student marks mapped into proposed levels of software method understanding as used by Boehm and Turner.

Table 6-2: The Revised Table of Levels of Software Method Understanding and Use

Levels of software method understanding and use (Boehm and Turner, 2003)		**The Revised Levels of software method understanding and use**	
Level	**Characteristics**	**Level**	**Student Marks**
3	Able to revise a method, breaking its rules to fit an unprecedented new situation.	**5**	71-100
2	Able to tailor a method to fit a precedented new situation.	**4**	61-70
1A	With training, able to perform discretionary method steps such as sizing stories to fit increments, composing patterns, compound refactoring, or complex COTS integration. With experience, can become Level 2.	**3**	51-60
1B	With training, able to perform procedural method steps such as coding a simple method, simple refactoring, following coding standards and CM procedures, or running tests. With experience, can master some Level 1A skills.	**2**	41-50
–1	May have technical skills, but unable or unwilling to collaborate or follow shared methods.	**1**	0-40

Figure 6-9 below, reports the method in determining the students' programming skill level in SP1. From the total number of 38 students, ten students possess low skills of 1 and 2, six students possess average skills and finally 22 students possess good skills (4 and 5). From this investigation the conclusion is that more than 50% of the students were considered good in their programming skills. This data is of use in determining if students with good programming skills might be reluctant to work in pairs or in the worst case scenario adopted a 'single expert' practice. In a real-world project, this data could be used to predict the risk of a 'single expert' practice and its consequences i.e. producing low quality product or having a lack of team members knowledgeable in part of the code. Further investigation and analysis will need to be done in order to determine the root cause of this issue. Another issue that needs further review is whether the product developed by a 'single expert' has similar, reduced or increased quality? This will be taken as the third variable, **V3 - Pair programming** for investigating the relationship with the quality of the product.

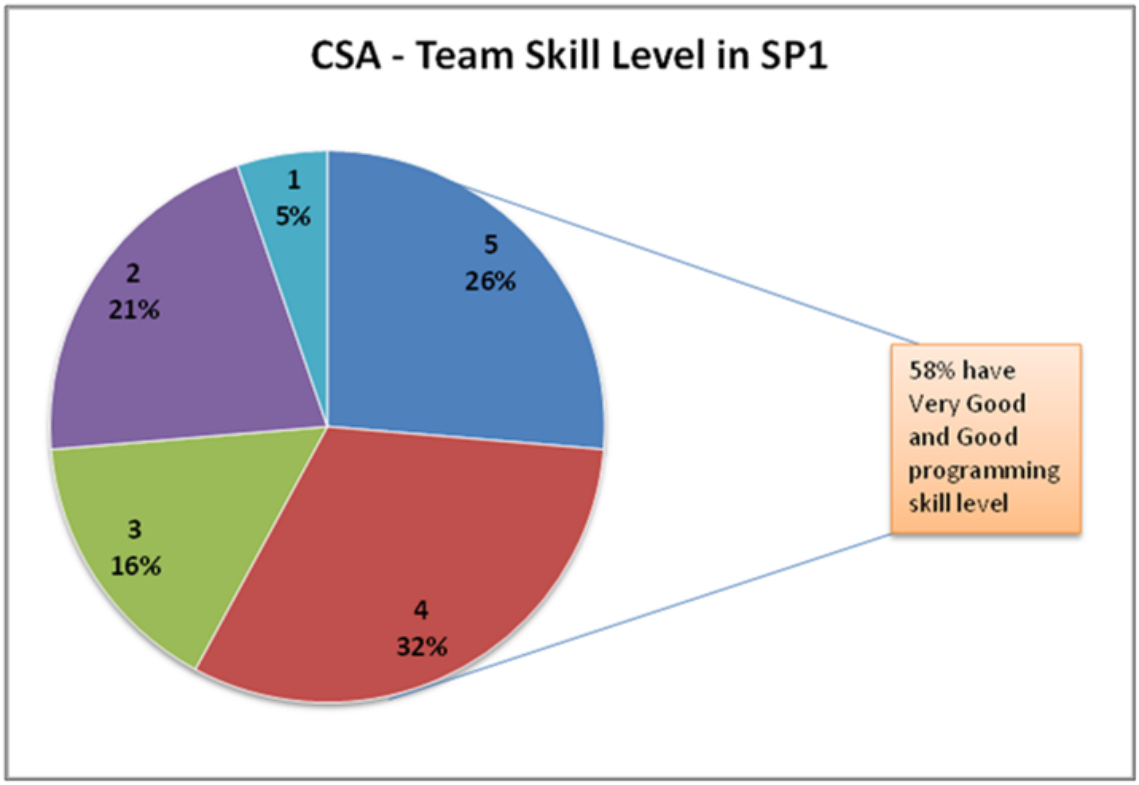


Figure 6-9: Percentage of team member programming skill level for CSA in SP1

6.3.3.3 Product Quality

Defining product quality is rather difficult as Kan (2002) discussed in his book regarding different views in product quality. In relation to software development, it is important to evaluate and measure product quality quantitatively in order to save on software lifecycle costs (Boehm et. al., 1976). In the previous sub section, the TRS and some other additional data gathered were presented in the case study. The three variables identified to be further investigated for their relationship to product quality are: task completion (V1), team performance (V2) and pair programming (V3). Data related to product quality was gathered from two different sources. The first was the educator who evaluated the students' product deliverables based on three categories:

i. Volume – Measured by percentage of completed user stories and acceptance tests;

ii. Quality – Qualitative assessment of code style, system design & patterns proposed by the educator as well as effectiveness of unit testing;

iii. Project management – Record of project files and minutes of meetings.

The category item in (i) and (ii) are used to evaluate the quality product and item (iii) is used to evaluate the quality of the process followed in developing the product.

Table 6-3: Summary of data gathered on team effort / performance, product evaluation and code analysis in CSA

Team	**SP**	**Team Effort/ Performance** (data obtained from ART process and tool support)		**Product Evaluation** (data obtained from the educator)	**Code Analysis** (data obtained from ReSharper tool)		
		TRS	*% Comp Task*	*Product Deliverables*	*C# Compiler Errors*	*Potential Code Quality Issues*	*Redundancies in Code*
Alp1	SP1	320	52.8%	Poor	38	6	393
	SP2	806	45.5%	Fair			
Alp2	SP1	233	**100.0%**	Good	**0**	25	470
	SP2	173	**100.0%**	**Very Good**			
Alp3	SP1	196	87.5%	Good	306	152	558
	SP2	237	**100.0%**	Good			
Alp4	SP1	194	91.2%	Poor	39	171	507
	SP2	156	97.2%	Fair			
Alp5	SP1	215	**100.0%**	Good	9	**5**	170
	SP2	**48**	96.8%	Good			
Alp6	SP1	151	**100.0%**	Good	699	68	**127**
	SP2	90	**100.0%**	Good			

The students were given total marks of 100% for each three categories. The average marks from these categories were taken and mapped into levels from 'very good', 'good', 'fair' or 'poor' for the deliverable. The table of this evaluation is included in the Appendix G.

The second source was taken from the evaluation of the code analysis tool, ReSharper[12]. The students' project code was retrieved from the TortoiseSVN repositories and their

[12] www.jetbrains.com/resharper/

project was imported using Microsoft Visual Studio with the Resharper plugin. The ReSharper tool analysed the students' source code and reported the total defects in terms of total number of C# compiler errors (the most severe and highest impact on the functionality of the product), potential code quality issues and redundancies in code. The outcome for measuring product quality from the tool analysis along with other data gathered earlier is presented in Table 6-3. The table shows the evaluation for each team based on (i) team effort or performance from the data obtained from the ART tool, (ii) product evaluation using data obtained from the educator and (iii) code analysis results as obtained from the ReSharper tool. The best result in each category was highlighted in bold. In the category of team effort or performance measurement, specifically in the total risk identified, it is found that team Alp5 produced less risk or had greatly reduced risk from SP1 to SP2. In relation to percentage of task completion it is also found that team Alp2 and Alp6 had managed to complete all tasks in both sprints. Further, in the category of product evaluation specified that team Alp2 has produced good and very good deliverables whereas in the category of the code analysis tool shows team Alp2 with zero compiler errors.

As mentioned earlier, there was a need to further qualitatively investigate the relationship of the three identified variables derived from the case studies in relation to each team's end product quality. From this investigation, useful indicative results are provided.

V1 (Task completion) – In relation to the high percentage of task completion associated to product quality, it is found that the team with a high percentage of completed tasks may or may not produce good quality product. This is evidenced where team Alp2 managed to complete all tasks with zero compiler errors whereas Alp6 managed to complete all tasks however, had a massive (699) compiler issue count.

V2 (Team performance) – In terms of the team that performed best in their project, the team which has fewest risk was team Alp5. Intuitively it would seem that the team reduced their risk by complying with established agile practices such as assigning a proper team member to a task, participating and reporting progress in team meeting and maintaining one role at a time in a project. The indication is that the team, most compliant with agile practices, may produce better quality software, as determined by the set up criteria.

V3 (Pair programming) – In order to investigate this variable, the investigation was extended, focusing on identifying each team member and whether or not they are practicing pair programming in completing their task. As a result, it is found that none of the team members of Alp3 did pair programming in their project. Referring to the Alp3 code analysis result in Table 6-3 above, it shows a large amount of compiler errors at 306. However, there was another team, Alp6 with a larger number of compiler errors at 699. Hence the results are rather inconclusive in confirming the view that the team who practiced 'single expert' programming do not produce good quality software. As a result, further analysis will need to be done in the future.

Overall, these results are preliminary and a more extensive quantitative analysis of behaviour is required to establish firm correlation between team behaviour and practices and code quality. Once again it is stressed that there is no claim that the study results can be generalized to all agile projects. At best the results could be applied to when new agile team is involved. Nonetheless, the findings show that the team (Alp5) that complied closest to agile practices had the lowest risk identified, the highest number of completed tasks (96-100%), as well as highest quality deliverables (fewest defects).

Chapter 7 - Study Results and Analysis 2: Case Study Beta (CSB)

7.1 Introduction

This chapter will describe the findings from the second case study, Case Study Beta (CSB). Based on the lessons learned from the previous case study, a new method of risk assessment is added in this work: Risk Factor Points (RFP). This is followed by presenting the findings and analysis of the risk results produced using the tool as well as an analysis of additional data collected from CSB artefacts. Again, the aim is not only to demonstrate the use of available data to assess risk using the proposed approach but also to analyse how violation of agile practices could trigger risks in the project. The risk results obtained from both case studies (CSA and CSB) provide useful insights on how agile practices are being applied especially related to the issues addressed in Chapter 3.

7.2 Risk Factor Points (RFP)

As part of the development process for the risk counting mechanism the output from execution of the ART prototype tool from the previous case study (CSA) was presented to the Product Owner for feedback. The Total Risk Score (TRS) for each group in CSA ranged from 151 to 320 in one sprint. This observation has yielded two arguments as follows.

i. The risk identified in CSA may not be realistic enough due to the fact that this project requires all teams to breakdown the user stories into a set of tasks. Although the given product backlog items were the same, there was a difference in the total number of the tasks and also estimated size of the tasks for each team. When the same risk that was associated with tasks in different teams was identified, because the task duration was different in each case, the TRS was inconsistent, leading to difficulty in prioritising the risk. For example, the same risk A found in task X and task Y may have a different critical value depending on the size of the associated task. The risk associated with a bigger task, say X, which was estimated in five hours will be seen as much more critical than the one which estimated as one hour (task Y), even though they are the same risk. Based on this observation, a Risk Factor Points (RFP) score was introduced for CSB.

ii. As one proceeded to run the environment data in the ART prototype tool for the second sprint of CSA, there was also an issue raised by the Product Owner regarding the rule conditions set up for pair programming. For example, the rule condition for a risk in pair programming was developed as *"IF the task is not using pair programming THEN there is a high risk of 'single expert' or the developer not sharing the code".* This is the same case as argued before in (i) where if the task involved is a smaller size of task, pair programming may not be appropriate and therefore this rule is less relevant. Hence, the rule was modified for CSB as described later.

Kontio and Basili (1997) defined a risk factor as a characteristic that affects the probability of a negative event occurring, including the characteristics of the environment as well as the risk event. In Chapter 4, risk rules associated with four goals (see Table 4-2) were developed. In relation to that, eight generic risks derived from these goals were implemented in the case studies. In the event where these risks were considered as contributing to the possibility of a negative event happening, these risks were identified as risk factors. The risks factors are then weighted based on the risk rank (risk matrix) as presented in Chapter 5 and the size of the task associated to the risk; this is calculated as Risk Factor Points (RFP). The table presented in Chapter 5 (Table 5-8) is now extended in Table 7-1 below to include the rank for each risk factor.

Table 7-1: The risk factor rank based on probability and impact used in both case studies

Risk Factor	Probability (P)	Impact (I)	Risk Exposure	Rank
R0001 Pair programming	3	5	Extreme	4
R0002 Task ownership	1	1	Low	1
R0003 High priority task assigned to low skilled team member cannot be completed on time	5	4	Extreme	4
R0005 Overload tasks can cause difficulty in time management	5	1	High	3
R0007 Developer absent in meeting possible of employee turnover	1	3	Medium	2
R0008 No progress report	1	3	Medium	2
R0009 Unable to understand agile process and meet target	3	3	High	3
R0019 Unable to comply with agile process	1	1	Low	1

Where rank; Extreme = 4, High = 3, Medium = 2 and Low = 1

Cohn (2005) emphasized that due to the fact that different people have different set of skills and experience, therefore measuring and estimating one task varies. One might think that three hours is appropriate and the other think five hours is right. Therefore, the size of the task is weighted according to the estimate hours of work and the total estimate hours work in the sprint.

In measuring the Risk Factor Points (RFP) where risk factor *r* denoted in Table 7-2 for agile projects is proposed as below. At the start of a sprint an estimate of hours work (EH) is made for each task *s* on day *d.* Risk Factor Points are then defined as follows:

$$RFP_s = Rank_r \; x \; \frac{EH_s}{\sum_{sp} EH}$$

The total RFP computed for day *d* is then

$$RFP_d = \sum_{s=1}^{s=u} RFP_s$$

where u=total number of task per day.

A simple scenario illustrated as below where on Day1 there was two ongoing tasks TS007 and TS006 that has triggered the same risk *r* R0005. The EH for Task *s* TS007 was five hours, EH for Task *s* TS006 was one hour and $\sum_{sp} EH$ for sprint SP1 was 178 hours. Therefore, the RFP for each Task *s* was computed as below:

Task TS007: $RFP_s = 3\ x\ \frac{5}{178} = 0.084$ whereby

Task TS006: $RFP_s = 3\ x\ \frac{1}{178} = 0.016$

While the total RFP calculated for Day1 is

$$RFP_d = 0.084 + 0.016 = 0.1$$

As denoted in the above scenario, the computed RFP produced a different value for the same risk R0005 triggered in Task TS007 and TS006. Hence, comparing the two tasks having the same risk occurring on the same day, although the risk exposure indicated a high rank risk is involved however, priority should to be given to TS007 with a higher RFP.

Although there were inconsistencies in counting TRS as reported earlier, the formula is still applicable to use in certain cases. For example, TRS is used in order to get an overview of the total number of risk score each day, while the RFP is use to indicate specific risks or tasks that need attention based on their weighted risk or risk points. TRS can be useful to represent a graph of overall risk score daily or per sprint, showing risk activities and much easier to compute as it does not involve detailed calculations. Therefore, TRS is still being applied in some analysis in this case study in addition to the RFP.

7.3 Case Study Beta (CSB) Results and Data Analysis

In this section the results and analysis are presented from the CSB group. Based on the lessons learnt from CSA a modification was made to rule R0001 – Pair programming. Some extra data collection methods were also introduced to better capture how the students performed in agile practices. One downside of this is that this has indirectly given this group of students a hint that their project is being considered from a risk perspective. In relation to this awareness, there was an assumption that all teams would generate fewer risks and further due to modification of R0001; there should be fewer risks R0001, even if the task was not done using pair programming. In contrast, if the size of the task is big (say more than five hours) only then is the risk of no pair programming is triggered. Similarly to the CSA, during the planning phase, each team was given Product backlog items that consist of list of user stories. All teams were again challenged in terms of their ability to plan and estimate the work effort required for each sprint, following knowledge from the course lectures. In sprint planning, each team were required to document the user stories and the breakdown of the tasks as well as to update their tasks status from either 'not started', 'in progress' or 'completed' using the Hartmann-Orona spreadsheet.

7.3.1 Summary of CSB Project Data

Table 7-2 below shows the summary of tasks for each team in CSB. Based on the data gathered, the total number of tasks for each team differs ranging from 42 tasks to a maximum of 99 tasks. One team, Bet3 who had a lower percentage of total completed tasks (69%) in SP1 however managed to get all tasks completed in SP2. The remaining teams managed to get between 85 to 100% tasks completed in both of their sprints. Three teams (Bet2, Bet3, Bet5) had an improvement in percentage of total completed tasks from SP1 to SP2 and two teams (Bet1, Bet4) has had a slight decrease of percentage completed tasks in SP2. Two teams (Bet6, Bet7) managed to complete all tasks in both sprints. In the CSB, the total number of tasks for each team was slightly lower than in the CSA. In the previous case study result, evidence where decomposing user stories were rather challenging was provided, especially when it was done for the first time. For CSB, the evidence points to the same outcome. The table below shows that they too, had a large gap between the total numbers of tasks among the teams, despite the same set of user stories being given to each team. Based on the evidence

obtained from both case studies, it is essential that developers who are new to agile should be given proper guidelines or training on task estimation Furthermore, erroneous task estimation can become a threat to a project. This was supported from the results obtained in the CSA, where most teams had high risk scores (Figure 6-2) as well as carrying out too many tasks (>1) at the same time (Figure 6-6). Similar to the investigation done in the CSA, there is a need to investigate the variable (V1), task completion and its association with the quality product, in CSB.

Table 7-2: Summary of Tasks for CSB in Sprint 1 (SP1) and Sprint 2 (SP2)

Team ID	**Team size**	**SP1 Tasks**		**SP2 Tasks**	
		Total	% Comp	Total	% Comp
Bet1	7	47	93.6%	57	82.5%
Bet2	5	75	93.6%	**96**	**100.0%**
Bet3	5	42	69.0%	**55**	**100.0%**
Bet4	8	**57**	**100.0%**	55	98.2%
Bet5	8	55	85.5%	99	96.0%
Bet6	7	**47**	**100.0%**	**72**	**100.0%**
Bet7	8	**72**	**100.0%**	**85**	**100.0%**

7.3.2 Total Risk Score (TRS) in Case Study Beta

Figure 7-1 and Figure 7-2 below shows the calculated TRS for both sprints. In SP1, the number of risk score ranged from as low as zero (either no risk being present or no ongoing task) up to the highest risk score of 43 in team Bet7, Day8 of the sprint. Whereas in SP2, the risk score ranged from zero to the highest risk score of 38, which was also found in team Bet7, Day9 of the sprint. Both graphs show the increasing and decreasing pattern of risk detected for each team with some peaks on certain days thus providing us a better visualization of the risks arising.

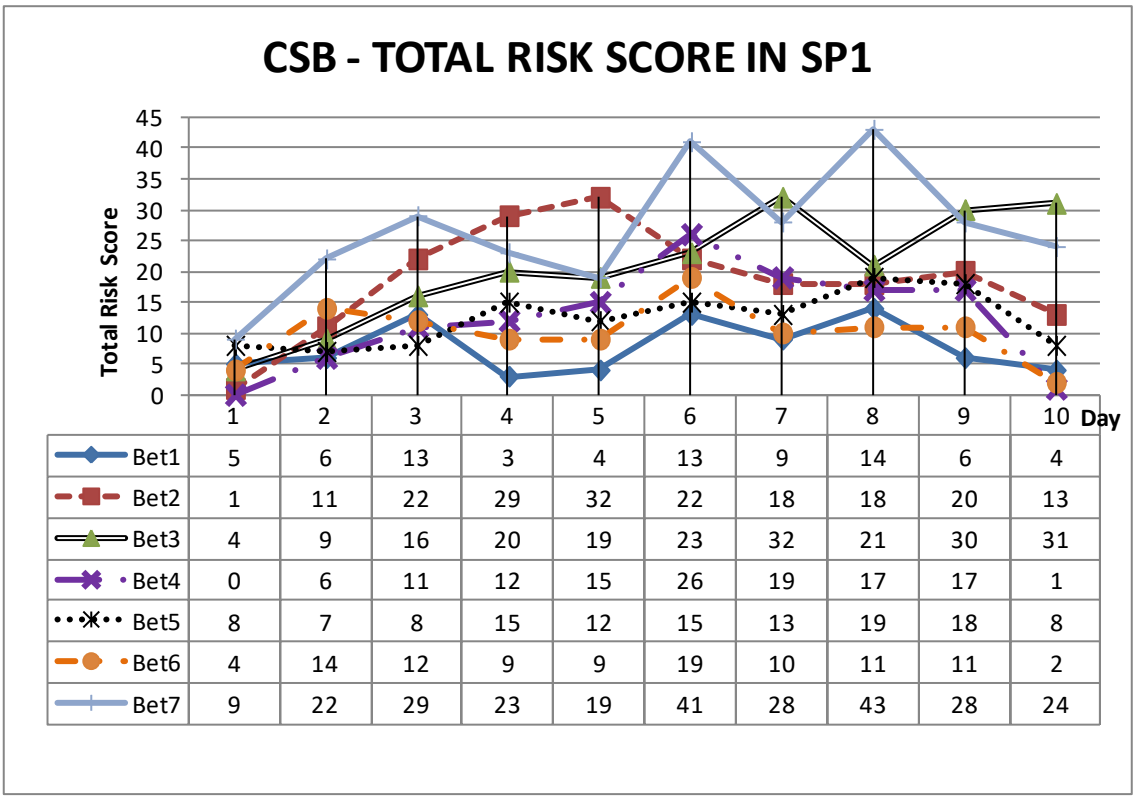


Figure 7-1: Graph of Total Risk Score for CSB in Sprint 1

Comparing with the TRS presented in the CSA earlier (Figure 6-1 and Figure 6-2) for both sprints, the TRS for CSB was much lower than the TRS in CSA. This has supported the assumption stated earlier that the indication of possible inspection with regard to risk may affect the developer or student behaviour during the project. Despite the modification of the pair programming rule for CSB, the TRS produced in this case study was more realistic so that pair programming was critical only in bigger tasks. This may be due to the fact student projects were used in the case study and a less motivation to do pair programming than if told to in a workplace. Thus, a high number of risks have arisen due to the violation of this rule. In practice, the project manager could modify this rule from time to time.

The graph in SP2 (Figure 7-2), shows that at certain points team Bet4 (Day1), Bet5 (Day4) and Bet1 (Day10) had no risks identified, showing a slight improvement from the previous sprint SP1. Although team Bet2 had no risk identified from Day1to Day4, it was not taken into account due to the fact that there were no tasks were carried out on the particular days.

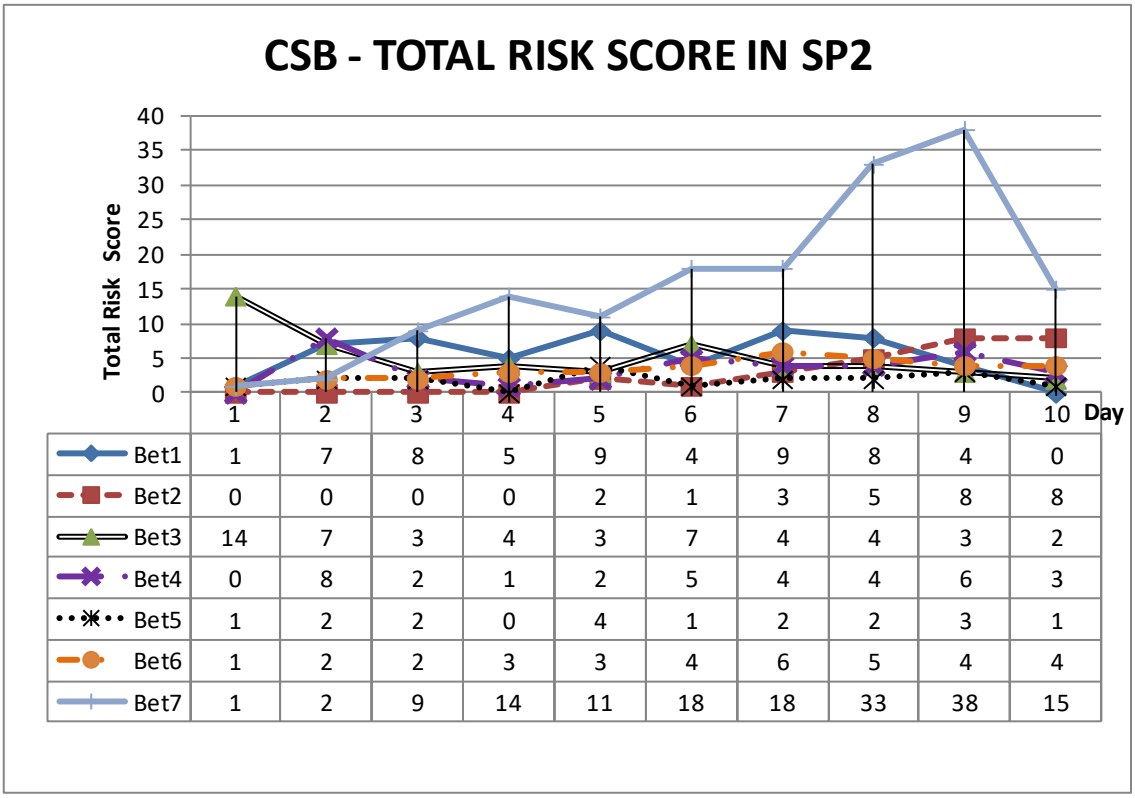


Figure 7-2: Graph of Total Risk Score for CSB in Sprint 2

Previously in the results obtained from the CSA, most of the teams have had high risk score in their project. In order to select one with the best performance the team with the highest difference in mean which was Alp5 was chosen, indicating that the team performed better and has improved in their second sprint in terms of risks. In CSB, all teams reduced risks in SP2 compared to SP1. This result supports the idea as in CSA that the students benefited from better knowledge and experience gained from the first sprint and so were better able to minimize risks in the second sprint. In terms of identifying and selecting the team that performed best in the CSB, each team's mean TRS in SP1 and SP2 was reviewed. Team Bet1 has the lowest mean of 8 risks in SP1 but maintained a high mean of 6 in SP2. Both team Bet2 and Bet3 had high difference in mean compared to SP1 and SP2, 16 and 15 respectively. However, among all teams, Bet5 had the least risks in SP2. Thus, for CSB, team Bet5 performed best in their project in term of having the least risk score in SP2. Similar to for CSA, there is a need to investigate if the team that performs best provides a good quality end product too? These results will be presented later in this chapter.

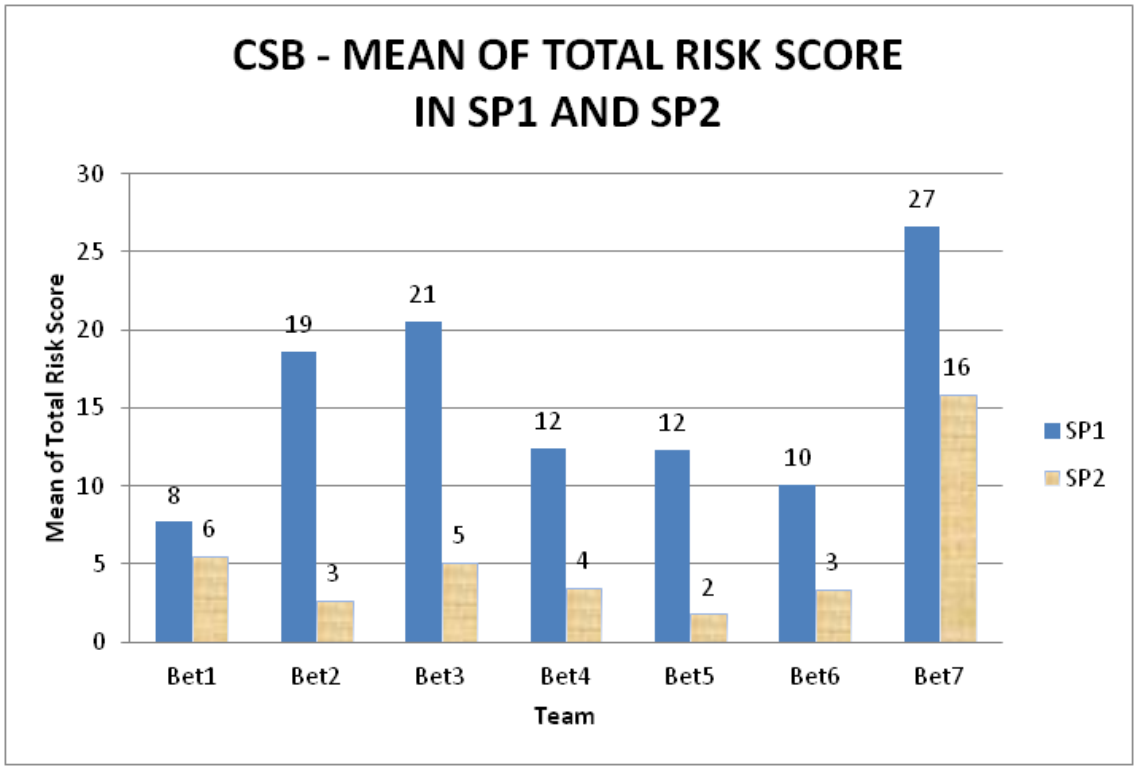
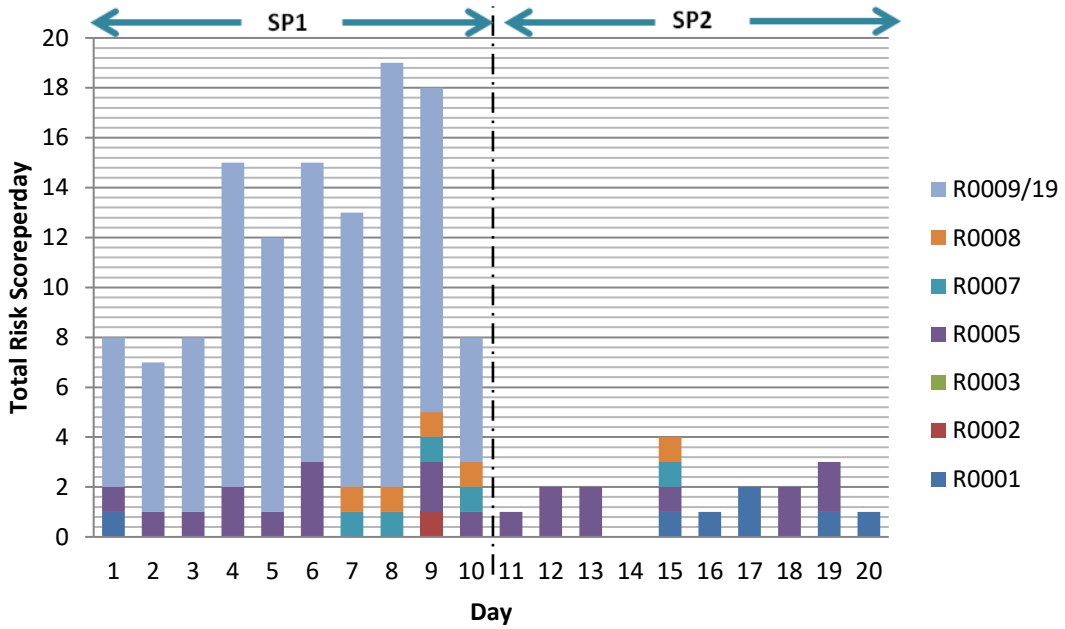


Figure 7-3: Mean of Total Risk Score for each team of Case Study Beta (CSB) in SP1 and SP2

In the light of viewing specific risks that triggered on a particular day, the data shown in Figure 7-4 provides better visualization of the type of risk triggered each day of team Bet5, for both sprints. It is stated that the advantage of having this method earlier in the CSA where it would be possible to view the type of risk triggered for the particular day as well as to view which risk is being triggered the most and therefore should be prioritized by the project manager. Additionally, it helps in deciding whether prompt action should be taken towards a risk that has occurred persistently. Referring to the summary of tasks of the CSB in Table 7-2 presented earlier, Bet5 had 55 total tasks in SP1 which was a worthy average number of tasks in a sprint; however they had the highest total number at 99 tasks in SP2. Looking at this data, and making comparison to the graph below (Figure 7-4), it can be seen that that, although the number of tasks allocated for SP2 increases compared to SP1, the team managed to maintain lowest risk score in SP2.

In addition, while the advantage of visualizing the type of risks occurred can be seen more clearly through this graph, there are still opportunities for better visualization. Since risks identified are associated with a task and a task is associated with a team member, one may extend this visual ability in future by indicating which task is affected

by a risk as well as the team member who are responsible for the risk. This i) provides additional benefits to a project manager in identifying the person responsible for the risk, thus providing an early warning ii) eliminates the possibility of there being no one to take responsibility for the risk, as raised in Chapter 3.



Risk ID and Risk Name

R0001 Pair programming	R0002 Task ownership	R0003 High priority task assigned to low skilled team member cannot be completed on time	R0005 Overload tasks can cause difficulty in time management
R0007 Developer absent in meeting possible of employee turnover	R0008 No progress report	R0009 Unable to understand agile process and meet target	R0019 Unable to comply with agile process

Figure 7-4: Graph showing the breakdown of Total Risk Score for team Bet5 in SP1 and SP2

7.3.3 Risk Factor Points (RFP) in Case Study Beta

Earlier in this chapter, RFP was introduced in order to resolve the anomaly of different task sizing in each team based on lessons learnt from CSA. In a given situation where multiple risks are triggered in a project, the project manager, as well as taking action on the risks, will need to prioritise which risks to attend to first.

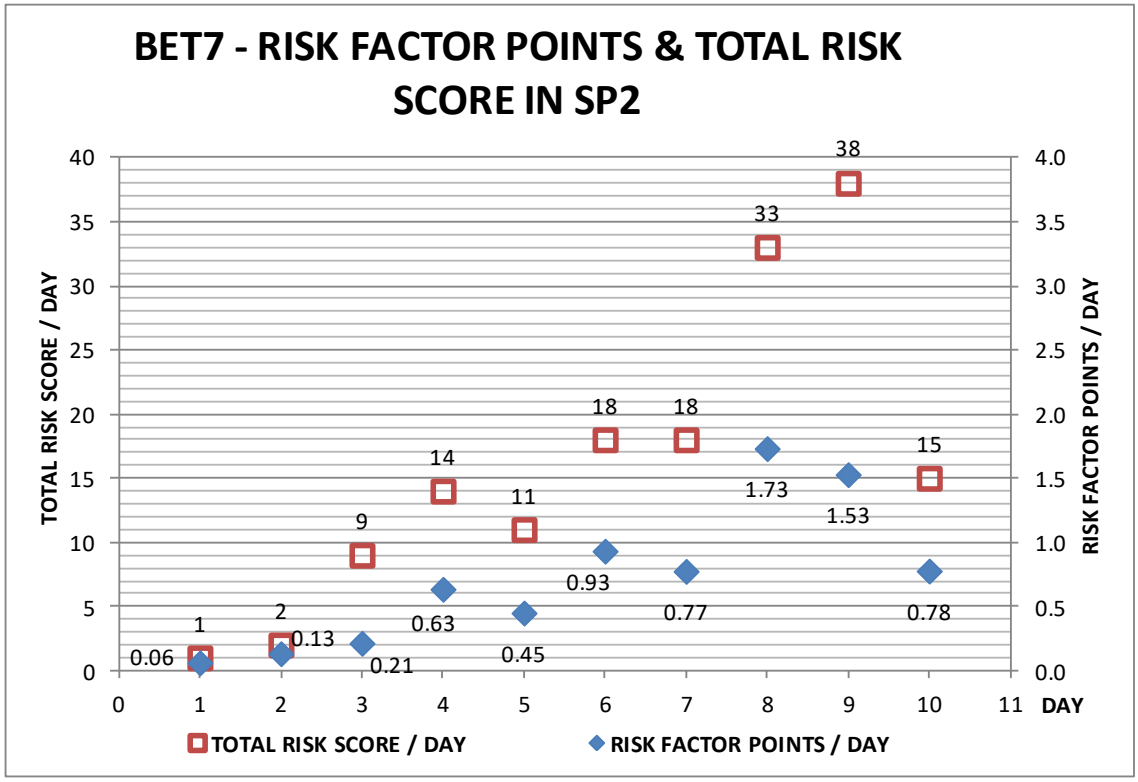


Figure 7-5: Graph showing Risk Factor Points and Total Risk Score calculated daily in Bet7 SP2

The graph above (Figure 7-5) shows two data sets from team Bet7 in SP2; the first data set was the Risk Factor Points (RFP) and the second data set was the Total Risk Score (TRS). As indicated earlier, RFP is measured by multiplying the rank of the risk by the estimated size of the task. Note that the highest risk score of 38 risks was on Day9. However, looking at the total RFP calculated, it is indicated that the perceived risks in Day8 were more critical (1.73) as compared to the one on Day9 (1.53). There were also 18 risks identified in both Day6 and Day7. However, the total RFP calculated shows that the risks found at Day6 needs more attention than the risks found in Day7. This was due to the fact that although the total risk detected in particular days were the same, the risk exposure and estimated size of the task for each risk maybe different. Given a situation where there are multiple risks triggered on the same day, the project manager should be able to decide, based on the RFP, which risk to handle first. In the case where the same risk occurred, the risk with the highest RFP can be given the priority. In addition, the project manager will be able to make a decision whether to attend to the older risk (risk A) that has not been resolved or to handle the new risk (risk B) that has just arisen, based on the RFP calculated for each of the risks. If the RFP calculated for

older risk (risk A) is lower than the RFP of newer risk B, then risk A could wait even though the risk is triggered earlier.

In relation to the case illustrated earlier, a clearer example of the case of multiple risks is demonstrated further. An example of the day with the highest risks occurred was taken which was Day9. The risks identified for that day is shown in detail as below (Figure 7-6). There were 27 affected tasks in Day9 where 11 tasks were affected with two different risks. Out of these 11 tasks, there were four tasks (TS046, TS047, TS048 and TS049) with the highest RFP of 0.09. This was due to the fact that the affected tasks have the highest estimated hours being 13 hours assigned to the tasks. In addition, the risk triggered was also highly ranked. Therefore, based on the RFP calculated, the four tasks indicated earlier should be given the highest priority. Although the RE indicated that the risk identified for the other three tasks (TS053, TS055 and TS058) had an extreme risk ranked however, due to the fact that the tasks had the biggest estimated hours therefore it should be set as priority among others. Tasks with bigger estimated hours and associated risk can cause chaos to the project due to the fact that if they go awry there is more impact.. Therefore, it is essential for the project manager to tackle this risk on this specific task before others.

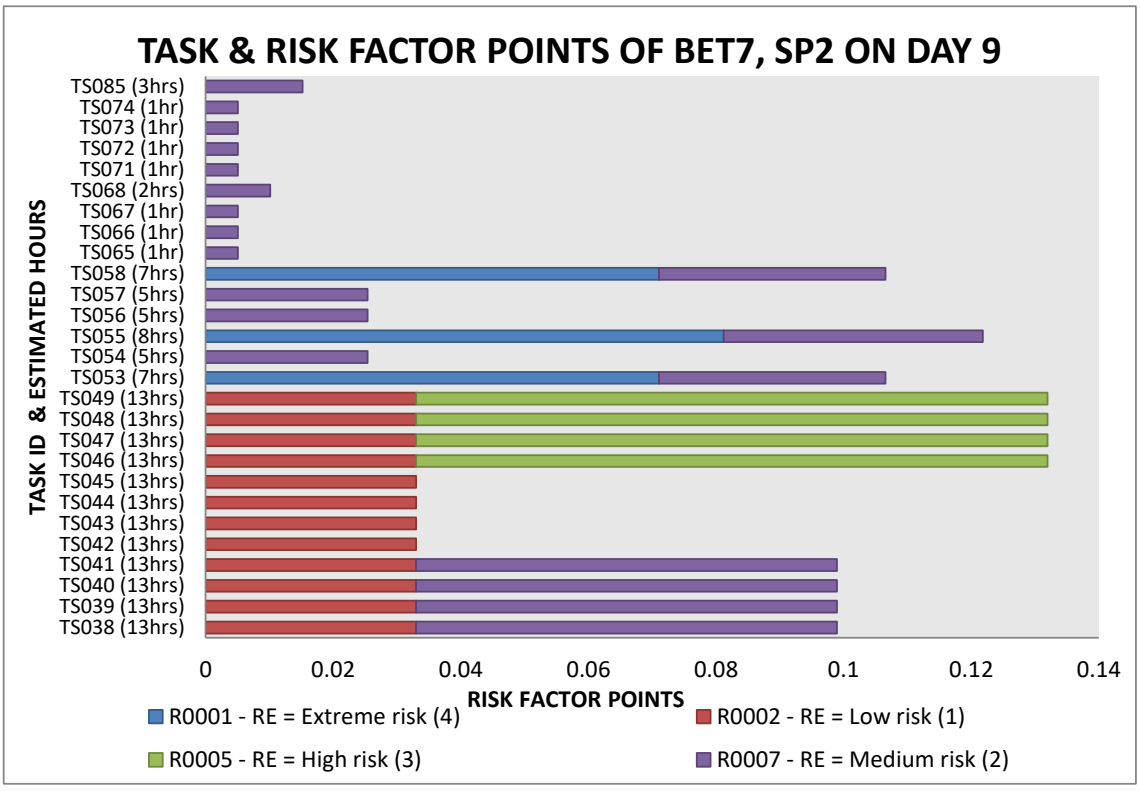


Figure 7-6: Graph of task affected with multiple risks of team Bet7, SP2 in Day9

7.3.4 Additional Data Gathered in Case Study Beta

Similar to CSA, the team member behaviour was investigated, focusing on a few practices such as how team members commit to the given tasks and to pair programming and in both cases the effect on the quality of their product. This leads to answering Research Question RQ1c: Do the findings of the application of risk management in agile projects give useful insights towards an improved agile process and risk management?

7.3.4.1 Total Number of Task Commits per day

Similar to CSA, the students were advised to carry out only one task at a time due to the time constraint, and they maintained the "task consecutive rule". This is the condition where the student has to complete one task before starting a new task. Figure 7-7 and Figure 7-8 below shows interesting information on each team's task commits in SP1 and SP2. In SP1, tasks were ongoing with as many as 31 tasks per day (Bet7, Day6) while in SP2, the total tasks were slightly higher, 35 tasks also in team Bet7, on Day9. When compared to the graphs presented earlier in CSA (Figure 6-5 and Figure 6-6), the results in CSB showed that the teams had varied greatly in the total number of tasks commits each day. Besides, a higher number of ongoing tasks found in this case study, the graphs (Figure 7-7 and Figure 7-8) showed a trendline with high peaks in each team in both sprints, possibly indicating inconsistent effort being applied.

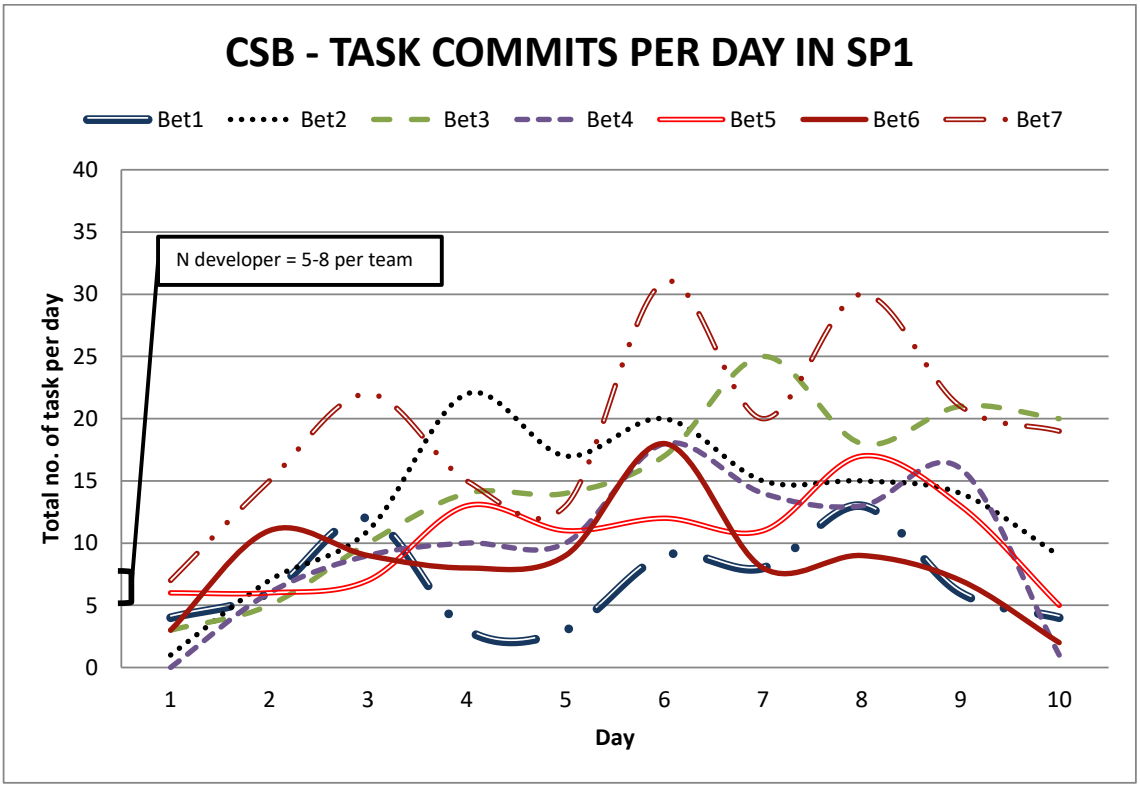


Figure 7-7: Graph showing task commits for all teams of CSB in Sprint 1

Note that the marker drawn on both graphs denotes that the limit of the total task that should be ideally carried out per day in the CSB was between five to eight to comply with the "task consecutive rule" concept. Comparing graphs for SP1 and SP2, there was no difference or improvement in term of adhering to this rule. All teams started on Day1, with the number of tasks being carried out from each team being between zero to seven tasks in SP1 and zero to twenty four tasks in SP2. Throughout the sprint, the graph increases and decreases at certain points whereby towards the end of the sprint, SP1 had a maximum of 20 ongoing tasks by team Bet3 on Day10 whereas in SP2, team Bet2 had 21 ongoing tasks on Day10. In the previous case study (CSA) result it showed an improvement of task commits from SP1 to SP2; however, in CSB that was not the case. In the SP2 graph for CSB it can be seen that there are as many as 35 tasks committed, nearly as many as were carried out in SP1. Therefore, the distribution of tasks for each team member in CSB was investigated using the data from the Hartmann-Orona spreadsheet. Surprisingly, the high number of tasks being carried out in one day may have been caused by the tasks being not distributed evenly between the team members. An example taken from team Bet7 in SP2 where, out of eight team members, two team members were allocated with as many as 21 and 16 tasks respectively while the remaining team members had only four tasks assigned throughout the sprint. This

has evidently showed that the team members with too many tasks assigned to them will possibly commit many tasks at one time, being under pressure to complete the project. Again, this form of practice in a real project can serve as a threat to a project whereby too many tasks commits at the same time can indicate negatively on the quality of the code produced (Intuitively, it means that the coding may have been rushed). Further to this, the team member with a large number of tasks is most likely not maintaining a sustainable pace in the project, again indicating agile non-compliance. Ideally the task commits graph should be decreasing towards the end of the sprint. However, due to the problem of uneven task distribution, this will not be the case. Further to this, teams failed to adhere in establishing consistent effort in completing tasks each day. For example in SP2, team Bet2 did not commit to any task starting from Day1 until Day4 which has meant that the team has had to commit more tasks on the last day of the sprint. Again, the project manager will be able to use this data in order to predict risk in relation to either inconsistency in the number of tasks being carried out each day or to identify where no task has been started or carried out. As a consequence of these situations, there will be a higher chance of incomplete tasks in the sprint.

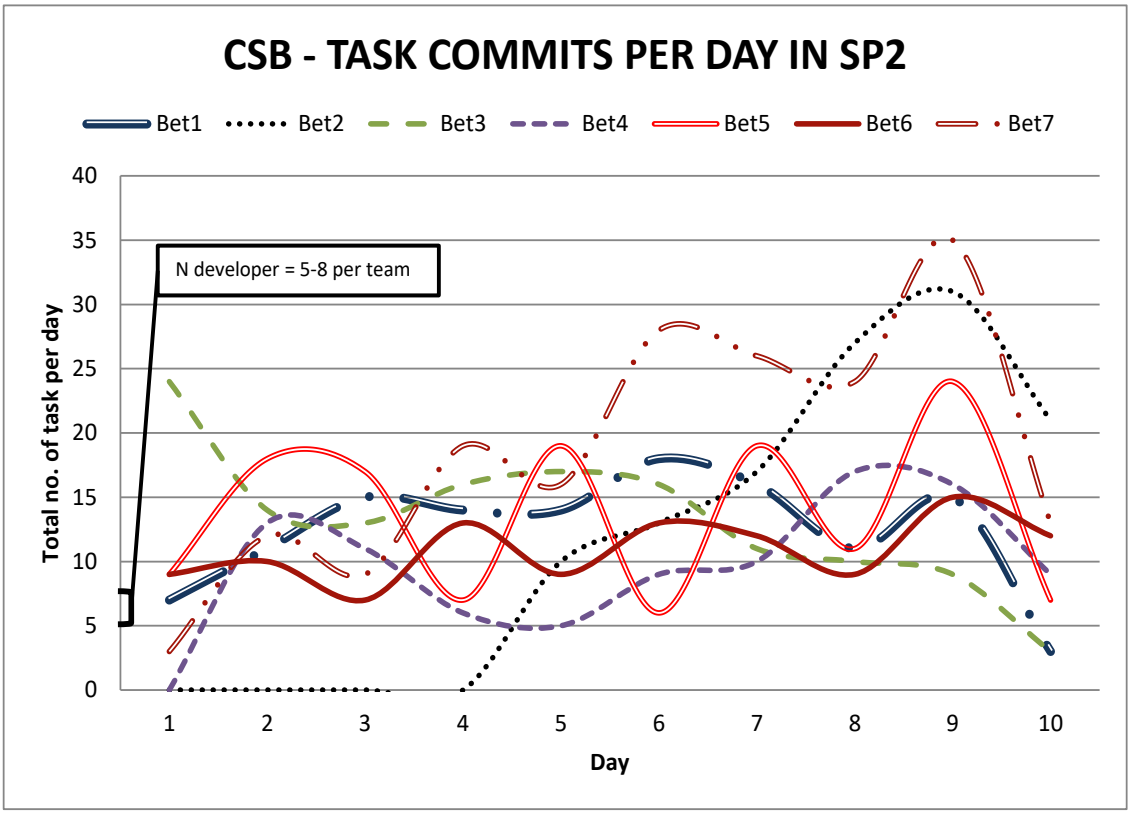
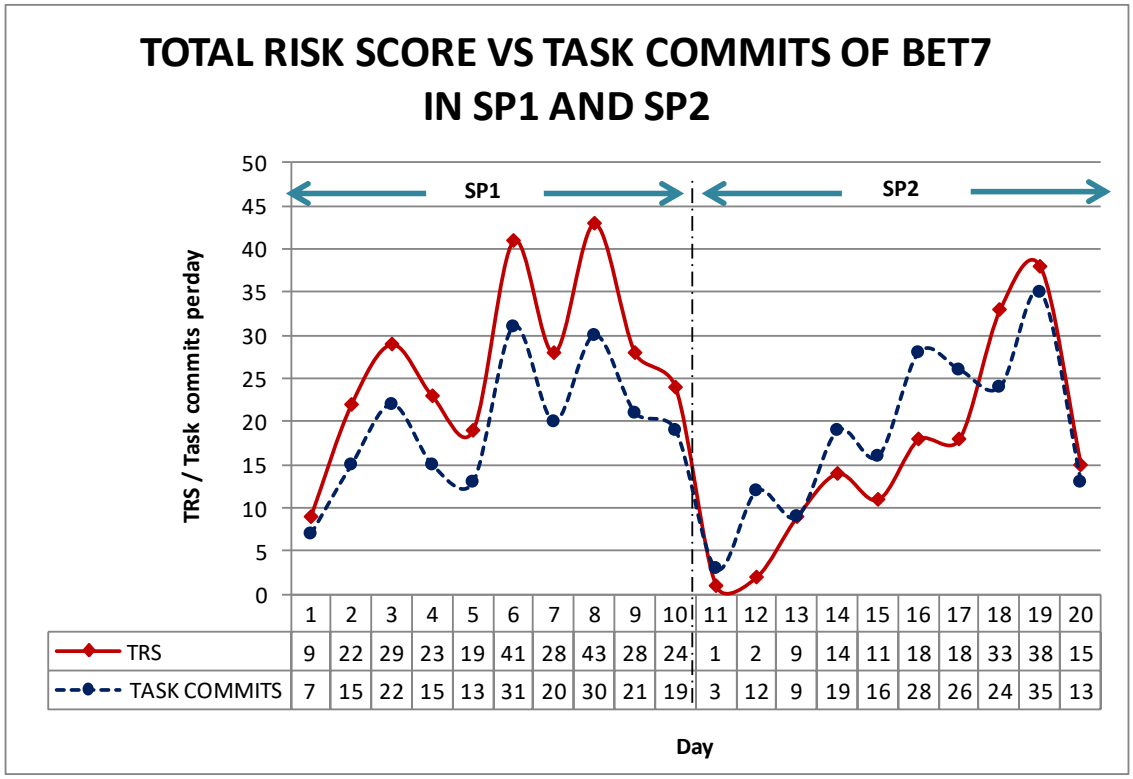


Figure 7-8: Graph showing task commits for all teams of CSB in Sprint 2

Further, from the additional data obtained from the archived documents of the student projects, the TRS and task commits in a sprint were compared and shown in Figure 7-9 below. In CSB, an example of the team Bet7 was taken which had the highest risk score in both sprints. In SP1, since all teams were new to agile projects, there may be a positive relationship between the task commits and the TRS for the sprint, where the TRS increases as the number of ongoing tasks increases. Ideally, in SP2 the risks should be starting to decrease gradually. However in this case, towards the end of the sprint particularly in Day8, Day9 and Day10 the total risk has started to increase again. This can provide an alert to the project manager that there may be problem which has occurred in these three days.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TRS	9	22	29	23	19	41	28	43	28	24	1	2	9	14	11	18	18	33	38	15
TASK COMMITS	7	15	22	15	13	31	20	30	21	19	3	12	9	19	16	28	26	24	35	13

Figure 7-9: Graph comparing between the risk score and task commits of team Bet7 in SP1 and SP2

7.3.4.2 Pair Programming

In the previous case study CSA, it was shown that most students failed to employ pair programming, with the majority of 68% (SP1) and 66% (SP2) students completing their tasks without practicing pair programming. In CSB, more than half (54%) of the students performed pair programming were identified in SP1, while in SP2 71% of the students did not practice pair programming (Figure 7-10). In addition, with the perspective of the percentage of task completed, more than half of the team members

completed their tasks without practicing pair programming in either sprint. In order to investigate further, each team's percentage of pair programming practice applied in SP1 and SP2 were examined in detail. In SP1, all teams performed at least some tasks in pairs. However; in SP2 it was identified that team Bet6 did not conduct any pair programming which contributed to higher number of 71% without pair programming in SP2. Team Bet6 had a total of 72 tasks with 100% completion in SP2 so that it is assumed that the team might be under pressure and wanted to complete all tasks by using all resources in terms of the team members and time and thus decided to sacrifice pair programming practices. Following from that, it would be interesting to know if the risks related to pair programming were picked up by the tool. The nuanced pair programming rule introduced in CSB means that only tasks with estimated hours of more than 5 hours will be triggered for non-compliance. In SP2, Bet6 had a total of 34 risks and out of this number, 14 risks were related to R0001 – Pair programming, which was triggered from Day2 until Day9. Evidently team Bet6 ignored the risk in an attempt to get all of the tasks done. While acknowledging that risk represents a threat to a project in a real world situation, there is a need to investigate further what the impact was on the quality of the end product produced by team Bet6, later in this chapter.

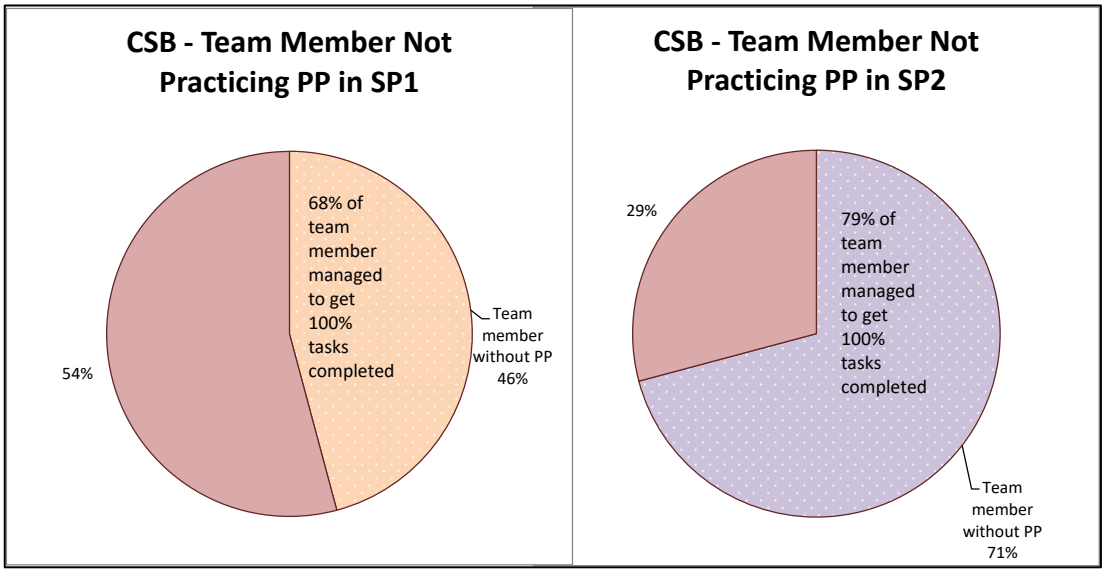


Figure 7-10: CSB – Percentage of team member not applying pair programming in SP1 and SP2

Each student's skills were examined in CSB and Figure 7-11 reports the assessed student programming skill level in SP1. From the total number of 48 students, 9 students possess a low skill of 2, 11 students possess average skill and 28 students possess good or very good skills. Similar to the result found in the CSA, more than 50%

of the students in CSB were either very good or good in programming. This has reinforced the 'single expert' practice adopted by students as evidenced by results obtained from the CSA and CSB. As stated earlier for CSA, further investigation needs to be done by interviewing the students in order to find the reason behind their preferences and work practices.

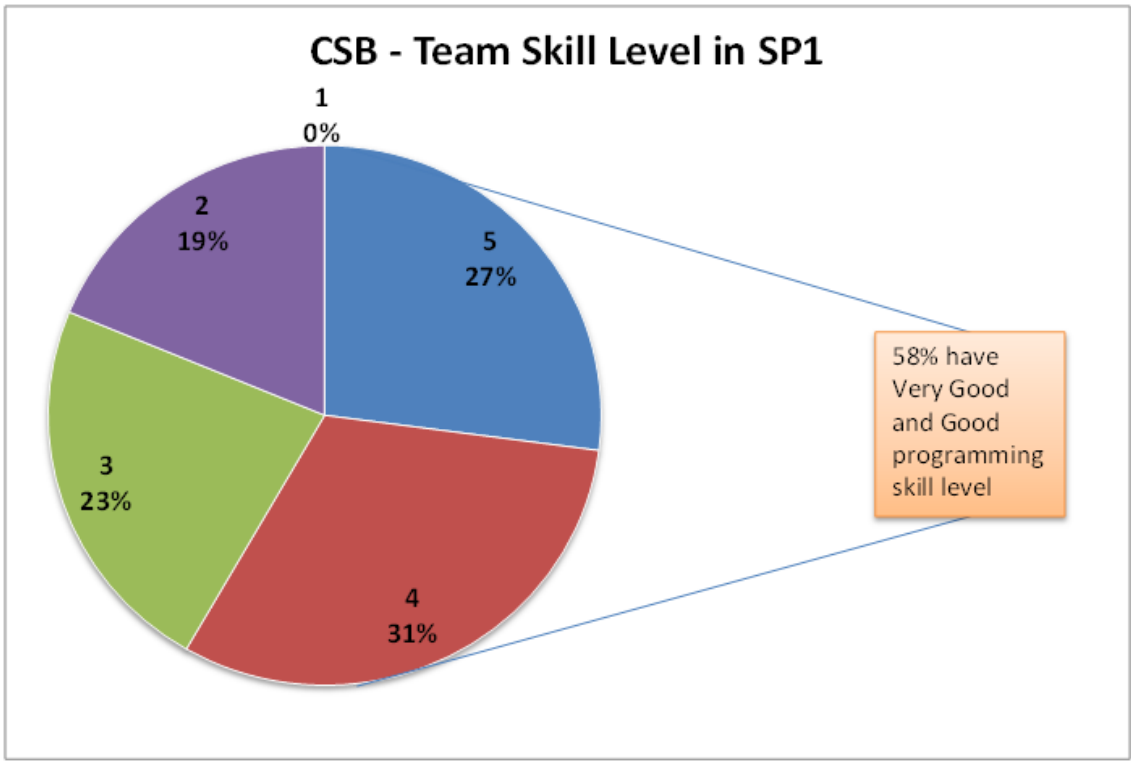


Figure 7-11: CSB – Team member programming skill level in SP1

7.3.4.3 Product Quality

An international standard for product quality, ISO/IEC 9126[13] presents a quality model that comprises of six characteristics and 27 sub characteristics of quality measurement. The characteristics highlighted are functionality, reliability, usability, efficiency, maintainability and portability. Due to the fact that the quality model is generic, it can be applied to any software product by tailoring to its specific purpose (Jung, 2004). Nonetheless, Kitchenham and Pfleeger (1996) argued that there is no universal definition of quality. Therefore, good product quality in the context here refers to developing the closest requirements from the Product Owner, no or minimal compiler errors, absence of code quality issues (as determined by code quality metrics) and absence of code redundancy.

[13] ISO/IEC 9126-1 (2001) Software Engineering product quality – Part 1: Quality Model, International Organisation for Standardisation.

In this case study there is a need to measure the quality of the product produced by all teams in the CSB. The same approach was used as in the CSA where the data that can be used to measure the product quality for CSB was collected. Data was gathered from two different sources: i) evaluation by the educator and ii) use of a code analysis tool, ReSharper[14]. The mark obtained from i) was mapped to levels: very good, good, fair or poor. Source ii) was used to analyse students' source code to determine the total defects in terms of the total number of C# compiler errors, potential code quality issues and redundancies in code. The outcome from the tool analysis along with other data gathered earlier is presented in the table below (Table 7-3).

In the category of team effort or performance measurement, specifically in the total risk identified, team Bet5 produced average risk in SP1 but managed to produce the least risk in SP2. In relation to the percentage of task completion, two teams Bet6 and Bet7 had managed to complete all tasks in both sprints. Further, according to the Product Owner evaluation team Bet7 produced good deliverables, whereas the code analysis tool shows that only a few teams (Bet3, Bet4, Bet5 and Bet7) produced zero compiler errors.

As in CSA, three variables were identified as useful to relate to product quality; (V1) task completion, (V2) team performance and (V3) pair programming in CSB. The results were summarized as follows:

V1 (Task completion) – The team with highest completed tasks percentage may or may not produce good quality product. This is shown where team Bet7 managed to complete all tasks with zero compiler errors but has developed it with a great deal of code redundancy (1706 lines of code), whereas Bet6 managed to complete all tasks but had the highest count of 162 compiler errors. In this case, team Bet6 may have checked all the tasks as being completed whereas the compiler indicates it should not have been checked in.

V2 (Team performance) – The team with fewest risks was team Bet5. It can be assumed that the team reduced their risks by following the correct practices, like assigning highly skilled team member to a high priority task, participating and reporting progress in team meeting and maintaining one role at a time in the project. The conclusion is that the team who performed best may produce better quality software as evidenced in the table showed a zero compiler errors from team Bet5. Although the team produced a massive 170 potential code issues and 497 code redundancies, having

[14] www.jetbrains.com/resharper/

zero compiler error confirms the product functionality as more likely to be working and with less serious flaws in the developed product.

Table 7-3: Summary of data gathered on team effort / performance, product evaluation and code analysis in CSB

Team	**SP**	**Team Effort/ Performance** (data obtained from ART process and tool support)		**Product Evaluation** (data obtained from the educator)	**Code Analysis** (data obtained from ReSharper tool)		
		TRS	*% Comp Task*	*Product Deliverables*	*C# Compiler Errors*	*Potential Code Quality Issues*	*Redundancies in Code*
Bet1	SP1	77	93.6%	Fair	23	75	674
	SP2	55	82.5%	Fair			
Bet2	SP1	186	93.6%	Fair	2	165	1191
	SP2	27	**100.0%**	**Good**			
Bet3	SP1	205	69.0%	Fair	**0**	67	421
	SP2	51	**100.0%**	Fair			
Bet4	SP1	124	**100.0%**	**Good**	**0**	13	**29**
	SP2	35	98.2%	Fair			
Bet5	SP1	123	85.5%	Fair	**0**	170	497
	SP2	**18**	96.0%	Fair			
Bet6	SP1	101	**100.0%**	Fair	162	**9**	38
	SP2	34	**100.0%**	Fair			
Bet7	SP1	266	**100.0%**	**Good**	**0**	81	1706
	SP2	159	**100.0%**	**Good**			

V3 (Pair programming) – In order to investigate this, the extended investigation, focusing on identifying team members who are practicing or not practicing pair programming in completing their task. As a result, it was found that all team members of Bet6 did not adhere to this practice in SP1. Bet6 code analysis result in Table 7-3 above showed highest compiler errors (162), which supported the evidence that the team practicing ‘single expert’ programming may not produce good quality software.

This is also supported by results from other measures where product deliverables evaluation is 'fair' for both sprints with average total of 68 risk score.

Bet4, the best team met almost all characteristics identified as leading to project success including reduced risk from SP1 to SP2, had 98-100% completed tasks, producing high quality deliverables with minimum defects. This has clearly supported one of the values in agile manifesto "*individuals and interactions over processes and tools*" which implies that having the right people doing the right process will lead to project success.

7.4 Conclusions based on results and analysis from CSA and CSB

The results obtained from both case studies have offered evidence that apart from the novelty of the proposed approach and tool support, the approach is usable and provides useful data. The following insights can be used as a guideline for performing agile projects and thus can help to implicitly reduce risk.

(i) Planning of effort and task estimation is challenging especially to those who do not possess any experience in agile projects (Deemer and Benefield, 2006). A large variance was found in the number of tasks identified and students had often an unrealistic breakdown of user stories into tasks and then were challenged in estimation of those tasks, especially in the first sprint. Two teams produced as few as 33 tasks in one sprint (CSA –Alp3 and Alp6) and one team produced as much as 123 tasks (CSA – Alp1).

(ii) Team formation is important as this will clearly affect the effectiveness of team member communication and cooperation. In this study, the students were allowed to choose their own team members. In the CSA, one team, Alp1 produced what seems like an excessive number of tasks (123 total tasks) in a sprint and correspondingly only between 45.5% and 52.8% task completion. They also had highest TRS in both sprints with 320 and 806 risks respectively. Further, their product deliverables were evaluated as being between 'fair' and 'poor' and had their code was observed to have a high number of defects. As confirmation of this result, Alp1's team performance was shown to the educator and he confirmed that this team was formed from students who were unable to self-organize. It can be inferred that those team members may have been less cooperative and

communicative, as a result of being ‘forced’ together. This supports the view that how the team is formed is important to the success of the project.

(iii) A team which is new to agile projects may tend to have difficulties in establishing consistency in maintaining and committing tasks in a sprint (Layman et. al. 2006; Deemer and Benefield, 2006). In both case studies, the graphs showed days which some teams had too many ongoing tasks and at certain days there were no tasks being carried out at all. This can reveal issues in the team’s ability to keep a sustainable pace. Besides, it was also found that there were tasks that were not being distributed evenly between the team members. Conforming to the “task consecutive rule” is important as the developer when accepting or committing to a task needs to do this based on their capability. For example, the level of expertise in programming or the availability should be considered. In addition, the “last minute attitude” should be avoided as most teams carried out more tasks at the end of the sprint.

(iv) The TRS and RFP graphs produced in both case studies provide a useful visualization method which supports identification and monitoring of risks in a dynamic agile project. The TRS graphs show the number of risks picks up by the ART agents and provide a realistic and interactive way of monitoring risks. Since the RFP graphs show how specific risks are associated with a task they assist the project manager in making decisions whenever multiple risks are triggered at the same time. In both case studies, it was found that there is a positive relationship between the task commits and TRS during the first sprint of their project. When the teams entered into the second sprint, presumably after they developed their confidence, the relationship is no longer applicable. However, this needs to be further investigated using a proper correlation testing in future projects.

(v) In both case studies, it was found that most of the students failed to adhere to the practice of pair programming. While exploring their programming background in detail, it is evident that most of the students who possessed ‘very good' and ‘good’ skills tended rather to apply a ‘single expert’ practice. However, this needs to be further investigated in future projects.

(vi) The relationship of quality of the end product to three variables; task completion, team performance and pair programming was also investigated qualitatively.

Based from the results obtained from both case studies, the following observations were developed:

(V1) The team who produced highest or 100% completion of task may not produce good quality software.

(V2) The team with highest performance (adhere to agile practices) in a project may produce good quality software.

(V3) The team who did not practice pair programming may not produce good quality software.

Due to the fact that the results obtained for the conclusion in (vi) from the case studies was based on a single data item the results cannot be generalized. As a result, a further testing was developed in order to test these conclusions. Correlation analysis was used in order to identify the strength of relationship between variables. Positive relationship indicates as one variable increases so does the other and negative relationship indicates that as one variable increases the other decreases. Since that the data is not normally distributed (based on normality test) and n<30, the Spearman Rho correlation analysis was used.

The correlation coefficient has to lie between -1 and +1, where a coefficient of +1 indicates a perfect positive relationship and a coefficient of -.01 indicates a perfect negative relationship. The size of the correlation coefficient value as suggested in Cohen (1988) refers to the following guidelines:

Small	r =.10 to .29
Medium	r=.30 to .49
Large	r=.5 to 1.0

The analysis was conducted using the SPSS[15] version 21, a dedicated tool for analyzing statistical data. The data obtained for V1 (task completion rate) and V2 (TRS) were based on the tables presented earlier; Table 6-3 in CSA and Table 7-3 in CSB under heading task completion percentage and TRS. While the data obtained for V3 and product quality are shown in Appendix G. The variables were entered into SPSS for the correlation to be obtained. The results are presented in Table 7-4 below.

[15] http://www-01.ibm.com/software/analytics/spss/

Table 7-4: The correlation coefficient of variables V1, V2 and V3 to Product Quality

Variables	Product Quality based on Educator's Evaluation	
	CSA	CSB
V1-Task Completion	0.715**	0.606*
V2-Team Performance	0.235	0.095
V3-Pair Programming	0.220	0.120
*. Correlation is significant at the 0.05 level (2-tailed). **. Correlation is significant at the 0.01 level (2-tailed).		

The correlation results show that there are associations between variables and product quality as reported in (vi). As indicated in the correlations table above there was a large, positive relationship between task completion (V1) and product quality where, r = .715 in CSA and .606 in CSB. However, this result is opposed to the one that was reported earlier where the team who produced higher task completion may not produce good quality software. Meanwhile, the correlation result between team performance (V2) and product quality as well as the correlation between pair programming (V3) and product quality indicate that there are no significant relationships between them.

There is a need to also clearly report on the significant level of the relationship. The significance level listed as Sig. (2-tailed) does not indicate how strong the two variables are associated but instead it indicates how much confidence one should have in the results obtained. The significance of r is strongly influenced by the size of the sample. Therefore, if the correlation has a small size of sample for example n<100, one may have a moderate correlations that do not reach statistical significance at the traditional p<.05 level.

The proposed approach and tool support demonstrated in this thesis is applicable only to the people and processes for agile projects. No claim is made that the approach can offer a better quality of the product but rather it provides new methods for risk identification and monitoring. However, this approach when used along with automated software quality tools could lead to more successful agile projects with higher quality products. As discussed in the literature review, having good quality developers is one of the success factors in an agile project. The proposed approach and tool support helps to some extent by considering programmer skill as a risk indicator.

7.5 Study Validity

Since the study presented introduces a new approach in managing risk in agile project, the main issues are focused on the internal threats. The first internal threat is in terms of the accuracy of the measured data, especially because the data used was based on historical artefacts. Further, confirmation of this data was not possible as the project had already been completed at the time of analysis. Justification was made before opting for this method of validation; one of the main reasons argued by Layman et al. (2006) is that when collecting information from industry case studies there is a difficult challenge in obtaining live data. He goes on to explain that successful studies of industrial based projects mostly use historical data. Even apart from this, agile traditionally requires lightweight methods and so minimal intrusion is preferred when collecting data.

Secondly, the approach used emphasized manual collection and translation of data from archived artefacts into the ART tool. This human effort was required before the ART agents could begin reacting towards environment data as they were designed to work in. This effort could be minimized by selecting a proper individual in the team to conduct this process, for example the Scrum Master in a Scrum project. In the second case study, CSB, the amount of time in manual input was greatly reduced. In future, there is a plan to extend the prototype into a workable risk management system that can be plugged in into a range of agile project management tools.

In extraction of data from the artefacts into the ART template, there was some missing information which resulted in some data being excluded from this study. Further, the use of the student's agile experience level in SP2 was based on educator's observation on student's performance in SP1. However, it was not possible to measure the degree of bias of the educator towards individual students. It is assumed that the absence of any favouritism and also perfect accuracy in the educator's assessment of students.

Considering external validity threats, the risk management approach and tool supports were designed to be as general as possible so that this is applicable in general to agile project environments. This includes taking into account two popular agile project management tools studied for this work so that the approach is as applicable as possible to other contexts but also lightweight and unobtrusive to the team daily activities. Nonetheless, no claim can be made of good fit with tools not studied. Additionally, the study used student project data rather than industrial data. Hence, there will be

arguments whether this is applicable to a real world environment. One might argue that in a well-established agile project where career development and job security is an issue for developers, there will be no issues with pair programming. However, the evidence seems to support the view that pair programming is not universally adopted in agile projects (Bustard et. al., 2013). Further, every developer in an industrial team might more likely be considered an expert, experienced in agile practices. On the contrary, one example reported by Vanhanen and Lassenius (2005) gives an example where developers abandoned the pair programming practice due to the fact that they felt it was unsuitable. On the other hand, this new approach might be more suitable for a newly set up or frequently changing agile team where the agile practices and experience are still at an early stage. Thus, the proposed approach and tool support will help in providing alertness to a threat that is likely to happen in the project or any overlooked risk.

Runeson and Höst (2009) review case study execution processes and terminology from other sources such as Wohlin et. al. (2003) and Yin (2009) and adapt them for Software Engineering research. They provide recommended practices and an extensive empirically derived and evaluated 50 point checklist to support case study research. These include items relating to the design of the case study, data collection, analysis and reporting. In as far as is practically possible, these guidelines have been followed for all the case studies reported in this research. The following chapter will provide further discussion in relation to the contributions of this study.

Chapter 8 - Discussions and Conclusions

8.1 Introduction

Every research work tries to makes a unique contribution to the body of knowledge in its field. For software engineering this entails a clear understanding of the knowledge coupled with a contribution of new methods or processes, technologies and tools and potentially solving some problem from a different perspective. Software risk management has been recognized as important from decades ago. However, due to it being highly human-centered and often a heavyweight process, it has often been ignored. Based on a literature review and investigations it is concluded that risk management should be tailored according to the needs of the project. For example, risk management for a new project or new team can generate different risks from a well-established project. Nonetheless current risk management models and processes can be used as a starting position but additional effort is needed at the start of the project in order to specify how the selected model can fit specifically with the project. This is to avoid waste of the effort such as from any unnecessary process.

A lightweight approach to risk management has been proposed coupled with autonomous agents acting on information gathered from the software engineering environment. This lightweight risk management approach coupled with autonomy via agents helps to ensure effective risk management. Risk management needs to be continuously practiced throughout a project starting from risk identification, analysis

and monitoring. There is no point in identifying and analysing risk if there is no effort made to monitor the risk, subsequently. This research work significantly contributes to such a direction and moves forward the idea that risk management can be effectively applied in agile projects in industry.

8.2 Discussion on Initial Investigations

This research work also has led to the following investigations and findings concerning the current state-of-the art of risk management.

(i) **Issues in software risk management including a regional empirical assessment of the current state of the art for risk management application in industry**.

In order to investigate the current issues in risk management particularly in the industry, an initial qualitative study on risk management practices and barriers to risk management was carried out for companies in Northern Ireland.

This study reports results from the survey of experienced project managers on their perception of software risk management and how it is performed. From a sample of 18 experienced project managers, a good awareness of risk management was found, but with low tool usage. Evidence suggests that the main barriers to performing risk management are related to its perceived high cost and comparative low value. Risk identification and monitoring, in particular, are seen as effort intensive and costly. There is some evidence also that indicates that risk management adds complexity to the project while the real pressure is to complete the project within schedule and budget. There is a further perception that agile methods appear to be lightweight and inappropriate for risk management and, following from that, that agile methods implicitly reduce risks anyway. The perception is that risk management is not prioritized highly enough or not considered valuable enough compared to other activities in a project. The conclusion is that more must be done to visibly prove a high value:cost ratio for risk management activities and especially to reduce the cost side of this relationship.

The results of this study are referred back to the literature review, the conclusion being that two major problem areas in risk management are (i) the extent of

human effort due to the multifaceted and complex steps in risk management and (ii) organizational structure and human behavior towards implementing risk management. Due to these reasons, risk management has often been ignored in modern iterative software development processes such as in agile methods. There are also claims that agile methods inherently diminish risks leading to the next investigation area.

(ii) Issues in agile projects that can be considered a threat to a project.

An initial review of agile methods discloses that due to its shorter cycles and hence easier cost prediction, a risk reduction is expected. Even though Agile Methods seem to reduce some risks, this does not mean that all risks are negligible. A further investigation on the relevant issues in agile projects was conducted to find which issues that, if not treated, can become threats to a project. The conclusion was that the main source of risks is in relation to people factors. This sets up somewhat of a challenge to one of the values in Agile Manifesto, "*individuals and interactions over processes and tools*". Among issues highlighted are difficulties in forming a team, the problem of multiple responsibilities, the possibility of a lack of accountability, what happens if there is a lack of collaboration, issues in team motivation and competency and the possibility of high personnel turnover. Meanwhile issues in practices indicate that teams have problems with task estimation, pair programming, collective code ownership and complying with the need for daily meetings. These issues are considered as possible risk factors in agile projects.

Due to the fact that issues found (i) are related to human effort in risk management and (ii) are closely related to the environment of the project i.e. the people and practices, one possible solution may be found in using software agents. Software agents help to perform on behalf of some other program or people and can employ rules in order to achieve goals. They are capable of performing tasks without direct control or at least with minimum supervision. Therefore, there is a need to propose to substitute some of the human involvement and minimize the need for manual input. In order to do this, software agents need to react based on a certain environment. Thus, for this purpose the following investigation was initiated in order to study what project environment data that is available and

could be used by agents in order to partially automate the risk management process.

(iii) Project environment data in typical agile projects that can be used in software risk management and on which to base automation.

The first step in this was to investigate what data is available or commonly used in software project environments in general and so can be captured with little or no cost i.e. it is already available from the environment toolset or its data stores. Two different project management tools were studied: Rally Software and eXtreme Manager. Information on the environment data model from both tools were gathered including the process model and objects, attributes and values used therein. From that study, a more generalized data definition was obtained that is used for this research work.

From this investigation, it is concluded that there are many available project environment data that can be used in order to automate risk. However, in order to identify which data is useful, a study of possible risks is required in order to decide on which available data is useful. Thus, the need for a Rule template was established as a means to state which risk that can negatively impact on project goals and which data can be used in identifying such a risk.

8.3 Discussion on Research Questions

Three main Research Questions (RQs) were established earlier as an expression of the aims and objectives of this project. This section will summarise how these have been responded to in the course of the research work

8.3.1 Research Question RQ1: Risk Management Barriers and Issues in Agile Projects

RQ1a: Is risk management being performed in industry; if not what are the barriers?

RQ1b: What are the possible risk factors that can be identified and managed throughout agile projects?

RQ1c: Do the findings in the application of risk management in agile projects give useful insights towards an improved agile process and risk management?

RQ1 aims to establish the risk management barriers and risk issues in agile projects. The findings from the investigations reveal that risk management was not being performed in industry. This was supported from the initial qualitative investigation as explained in Section 3.3 and as summarized in Section 8.2(i).

In order to develop possible risk factors, issues that may be a threat to agile projects were investigated using existing literature with regards to any problem or difficulties in applying agile projects, as discussed in Section 3.5 and as summarized in Section 8.2(ii). Then, these issues were mapped to the sprint goals of the case study projects (Table 4-2). Later, these were translated into a set of risks as shown in Table 5-8.

Validation was carried out using the ART process model and its supporting prototype tool. Besides demonstrating the usability of the process and tool support, this also provides useful data in relation to (1) managing risk in agile project using the graphs produced and (2) development of guidelines for performing agile projects so as to implicitly reduce risks. Section 7.4 discussed some useful insights of the results obtained from both case studies.

Overall, the reason why risk management is not being performed in the industry is due to the extent of human effort needed to implement the complex steps in risk management and also human behaviour towards implementing risk management. As such, further investigation on agile risk management revealed that the main risk factors found involves the people and practices of agile projects. Using the two case studies empirical evidence was presented that provided evidence towards the feasibility and applicability of the ART model and tool support but also provided useful insights regarding team behaviour.

8.3.2 Research Question RQ2: Software Engineering (SE) Environment Data

RQ2a: Is it possible to support risk management using existing SE environment data to overcome barriers in RQ1a; if it is possible, which data is feasible to use?

RQ2b: Can data collection be conducted with minimal intrusion and effort?

This RQ2 relates to an investigation on Software Engineering (SE) environment data which focuses on the usage of Agile Project Management tool (APM). The details of the investigation on existing agile project management tools are explained in Section 4.2.2 and as summarized in Section 8.2(iii). The conclusion provides evidence that is that it is possible to use the project environment data to identify risk and so overcome the main barriers as defined in RQ1a. This supports the notion that, in agile processes, a lightweight risk management approach is required that automates some of the risk processes. As a result of this, a solution was developed using software agents to react to the project environment, based on designated rules in order to manage risk. As mentioned earlier, there are many environment data items that can be used as listed in Table 4-4. However, a tailorable approach is adopted where it is left up to the project manager to define which data is useful and related to which risks in their project.

Section 5.2 specifies the method used for both case studies evidencing that data collection conducted from both case studies involved minimal intrusion and effort, and with no cost involved. In the cases studies employed, the environment data used includes the student project data, in this instance from archived data. The data was stored in SVN repositories and was retrieved by the educator for use in the case study. Both case studies did not include any interaction with the participants in the study setting in order to meet the purpose of the study. This includes to identify risk in their project and to investigate compliance of the team member with agile practices. The collected data was validated by the Educator/Product Owner of the student's project based on discussion sessions prior to and after the implementation of the project.

The study revealed that it is possible to use existing SE environment data to support risk management. However, there are many environment data items that can be used to detect risk thus, acquiring project manager to define which ones that are related to their project. Further to this, the empirical evidence also revealed that data can be collected with minimal intrusion and effort.

8.3.3 Research Question RQ3: Software Agents and Rule Engine

> *RQ3a: Can software agents coupled with a rule engine provide a means to automate risk management in agile projects using data from the Software Development environment?*

RQ3b: Is an approach using Agents operating in the Software Development Environment useful?

RQ3 refers to part of the development of the proposed solution approach and tool support. The initial development of software agents is described in Section 4.1.1.2 followed by the development of the rule engine in Section 4.1.1.3. A prototype tool was developed in order to validate the interaction between agents, agents' compliance with the designated rules and how agents react to changes in project environment data. This is discussed throughout Section 4.3 starting from defining the input, processing the input and producing the output. Here, a walkthrough of the ART process is provided supported by the prototype tool. This demonstrates that software agents coupled with a rule engine can automate risk management using data from the project environment.

As far as human intervention is concerned, this proposed solution approach provides a means to minimize the human effort in managing risk. Thus, it helps to overcome the problem of needing highly intensive effort to manage risk. Once the input for the ART process model has been defined, the ART agents offer an effective and lightweight risk management starting from identifying risk, assessing and prioritizing risk as well as monitoring risk. The ART agents will trigger risk based on the rules defined by the project manager and react to dynamic changes in its environment as they are detected. Thus, this provides evidence of the usefulness of the solution approach.

8.4 Discussion on Research Contributions

The primary objective and main contribution of this research was the development of the new proposed approach model and tool support, Agile Risk Tool (ART). The model provides significant characteristics that are not found in other methods. These characteristics are embedded in the ART process model and include the definition of project or sprint goals and subsequent decomposing of these goals into problem scenarios that threaten the goals and so the project. Later, risk indicators are defined in order to monitor if the goals are not met. The model uses a template called the Rule template which enables the project manager to define the project or sprint goals, possible problem scenarios and the risk indicators for the scenarios. The template also allows the project manager to specify the environment data from which they can trigger the risk identification. The information in the Rule template is useful in terms of brainstorming the possible risk events in a project and to monitors it once it has been

input to the ART model. The ART model and the tool support are novel contributions which have been validated here using case studies, along with the observation that they seem to improve knowledge on managing risks. The ART model and its process is discussed in Chapter 4.

Besides its unique characteristics, the ART model and tool support contributes to an implementation of Continuous Risk Management (CRM) as described by the SEI (SEI, 2008). As discussed in Section 2.4.1, the CRM model provides the fundamental theory on how the risks can be managed continuously throughout the project lifecycle. The cycle includes the following elements; identify, analyze, plan, track, control and communicate all of which are ongoing until the project ends. However, this model only provides a theoretical concept and has to date mainly required manual input and implementation (Dorofee et. al., 1996). The ART model and tool support provide a way where the CRM cycle is done intelligently thus reducing human effort in managing risks. This claim was discussed further in Chapter 4.

The research also contributes in supporting three main areas in software engineering as discussed below:

(i) A New Lightweight Risk Management Approach

The ART process and tool constitute a new lightweight risk management approach that helps to reduce the barriers stated following from RQ1. The new process involves four out of six steps of the Boehm's definition of risk management (Boehm, 1989): risk identification, risk assessment, risk prioritization and risk monitoring. The other two steps, risk management planning and risk resolution can be added as an extension in future. This new approach contributes to towards an improved more realistic risk management discipline.

(ii) Risk Management Addition to Agile development

Agile development processes claim to reduce software risks inherently. They do not claim to remove risks completely and the emphasis on lightweight approaches could mean resistance to the addition of more process. This research includes investigation on possible environment data in agile projects that can be used to support risk management. The defined data arising from answering RQ2 is collected non-intrusively, without hurting

agility. Introducing a more lightweight risk management method attached to an agile process is novel and could serve as an enabler for software companies to adopt agile without disregarding risk management.

(iii) Contribution to Autonomous Computing

Autonomy of software is sought widely in many other areas like network and web applications, as well as in business and finance. This approach however, to the author's knowledge and understanding has never been applied in risk management. This research offers use of software agents to provide continuous risk management. In addition, the resulting risk management process is naturally lightweight since each software agent is design to achieve a designated goal i.e. to identify, assess, prioritize or monitor risk. Answering RQ3 led to using designated software agents to facilitate the risk management process. Therefore, this research work demonstrates the potential of autonomous computing being applied to risk management. Software agents have been used to assist the human oriented and complex risk management process.

In general, to the author knowledge, this research work is the first to integrate the software engineering activities of risk management, agile development and use of software agents. Comparatively, one approach proposed by Lorenz et. al. (2005) discussed in Chapter 3, include the idea which integrates risk management and software agents in decision making. However, the examples illustrated include initial feasibility study where no prototype is developed to support its validation. This research work improves this as it extends and focuses on managing risk in agile projects. In addition, the ART prototype is developed to validate the approach and real project data is used in the case studies.

The application of the ART model and tool support in the two case studies imply that risk can be easily managed and data in relation to the risk can be collected using non-intrusive methods. The process simplifies how the data is collected and improves the process using ART model and tool support as described in Chapter 5.

This research work also provides empirical evidence of the feasibility and applicability of the ART model and tool support. Besides, it also provides advantages and disadvantages of its application. The ART model and tool support were carried out in

two case studies of groups of student project at Queen's University Belfast using their archived project data. The results discussed in Chapter 6 and 7, produced various interesting risks results associated with people and process, both of importance in agile methods. The results analysis has contributed to evidence on how non-compliance with established agile practices generates risk and affects product quality. It also provides useful insights regarding team behaviour in the project. For example on planning and task estimation, team formation was discussed as well as the relationship of product quality with three variables: task completion, team performance and pair programming.

The Risk Burndown Chart technique discussed in Chapter 2 normally shows a downward risk exposure graph computed from the probability and size of potential loss in every sprint. However, that established technique does not show or visualize specifically what risk is being identified, assessed and monitored in the project. The graphs produced in this work show the increasing and decreasing pattern of risk detected for each team in every sprint enabling highlighting of some peaks at certain days thus giving an alternative visualization of risk.

As far as the novelty of this contribution is concerned, this work has added to the related work discussed earlier in Section 3.6. This work has expanded the investigation of risk management practices as developed by Coyle and Conboy (2009). The initial study carried out evidently showed that Software Risk Management is not being fully applied and the major problem seems to be the extent of human effort required. This work also has built on the work of Machado and Pereira (2012) which proposes expert system that focuses on risk identification phase of risk management. ART has provided extension of risk management phase that include not only risk identification phase but also risk analysis and prioritization and risk monitoring using software agents. The work is similar to Lamersdorf et. al. (2011) in respect of using logical rules to capture and identify risks but different in that ART agents capture risk, analyse the risk and provide mode of monitoring the risk. This includes the TRS and RFP metrics that respectively provide calculation of risk score and risk factor point s tailored to the size of the task. The same applies to the work of Nyfjord and Kajko-Mattson (2008) which proposes a model that combines agile process with risk management; however this work is different where the proposed solution towards agile risk management adopted software agents to ensure the approach remains lightweight. The ART provides an alternative to "Pocahontas" (Masticola, 2007) and ATMs (Ramamoorthy et. al., 1993) in that ART provides risk identification, analysis, prioritization and monitoring of risk related to

people and process based on data collected in agile project environment. ART has some similarities to the agent based system of Nienaber and Barnard's (2007), but focuses on one specific area in software project management (SPM) that is managing risk specifically in agile projects.

8.5 Limitations

Due to the fact that the study presents a new solution approach, several limitations exist arising in the implementation and validation of the approach. There are some concerns with regards to setting up the approach for the first time. This includes manual collection and translation of data from the artefacts to the ART tool and this effort might be seen as effort-intensive and requiring extra time at the first set up. Due to this, there might be some arguments on the worthiness of applying such approach, especially when a small project is involved. Although the risk management approach has been designed to be applicable in general to an agile project environment, this approach might be more suitable for a newly set up agile team were the agile practices and experience are still at the infancy stage. Additionally, in a large project context where the project contains huge number of project or sprint goals and risk factors the complexity of the approach increases.

There are also possibilities that the ART process model and tool support needs to be tailored according to the specific project needs. At the first instance, the agile development team needs to initialize their project scope and plan, including defining generic risks that can possibly happen according to their defined project goals. These generic risks can be used as a guideline where a more specific risk can be established later throughout the project.

In terms of adaptability, the validation of the solution approach was implemented using case studies of student projects and not employed in a real world environment. Therefore, one may argue that the solution approach may not be relevant to an established project or to a real world project.

8.6 Future Work

The ART process model and tool support provide a new means of managing risks in agile projects. The main advantage of this model is that it provides a simple and lightweight method to manage risk in a dynamic software development project, despite of an intensive effort in defining the input for the process. This model supports early identification of risks, leaving aside the designated ART agents to manage risks.

The ART model and tool support has great potential to evolve starting from extending the risk management process itself. This research work has demonstrated that software agents are usable and helpful in reducing the human effort in managing risk. Due to the fact that the approach is new and tool support is still only as a prototype, the initial implementation of this work includes only four main steps of risk management. Other risk management steps that are not included in this work i.e. risk management planning and risk resolution. This implies that there is a need to expand this model by adding these risk management steps.

Despite the usage of the software agents to manage risk, the research work also aims at expanding the agents' behavior. This includes giving the software agents ability to continuously manage risk as well as the necessary intelligence to make judgments and decision regarding risks. Adding intelligence equivalent to a human risk manager would be very ambitious. However, it is hoped that in future this research work could move towards this. One possible approach is to use previous risk data from the same or similar project in order to support current risks as well as to speculate on future risks. The agents are able to use past data in order to predict the risks for the specific project that has the same characteristics.

In Chapter 6 and 7 some discussion was given of preliminary results on comparing means and team performance between sprints. The indication was that teams improved in terms of managing risk between sprints one and two. In future, it would be interesting to carry out some detailed statistical analysis on this topic. A more extensive quantitative analysis of behaviour would be required to establish firm correlation between team behaviour and practices and code quality. It would be even more interesting if this analysis were done in an industrial setting.

Last but not least, in order to comprehend the physical implementation of the ART model and tool support, there is a need to integrate this with existing Agile Project

Management tools, perhaps as a plug-in, so that automated risk management can be fully realised. This would allow more practical risk management whereby while a project runs in the foreground, software agents are in the background ready to manage risks. In addition, this will also develop more promising support towards Continuous Risk Management (CRM) as discussed earlier in Section 4.3.3 as well as will opens more opportunities for experiments in the industrial setting.

8.7 Conclusions

This research work provides several significant investigations on the problems and issues in risk management specifically in agile projects. The development of the ART model and tool support has been demonstrated to help by at least reducing the problems previously identified with risk management.

The ART model moves the body of knowledge forward via novel contributions towards building a reliable model of risk management. The approach is necessarily supported by a prototype tool which has been shown to manage risks in example agile projects. The role of risk management in iterative and agile processes has to date been neglected but this model integrates risk management model with agile methods in a way that does not bloat the agile process.

Generally, this research work initiates a new method in effectively managing risks in agile projects. The approach is some distance off being implemented commercially but this remains a possibility, where the research work can be realistically applied in the industry.

References

Ahmed, A., Kayis, B. & Amornsawadwatana, S. 2007, "A review of techniques for risk management in projects", *Benchmarking: An International Journal*, vol. 14, no. 1, pp. 22-36.

Ambler, S. 2006, Results from Scott Ambler's 2006 Agile Adoption Rate Survey, www.ambysoft.com.

Ambler, S. 2008, Results from Scott Ambler's 2008 Agile Adoption Rate Survey, www.ambysoft.com.

Azizyan, G., Magarian, M.K. & Kajko-Matsson, M. 2011, "Survey of Agile Tool Usage and Needs", *Agile Conference (AGILE), 2011*IEEE, pp. 29.

Bandyopadhyay, K., Mykytyn, P.P. & Mykytyn, K. 1999, "A framework for integrated risk management in information technology", *Management Decision*, vol. 37, no. 5, pp. 437-4location.

Bannerman, P.L. 2008, "Risk and risk management in software projects: A reassessment", *Journal of Systems and Software*, vol. 81, no. 12, pp. 2118-2133.

Basili, V.R. 1993, "The experience factory and its relationship to other improvement paradigms" in *Software Engineering—ESEC'93* Springer, pp. 68-83.

Beck, K. & Fowler, M. "Planning extreme programming. 2001".

Beck, K. 2000, "extreme programming eXplained: embrace change Addison-Wesley", *Reading, Ma*.

Behrens, P. & Consulting, T.R. 2006, "Agile Project Management (APM) tooling survey results", *Trail Ridge consulting*.

Bell, T.E. 1989, "Managing Murphy's law: engineering a minimum-risk system", *Spectrum, IEEE*, vol. 26, no. 6, pp. 24-27.

Bellifemine, F.L., Caire, G. & Greenwood, D. 2007, *Developing multi-agent systems with JADE*, Wiley.com.

Benefield, G., Deemer, P., Larman, C. & Vodde, B. 2006, "The Scrum Primer".

Beynon-Davies, P. 1995, "Information systems failure and risk assessment: the case of the London Ambulance Service Computer Aided Despatch System", *European Conference on Information Systems*, pp. 1153.

Boehm, B. & Bose, P. 1994, "A collaborative spiral software process model based on theory W", *Software Process, 1994.'Applying the Software Process', Proceedings, Third International Conference on the* IEEE, pp. 59.

Boehm, B. & Turner, R. 2003, "Using risk to balance agile and plan-driven methods", *Computer,* vol. 36, no. 6, pp. 57-66.

Boehm, B. & Turner, R. 2005, "Management challenges to implementing agile processes in traditional development organizations", *Software, IEEE,* vol. 22, no. 5, pp. 30-39.

Boehm, B. 2006, "Some future trends and implications for systems and software engineering processes", *Systems Engineering,* vol. 9, no. 1, pp. 1-19.

Boehm, B., Egyed, A., Kwan, J., Port, D., Shah, A. & Madachy, R. 1998, "Using the WinWin spiral model: a case study", *Computer,* vol. 31, no. 7, pp. 33-44.

Boehm, B.W. 1988, "A spiral model of software development and enhancement", *Computer,* vol. 21, no. 5, pp. 61-72.

Boehm, B.W. 1989, *Tutorial: Software risk management,* IEEE Computer Society Press.

Boehm, B.W. 1991, *Software risk management: principles and practices*.

Boehm, B.W. 2007, ""*Some Future Trends and Implications for Systems and Software Engineering Processes*" Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research, R.W. Selby, ed., IEEE CS Press-John Wiley & Sons", pp. 545-571.

Boehm, B.W., Brown, J.R. & Lipow, M. 1976, "Quantitative evaluation of software quality", *Proceedings of the 2nd international conference on Software engineering* IEEE Computer Society Press, pp. 592.

Bourret, R. 1999, *XML and Databases.*

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. & Mylopoulos, J. 2002, "Modeling early requirements in Tropos: a transformation based approach" in *Agent-Oriented Software Engineering II* Springer, pp. 151-168.

Brooks, F.P. 1986, "No Silver Bullet—Essence and Accidents of Software Engineering Information", *Processing 86. H.J. Kugler, ed. Amsterdam: Elsevier Science Publishers B.V. (North Holland): 1069-1076.*

Brown, N. 1996, "Industrial-strength management strategies", *Software, IEEE,* vol. 13, no. 4, pp. 94-103.

Bustard, D., Wilkie, G. & Greer, D. 2013, "The Maturation of Agile Software Development Principles and Practice: Observations on Successive Industrial Studies in 2010 and 2012", *Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the* IEEE, pp. 139.

Cao, L., Mohan, K., Xu, P. & Ramesh, B. 2004, "How extreme does extreme programming have to be? Adapting XP practices to large-scale projects", *System*

Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on IEEE, pp. 10.

Carter, E., Hancock, T., Morin, J. & Robins, N. 1994, "Introducing RISKMAN methodology", *NCC Blackwell*.

Chakradhar, K. 2009, *Risk Management in Agile Development*, White paper edn, Polaris Software Lab Limited.

Charette R. N., O'Brien P. 1999, *Draft IEEE P1540 Risk Management Standard – Process and Implications. [Online] URL: http://www.psmsc.com/UG1999/Presentations/Software%20Risk%20Management%20Standard%20Update.pdf.*

Charette, R.N. 1989, *Software engineering risk analysis and management*, Intertext Publications.

Charette, R.N. 1991, *Applications strategies for risk analysis*, Intertext Publications, Inc., McGraw-Hill, Inc.

Charette, R.N. 1996, "Large-scale project management is risk management", *Software, IEEE*, vol. 13, no. 4, pp. 110-117.

Charette, R.N. 2005, *Why Software Fails, IEEE Spectrum*. Available: http://spectrum.ieee.org/computing/software/why-software-fails [2013, November].

Charette, R.N. 2006, "A Risk of Too Many Risk Standards?", *Sixteenth Annual International Symposium of the International Council On Systems Engineering (INCOSE)*, 8 - 14 July.

Cho, J. 2008, "Issues and Challenges of agile software development with SCRUM", *Issues in Information System*, vol. 9, no. 2.

Cockburn, A. & Highsmith, J. 2001, "Agile software development, the people factor", *Computer*, vol. 34, no. 11, pp. 131-133.

Cockburn, A. 2004, *Crystal clear: a human-powered methodology for small teams*, Pearson Education.

Cohn, M. 2005, *Agile estimating and planning*, Pearson Education.

Cohn, M. 2010, *Succeeding with agile: software development using Scrum*, Pearson Education.

Cohn, M. 8 April 2010, *Managing Risk on Agile Projects with the Risk Burndown Chart*.

Conboy, K., Coyle, S., Wang, X. & Pikkarainen, M. 2010, "People over process: key people challenges in agile development", *IEEE Software*.

Coyle, S. & Conboy, K. 2009, "A case study of risk management in agile systems development".

Craft, R., Vandewart, R., Wyss, G. & Funkhouser, D. 1998, *An open framework for risk management*.

Dardenne, A., Van Lamsweerde, A. & Fickas, S. 1993, "Goal-directed requirements acquisition", *Science of computer programming*, vol. 20, no. 1, pp. 3-50.

Darke, P., Shanks, G. & Broadbent, M. 1998, "Successfully completing case study research: combining rigour, relevance and pragmatism", *Information systems journal*, vol. 8, no. 4, pp. 273-289.

De Bakker, K., Boonstra, A. & Wortmann, H. 2010, "Does risk management contribute to IT project success? A meta-analysis of empirical evidence", *International Journal of Project Management*, vol. 28, no. 5, pp. 493-503.

Dedolph, F.M. 2003, "The neglected management activity: software risk management", *Bell Labs Technical Journal*, vol. 8, no. 3, pp. 91-5.

Deemer, P., Benefield, G., Larman, C. & Vodde, B. 2010, "The scrum primer", *Scrum Primer is an in-depth introduction to the theory and practice of Scrum, albeit primarily from a software development perspective, available at: http://assets.scrumtraininginstitute.com/downloads/1/scrumprimer121.pdf*, vol. 1285931497.

Dey, P.K., Kinch, J. & Ogunlana, S.O. 2007, "Managing risk in software development projects: a case study", *Industrial Management & Data Systems*, vol. 107, no. 2, pp. 284-303.

Dorofee, A.J., Walker, J.A., Alberts, C.J., Higuera, R.P. & Murphy, R.L. 1996, *Continuous Risk Management Guidebook.*

Dumas, J.S. 1999, *A practical guide to usability testing*, Intellect Books.

Easterbrook, S., Singer, J., Storey, M. & Damian, D. 2008, "Selecting empirical methods for software engineering research" in *Guide to advanced empirical software engineering* Springer, , pp. 285-311.

Ewusi-Mensah, K. & Przasnyski, Z.H. 1991, "On information systems project abandonment: an exploratory study of organizational practices", *MIS quarterly*, pp. 67-86.

Fairley, R. 1994, "Risk management for software projects", *IEEE Software*, vol. 11, no. 3, pp. 57-67.

Fitzgerald, B., Hartnett, G. & Conboy, K. 2006, "Customising agile methods to software practices at Intel Shannon", *European Journal of Information Systems*, vol. 15, no. 2, pp. 200-213.

Foo, S. & Muruganantham, A. 2000, "Software risk assessment model", *Management of Innovation and Technology, 2000. ICMIT 2000. Proceedings of the 2000 IEEE International Conference on* IEEE, pp. 536.

Fowler, M. & Highsmith, J. 2001, "The agile manifesto", *Software Development*, vol. 9, no. 8, pp. 28-35.

Freimut, B., Hartkopf, S., Kaiser, P., Kontio, J. & Kobitzsch, W. 2001, "An industrial case study of implementing software risk management", *Joint 8th European*

*Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9)*ACM, Vienna, Austria, 10-14 Sept. 2001, pp. 277.

Friedman-Hill, E. 2003, *JESS in Action,* Manning Greenwich, CT.

Gemmer, A. 1997, "Risk management: moving beyond process", *Computer,* vol. 30, no. 5, pp. 33-43.

Gilb, T. 1988, "Principles of software engineering management".

Glass, R.L. 1998, *Software runaways: monumental software disasters,* Prentice-Hall, Inc.

Gotterbarn, D. & Rogerson, S. 2005, "Responsible risk analysis for software development: creating the software development impact statement", *Communications of the Association for Information Systems (Volume 15, 2005),* vol. 730, no. 210, pp. 750.

Greer, D. & Conradi, R. 2009, "Software project initiation and planning-an empirical study", *Software, IET,* vol. 3, no. 5, pp. 356-368.

Hall, E.M. 1998, *Managing risk: Methods for software systems development,* Pearson Education.

Han, W. & Huang, S. 2007, "An empirical analysis of risk components and performance on software projects", *Journal of Systems and Software,* vol. 80, no. 1, pp. 42-50.

Harold, E.R. 2004, *XML 1.1 Bible*, Wiley.com.

Highsmith, J.A. 2002, *Agile software development ecosystems,* Addison-Wesley Professional.

Higuera, R.P. & Haimes, Y.Y. 1996, *Software Risk Management*.

Hossain, E., Babar, M.A., Paik, H. & Verner, J. 2009, "Risk identification and mitigation processes for using Scrum in global software development: A conceptual framework", *Software Engineering Conference, 2009. APSEC'09, Asia-Pacific,* IEEE, pp. 457.

Huang, S. & Han, W. 2008, "Exploring the relationship between software project duration and risk exposure: A cluster analysis", *Information & Management,* vol. 45, no. 3, pp. 175-182.

Ibbs, C.W. & Kwak, Y.H. 2000, "Assessing project management maturity", *Project Management Journal,* vol. 31, no. 1, pp. 32-43.

*IEEE 1540 Standard for Lifecycle Processes-Risk Management. The Institute of Electrical and Electronics Engineers Inc., New York, NY.*2001.

IEEE Guide Adoption of PMI Standard a Guide to the Project Management Body of Knowledge, *IEEE Std 1490-2003 (Revision of IEEE Std 1490-1998)* , vol., no., pp.0_1,216, 2004", .

Islam, S. 2009, "Software development risk management model: a goal driven approach", *Proceedings of the doctoral symposium for ESEC/FSE on Doctoral symposium*, ACM, pp. 5.

Islam, S. 2011, *Software development risk management model: a goal-driven approach*.

Jeffries, R.E. 2001, *Extreme programming installed*, Addison-Wesley Professional.

Jung, H., Kim, S. & Chung, C. 2004, "Measuring software product quality: A survey of ISO/IEC 9126", *Software, IEEE*, vol. 21, no. 5, pp. 88-92.

Kajko-Mattsson, M., Lundholm, J. & Norrby, J. 2009, "Insight into Risk Management in Five Software Organizations", *Software Engineering and Advanced Applications, 2009. SEAA'09. 35th Euromicro Conference on* IEEE, pp. 321.

Kan, S.H. 2002, *Metrics and models in software quality engineering*, Addison-Wesley Longman Publishing Co., Inc.

Karolak, D. 1998, "Software engineering risk and Just-In-Time development", *International Journal of Computer Science and Information Management*, vol. 1, no. 4, pp. 19-28.

Keil, M., Cule, P.E., Lyytinen, K. & Schmidt, R.C. 1998, "A framework for identifying software project risks", *Communications of the ACM*, vol. 41, no. 11, pp. 76-83.

Keshlaf, A.A. & Hashim, K. 2000, *A model and prototype tool to manage software risks*.

Kirkpatrick, R.J., Walker, J.A. & Firth, R. 1992, "Software development risk management: an SEI appraisal", *SEI Technical Review*, vol. 92.

Kitchenham, B. & Pfleeger, S.L. 1996, "Software quality: the elusive target [special issues section]", *Software, IEEE*, vol. 13, no. 1, pp. 12-21.

Kitchenham, B., Pickard, L. & Pfleeger, S.L. 1995, "Case studies for method and tool evaluation", *Software, IEEE*, vol. 12, no. 4, pp. 52-62.

Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., El Emam, K. & Rosenberg, J. 2002, "Preliminary guidelines for empirical research in software engineering", *Software Engineering, IEEE Transactions on*, vol. 28, no. 8, pp. 721-734.

Kontio, J. & Basili, V.R. 1997, "Empirical evaluation of a risk management method", *Proceedings of the SEI Conference on Risk Management*.

Kontio, J. 1997, "The RISKIT method for software risk management, version 1.00", *Computer Science Technical Reports, University of Maryland, College Park, MD, USA*.

Kontio, J. 1999, "Risk management in software development: A technology overview and the riskit method", *Proceedings of the 21st international conference on Software engineering*, ACM, pp. 679.

Kontio, J. 2001, *Software Engineering Risk Management: A Method, Improvement Framework, and Empirical Evaluation. Ph.D. Thesis*, Department of Computer Science and Engineering, Helsinki University of Technology, Finland.

Kontio, J. 2002, ""Risk Management: What went wrong and what is the new agenda?," in Kontio, J. and Conradi, R. (Eds.) Software Quality - ECSQ 2002, Quality Connection - 7th European Conference on Software Quality -- Conference Notes Helsinki: Center for Excellence Finland".

Kontio, J., Getto, G. & Landes, D. 1998, "Experiences in improving risk management processes using the concepts of the Riskit method", *ACM SIGSOFT Software Engineering Notes*, vol. 23, no. 6, pp. 163-174.

Kruchten, P. 2004, *The rational unified process: an introduction*, Addison-Wesley Professional.

Kwak, Y. & Stoddard, J. 2004, "Project risk management: lessons learned from software development environment", *Technovation*, vol. 24, no. 11, pp. 915-920.

Lamersdorf, A., Munch, J., Fernández-del Viso Torre, A. & Rebate Sanchez, C. 2011, "A risk-driven model for work allocation in global software development projects", *Global Software Engineering (ICGSE), 2011 6th IEEE International Conference on* IEEE, pp. 15.

Larman, C. & Basili, V.R. 2003, "Iterative and incremental developments. A brief history", *Computer*, vol. 36, no. 6, pp. 47-56.

Layman, L., Williams, L. & Cunningham, L. 2006, "Motivations and measurements in an agile case study", *Journal of Systems Architecture*, vol. 52, no. 11, pp. 654-667.

Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L. & Zelkowitz, M. 2002, "Empirical findings in agile methods" in *Extreme Programming and Agile Methods—XP/Agile Universe 2002* Springer, pp. 197-207.

Lorenz, M., Gehrke, J.D., Langer, H., Timm, I.J. & Hammer, J. 2005, "Situation-aware risk management in autonomous agents", *Proceedings of the 14th ACM international conference on Information and knowledge management*, ACM, pp. 363.

Machado, J.P., SL. 2012, "Automatic Risk Identification in Software Projects: An Approach Based on Inductive Learning", *Intelligent Information Management*, vol. 4, pp. 291-295.

Marcal, A.S.C., de Freitas, Bruno Celso C, Furtado Soares, F. & Belchior, A.D. 2007, "Mapping CMMI project management process areas to scrum practices", *Software Engineering Workshop, 2007. SEW 2007. 31st IEEE*, pp. 13.

Martin, J. 1991, *Rapid application development*, Macmillan Publishing Co., Inc.

Masticola, S.P. 2007, "Lightweight risk mitigation for software development projects using repository mining", *Proceedings of the Fourth International Workshop on Mining Software Repositories* IEEE Computer Society, pp. 13.

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

Melnik, G. & Maurer, F. 2006, "Comparative analysis of job satisfaction in agile and non-agile software development teams" in *Extreme Programming and Agile Processes in Software Engineering* Springer, pp. 32-42.

Melo, C., Cruzes, D.S., Kon, F. & Conradi, R. 2011, "Agile team perceptions of productivity factors", *Agile Conference (AGILE), 2011* IEEE, pp. 57.

Menzies, T., Williams, S., Elrawas, O., Baker, D., Boehm, B., Hihn, J., Lum, K. & Madachy, R. 2009, "Accurate estimates without local data?", *Software Process: Improvement and Practice,* vol. 14, no. 4, pp. 213-225.

Misra, S., Kumar, V. & Kumar, U. 2006, "Different Techniques for Risk Management In Software Engineering: A Review", *Conference of the Administrative Sciences Association of Canada (ASAClocation).*

Misra, S.C., Kumar, V. & Kumar, U. 2005, "Modeling Strategic Actor Relationships to Support Risk Analysis and Control in Software Projects" *ICEIS (3),* pp. 288.

Misra, S.C., Kumar, V. & Kumar, U. 2009, "Identifying some important success factors in adopting agile software development practices", *Journal of Systems and Software,* vol. 82, no. 11, pp. 1869-1890.

Murata, M., Lee, D., Mani, M. & Kawaguchi, K. 2005, "Taxonomy of XML schema languages using formal language theory", *ACM Transactions on Internet Technology (TOIT),* vol. 5, no. 4, pp. 660-704.

Nelson, C.R., Taran, G. & de Lascurain Hinojosa, L. 2008, "Explicit risk management in agile processes" in *Agile Processes in Software Engineering and Extreme Programming,* Springer, pp. 190-201.

Nerur, S., Mahapatra, R. & Mangalaraj, G. 2005, "Challenges of migrating to agile methodologies", *Communications of the ACM,* vol. 48, no. 5, pp. 72-78.

Nielsen, J. 1994, "Usability inspection methods", *Conference companion on Human factors in computing systems,* ACM, pp. 413.

Nienaber, R.C. & Barnard, A. 2007, "A Generic Agent Framework to Support the Various Software Project Management Processes", *Interdisciplinary Journal of Information, Knowledge, and Management,* vol. 2, pp. 149-162.

Nienaber, R.C. 2009, *A model for enhancing software project management using software agent technology*.

Nyfjord, J. & Kajko-Mattsson, M. 2007, "Commonalities in risk management and agile process models", *Software Engineering Advances, 2007. ICSEA 2007. International Conference on* IEEE, pp. 18.

Nyfjord, J. & Kajko-Mattsson, M. 2008, "Integrating risk management with software development: State of practice", *Proceedings, IAENG International Conference on Software Engineering, BrownWalker Press, Boca Raton, USA* Citeseer.

Odzaly, E.E., Greer, D. & Sage, P. "Software risk management barriers: An empirical study", *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd Interlocation*.

Padayachee, K. "An interpretive study of software risk management perspectives", *Proceedings of the 2002 annual research conference of the South African institute of computer slocation*.

Palmer, S. & Felsing, M. "A Practical Guide to Feature Driven Development 2002".

Patterson, F.D. & Neailey, K. 2002, "A risk register database system to aid the management of project risk", *International Journal of Project Management,* vol. 20, no. 5, pp. 365-374.

Paulk, M.C. 2001, "Extreme programming from a CMM perspective", *Software, IEEE,* vol. 18, no. 6, pp. 19-26.

Paulk, M.C. 2002, "Agile methodologies and process discipline", *Institute for Software Research,* pp. 3.

Pfleeger, S.L. 2000, "Risky business: what we have yet to learn about risk management", *Journal of Systems and Software,* vol. 53, no. 3, pp. 265-273.

Poison, P.G., Lewis, C., Rieman, J. & Wharton, C. "Cognitive Walkthroughs: A Method for Theory-Based Evaluation of User Interfaces".

Poppendieck, M. & Poppendieck, T. 2006, *Implementing lean software development: From concept to cash (the addison-wesley signature series),* Addison-Wesley Professional.

PRINCE2 2002, *Managing successful projects with PRINCE2, Great Britain. Office of Government Commerce,* TSO Shop.

Project Management Institute 2008, "A Guide to the Project Management Body of Knowledge: PMBOK® Guide", Project Management Institute.

Rabbi, M.F. & Bin Mannan, K. 2008, "A review of software risk management for selection of best tools and techniques", *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPD'08. Ninth ACIS International Conference on* IEEE, pp. 773.

Ramamoorthy, C., Chandra, C., Ishihara, S. & Ng, Y. 1993, "Knowledge based tools for risk assessment in software development and reuse", *Tools with Artificial Intelligence, 1993. TAI'93 Proceedings, Fifth International Conference on* IEEE, pp. 364.

Ramesh, B., Cao, L. & Baskerville, R. 2010, "Agile requirements engineering practices and challenges: an empirical study", *Information Systems Journal,* vol. 20, no. 5, pp. 449-480.

Raz, T. & Hillson, D. 2005, "A comparative review of risk management standards", *Risk Management,* pp. 53-66.

Ribeiro, L., Gusmão, C., Feijó, W. & Bezerra, V. 2009, "A case study for the implementation of an agile risk management process in multiple projects

environments", *Management of Engineering & Technology, 2009. PICMET 2009. Portland International Conference on* IEEE, pp. 1396.

Riehle, R. 2007, "Institutional memory and risk management", *ACM SIGSOFT Software Engineering Notes*, vol. 32, no. 6, pp. 5.

Rising, L. & Janoff, N.S. 2000, "The Scrum software development process for small teams", *Software, IEEE*, vol. 17, no. 4, pp. 26-32.

Ropponen, J. & Lyytinen, K. 1997, "Can software risk management improve system development: an exploratory study", *European Journal of Information Systems*, vol. 6, no. 1, pp. 41-50.

Ropponen, J. & Lyytinen, K. 2000, *Components of software development risk: how to address them? A project manager survey*.

Rothfeder, J. 1998, *It's Late, Costly, and Incompetent – But Try Firing a Computer System*, Business Week, Nov. 7, 164-165 Edn.

Royce, W.W. 1970, "Managing the development of large software systems", *proceedings of IEEE WESCON*Los Angeles.

Rubin, K.S. 2012, *Essential scrum: a practical guide to the most popular agile process*, Addison-Wesley.

Rumpe, B. & Schröder, A. 2002, "Quantitative survey on extreme programming projects", *Third International Conference on Extreme Programming and Flexible Processes in Software Engineering, XP2002, May*, pp. 26.

Runeson, P. & Höst, M. 2009, "Guidelines for conducting and reporting case study research in software engineering", *Empirical Software Engineering*, vol. 14, no. 2, pp. 131-164.

Salo, O. & Abrahamsson, P. 2008, "Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum", *Software, IET*, vol. 2, no. 1, pp. 58-64.

Schatz, B. & Abdelshafi, I. 2005, "Primavera gets agile: a successful transition to agile development", *Software, IEEE*, vol. 22, no. 3, pp. 36-42.

Schwaber, K. & Beedle, M. 2002, *Agile software development with Scrum*, Prentice Hall Upper Saddle River.

Schwaber, K. 1995, "SCRUM Development Process. OOPSLA'95 Workshop on Business Object Design and Implementation", *Austin, USA*.

Seaman, C.B. 1999, "Qualitative methods in empirical studies of software engineering", *Software Engineering, IEEE Transactions on*, vol. 25, no. 4, pp. 557-572.

Smith, P.G. & Pichler, R. 2005, "Agile risks/Agile rewards", *Software Development*, vol. 13, no. 4, pp. 50-53.

*Software Engineering Institute/Carnegie Mellon University, Risk Management Overview*2013, Available: http://www.sei.cmu.edu/risk/. [2008]

Sommerville, I. 2004, "Software Engineering. International computer science series".

Sommerville, I. 2011, "Software Engineering, and Pearson edition".

Standards Australia and New Zealand (2004), 2004, *Australian/New Zealand Standard Risk Management AS/NZS 4360:2004*, 3rd Ed., Standards Australia/ New Zealand, Sydney/Wellington.

Stapleton, J. 1997, *DSDM Dynamic Systems Development Method: the method in practice*, Cambridge University Press.

Takeuchi, H. & Nonaka, I. 1986, "The new new product development game", *Harvard business review*, vol. 64, no. 1, pp. 137-146.

Thomas Van Raalte, Peter Purich 2009, *Oracle Fusion Middleware User's Guide for Oracle Business Rules 11g Release 1 (11.1.1)*.

Van der Vlist, E. 2011, *XML Schema: The W3C's Object-Oriented Descriptions for XML*, O'Reilly Media, Inc.

Van Deursen, A. & Kuipers, T. 2003, "Source-based software risk assessment", *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on* IEEE, pp. 385.

Van Solingen, R., Basili, V., Caldiera, G. & Rombach, H.D. 2002, "Goal Question Metric (GQM) Approach", *Encyclopedia of Software Engineering*.

Vanhanen, J. & Lassenius, C. 2005, "Effects of pair programming at the development team level: an experiment", *Empirical Software Engineering, 2005 International Symposium on* IEEE, pp.10.

VersionOne 2013, "*7th Annual State of Agile Development Survey"*, www.versionone.com.

Vijayasarathy, L. & Turk, D. 2008, "Agile software development: A survey of early adopters", *Journal of Information Technology Management*, vol. 19, no. 2, pp. 1-8.

Wallace, L., Keil, M. & Rai, A. 2004, "How software project risk affects project performance: An investigation of the dimensions of risk and an exploratory model*", *Decision Sciences*, vol. 35, no. 2, pp. 289-321.

Walsham, G. 1995, "Interpretive case studies in IS research: nature and method", *European Journal of information systems*, vol. 4, no. 2, pp. 74-81.

Westfall, L. 2001, "Software risk management", *ANNUAL QUALITY CONGRESS PROCEEDINGS-AMERICAN SOCIETY FOR QUALITY CONTROL*ASQ; 1999, pp. 32.

Willams, T.M. 1994, "Using a risk register to integrate risk management in project definition", *International Journal of Project Management*, vol. 12, no. 1, pp. 17-22.

Williams, R.C. 2003, *New Directions in Risk Management at the SEI*, Carnegie Mellon University.

Williams, R.C., Pandelios, G.J. & Behrens, S.G. 1999, *Software Risk Evaluation (SRE) Method Description: Version 2.0,* Carnegie Mellon University, Software Engineering Institute.

Williams, R.C., Walker, J.A. & Dorofee, A.J. 1997, "Putting risk management into practice", *Software, IEEE,* vol. 14, no. 3, pp. 75-82.

Wohlin, C., Host, M., Henningsson, K., 2003, "Empirical Research Methods in Software Engineering," *Empirical Methods and Studies in Software Engineering,* R. Conradi and A. Wang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp 7–23.

Wooldridge, M. & Jennings, N.R. 1995, "Intelligent agents: Theory and practice", Knowledge Engineering Review, vol. 10, no. 2, pp. 115-152.

Yin, R. K., 2009, "Case study research: Design and methods", vol. 5. SAGE Publications.

Appendices

Appendix A: Initial Qualitative Survey Questionnaire

QUESTIONNAIRE ON SOFTWARE RISK MANAGEMENT

Introduction

This survey aims to find out about the application of risk management in software development projects for randomly selected software companies at Northern Ireland. The objectives of the survey are to answer the Research Questions below:

1. What, if any, Software Risk Management is carried out in industry?
2. What is missing in the risk management discipline?
3. What barriers exist to performing software risk management?
4. What can be done to improve the current state of the art for risk management?"

The target respondents of this survey are those familiar with how software projects are managed in a company. It is important for us that you answer the questions as accurately and honestly as possible. Please be assured that will use the survey results for research purposes only and that we guarantee complete confidentiality. Any results published will be generalizations and will be reported so as not to identify individuals.

Glossary

Risk is a concept that denotes a potential negative impact to an asset or some characteristic of value that may arise from some present process or future event.

Software Risk Management (SRM) is a set of practices for software development projects to identify, prioritize, and manage risks.

Context Questions	
Name:	
Respondent job function/ role:	
Years of experience in project management and risk management:	
Number of employees in company:	
Number of software developers in company:	
Main company operations:	

Section 1: Questions on the awareness of SRM

Please tick the correct response in each case

1. I am fully aware of the application of Software Risk Management
   - ❑Strongly Agree
   - ❑Agree
   - ❑Neutral/Don't Know
   - ❑Disagree
   - ❑Strongly Disagree

2. My organization already has/had a process/ methodology for software risk management?
   ❑Yes ❑ No ❑Don't know
   If Yes, please name the process/ methodology

Section 2: Questions on the of SRM process

The main processes of SRM have been named as Risk Identification, Risk Analysis, Risk Prioritization, Risk Management Planning, Risk Resolution and Risk Monitoring.

- **Risk Identification** – produces list of the project-specific risk items likely to compromise a project's satisfactory outcome.
- **Risk Analysis** – produces assessments of probability of loss associated with the identified risk items.
- **Risk Prioritization** – produces prioritized ordering of the risk items identified and

analyzed.

- **Risk Management Planning** – produces plans of addressing each risk item.
- **Risk Resolution** – produces a situation in which the risk items are eliminated or otherwise resolved.
- **Risk Monitoring** – involves tracking the project's progress towards resolving its risks items and taking corrective action when appropriate.

1. My organization currently carries out (Please rate how widespread in your projects where; **Every** project= 100%, **Almost all** projects = 99 to 80%, **Some** projects = 79 to 60%, **A Few** projects= 59 to 40% and **Very Few** projects = less than 40%)

RM Step	**Performed?**	**If "Yes" how widespread is it in projects?**
Risk Identification	❑Yes ❑No	❑Every/ ❑Almost all ❑Some ❑A few ❑Very few
Risk Analysis	❑Yes ❑No	❑Every/ ❑Almost all ❑Some ❑A few ❑Very few
Risk Prioritization	❑Yes ❑No	❑Every/ ❑Almost all ❑Some ❑A few ❑Very few
Risk Management Planning	❑Yes ❑No	❑Every/ ❑Almost all ❑Some ❑A few ❑Very few
Risk Resolution	❑Yes ❑No	❑Every/ ❑Almost all ❑Some ❑A few ❑Very few
Risk Monitoring	❑Yes ❑No	❑Every/ ❑Almost all ❑Some ❑A few ❑Very few

For those that were answered "Yes", please state how well it has been performed

RM Step	**Performed Well**	**Comment / Explanation**
Risk Identification	❑Strongly Agree ❑Agree ❑ Neutral/Don't Know ❑Disagree ❑Strongly Disagree	
Risk Analysis	❑Strongly Agree ❑Agree ❑ Neutral/Don't Know ❑Disagree ❑Strongly Disagree	
Risk Prioritization	❑Strongly Agree ❑Agree ❑ Neutral/Don't Know ❑Disagree ❑Strongly Disagree	
Risk Management Planning	❑Strongly Agree ❑Agree ❑ Neutral/Don't Know ❑Disagree ❑Strongly Disagree	

Risk Resolution	❑Strongly Agree ❑Agree ❑ Neutral/Don't Know ❑Disagree ❑Strongly Disagree	
Risk Monitoring	❑Strongly Agree ❑Agree ❑ Neutral/Don't Know ❑Disagree ❑Strongly Disagree	

For those steps not being performed or where you answered "Disagree" or "Strongly Disagree", please state why you think that the step is not performed or not performed well.

RM Step	**Reason Not Performed Well**
Risk Identification	
Risk Analysis	
Risk Prioritization	
Risk Management Planning	
Risk Resolution	
Risk Monitoring	

2. Thinking about the effort required to do SRM, what aspects of SRM are most effort intensive and why?

4. Which techniques or steps in SRM do you think are most complicated to put into practice and why?

Section 3: Questions on implementation of SRM

1. When each step in SRM is is performed (during pre-contract, requirements, design, implementation etc)?

RM Step	**When performed**	**Comment**
Risk Identification		

Risk Analysis		
Risk Prioritization		
Risk Management Planning		
Risk Resolution		
Risk Monitoring		

2. The following statements are about possible barriers to SRM. **Please rank them starting with "1" for the one you most agree with.**

Barriers	Rank (Highest = 1)	Comment (optional)
i. Visible (and tangible) development costs get more attention than intangibles like loss of net profit and downstream liability.		
ii. The value of risk management cannot easily be proved.		
iii. There are no resources available for SRM.		
iv. Risk management seems difficult or there are too many risks to handle.		
v. Project teams (and managers) see reward for problem-solving, not prevention		
vi. Mitigation actions may require organizational or process changes.		
vii. Discussing risks goes against cultural norms (e.g., bringing up potential issues is viewed as negative thinking or as causing conflict within the group.		
viii. Overconfidence (e.g. risks are already taken care of, implicitly).		
ix. Fatalism (e.g., software is always late anyway; there is no way to change that).		
x. Other Reasons		

Questions On Tool Usage

3. Are you using, have used or are aware of any tool support for each of the following steps

RM Step	**Tool support using/used**	**Tool support that you aware of**
Risk Identification		

Risk Analysis		
Risk Prioritization		
Risk Management Planning		
Risk Resolution		
Risk Monitoring		

(If you do not use a tool in SRM, please skip the remainder of this section)

4. Which tool do you use for SRM?

5. The tool used is helpful? (Please tick one statement)
   - ❑Strongly Agree
   - ❑Agree
   - ❑ Neutral/Don't Know
   - ❑Disagree
   - ❑Strongly Disagree

6. The tool used assists in managing the project timeline? (Please tick one statement)
   - ❑Strongly Agree
   - ❑Agree
   - ❑ Neutral/Don't Know
   - ❑Disagree
   - ❑Strongly Disagree

7. The tool used assists in managing the project budget? (Please tick one statement)
   - ❑Strongly Agree
   - ❑Agree
   - ❑ Neutral/Don't Know
   - ❑Disagree
   - ❑Strongly Disagree

8. The tool used helps to improve product quality?(Please tick one statement)
   - ❑Strongly Agree
   - ❑Agree
   - ❑Neutral/Don't Know
   - ❑Disagree
   - ❑Strongly Disagree

9. Who is the primary user(s) (job function) of the risk management tool?

10. How long does it take for a project manager/ engineer to learn to satisfactorily use the risk assessment tool?
    - ❑One day
    - ❑One week
    - ❑One month
    - ❑Other _____

Section 4: Questions on future/suggestion on SRM

1. Are you happy with your current risk management process/tool?
   ❑Yes ❑No
   If No, why not?

2. Do you have any suggestion/problem you would like to address regarding the risk assessment tool or SRM processes?

3. Are you willing to be contacted again about possible research collaboration on improving software risk management?
   ❑Yes ❑No Email: _____

Appendix B: Risk Exposure Matrix

AS/NZS 4260: 1999

Table B1: Qualitative risk exposure matrix – level of risk

Probability	Impact				
	Insignificant 1	Minor 2	Moderate 3	Major 4	Catastrophic 5
5 (almost certain)	H	H	E	E	E
4 (likely)	M	H	H	E	E
3 (moderate)	L	M	H	E	E
2 (unlikely)	L	L	M	H	E
1 (rare)	L	L	M	H	H

Note: The number of categories should reflect the needs of the study.

Legend:-
E: extreme risk; immediate action required
H: high risk; senior management attention needed
M: moderate risk; management responsibility must be specified
L: low risk; manage by routine procedures

Table B2: Qualitative measures of consequence or impact

Level	Descriptor	Example detail description
1	Insignificant	No injuries, low financial loss
2	Minor	First aid treatment, on-site release immediately contained, medium financial loss
3	Moderate	Medical treatment required, on-site release contained with outside assistance, high financial loss
4	Major	Extensive injuries, loss of production capability, off-site release with no detrimental effects, major financial loss
5	Catastrophic	Death, toxic release off-site with detrimental effect, huge financial loss

Note: Measures used should reflect the needs and nature of the organization and activity under study.

Table B3: Qualitative measures of likelihood or probability		
Level	**Descriptor**	**Example detail description**
1	Almost certain	Is expected to occur in most circumstances
2	Likely	Will probably occur in most circumstances
3	Possible	Might occur at some time
4	Unlikely	Could occur at some time
5	Rare	May occur only in exceptional circumstances
Note: These tables need to be tailored to meet the needs of an individual organization.		

Appendix C: Extreme Manager and Rally Software (Extended)

Table C1: Comparison of the objects and attributes in two agile project management tools; Extreme manager and Rally software

Scrum Process	Extreme Manager			Rally Software		
	Objects	Attributes	Values	Objects	Attributes	Values
Product	Product	• Product Internal Name • Product Market Name • Product Description • Parent Product • Unit Tests Results URL • Preferred Iteration Name Prefix • Product Iteration Start Day • Iteration duration in Weeks • Work Unit	Ideal Days , Gummi Bears	Project	• Name	
Product Backlog	Requirements/ User stories	• Requirement/User Story Id • User Defined Tracking Id • Title • Priority • Outline Description • Detailed Description • Reference and Support Information links • Attached documents for the	Highest, High, Normal, Low, Lowest	Backlog	• Rank • Id • Name • Plan Estimate • Priority • Owner • Accepted Date • Task Status	None, Resolve Immediately, High attention, Normal, Low

		requirement • Requirements Group • Requirements Type • Customer Reference • Customer Representative • Product • Parent Requirement • Status • Created By • Date of creation • Estimation in Ideal days (Gummy Bears) • Estimated by • Days taken to Implement • Assigned to • Developed by • Paired by • Development Started on • Development Completed On • Release • Iteration	 Bug Report, New Feature, Product Integration, Scalability/Operational, Technical Refinement Draft, Confirmed, Estimated, Pending/Scheduled, In Progress, Completed, Deleted, All		• Iteration • Parent • Project • Release	

Sprint Backlog	Task	• Task Id • User Defined Tracking Id • Title • Priority • Task Type • Outline Description • Detailed Description • Parent Task • Requirement • Status • Created By • Date of Creation • Estimation in Ideal Days • Days Taken to Implement • Assigned To • Developed By • Paired By • Development Started On • Development Completed On • Iteration • Release	New Development, Refactoring, Integration, Investigation, Quality Assurance, Documentation	User stories	• Rank • Id • Name • Release • Iteration • State • Plan Estimate • Task Estimate • To Do • Owner • Package	Defined, In Progress, Completed, Accepted
Release	Acceptance Test	• Requirement/User story • Title • Detailed Description • Detailed Failure Reason		Release	• Name • Theme • Start Date • Release Date	

		• Manual/Auto • Result Collector Class • Test System Link Id • Category • Status • Created By • Creation Date • Tested By • Testing Start Date • Testing Finish Date • Approved By • Approval Date	All, Created, In Progress, Closed, Deleted Unknown, Passed, Failed, Error, Blocked		• Resources • Plan Estimate • Task Estimate • To Do • State	Planning, Active, Accepted
	Release	• Product • Release Name • Business Name • Description • Release Start Date • Release Finish Date • Release Reference Velocity	Past Release Velocity, System Defined Default Velocity, User Defined value for	Iteration	• Name • Theme • Start Date • End Date • Resources • Plan Estimate • Task Estimate • To Do • State	Planning, Committed, Accepted

		• Release Manager • Development Manager • Unit Tests Results URL	reference velocity			
	Iteration Planning	• Id • Name • Start Date • End Date • Work Capacity in Ideal days • Estimation of selected tasks in ideal days • Selected work vs Capacity ratio • Velocity • Development Resources		Planning	• Plan Iteration Status • Rank • Id • Name • Estimate • State	Resources, Estimate, Available

Appendix D: Student Project Artefacts (Examples)

Figure D1: Hartmann-Orona Spreadsheets - Task Commits

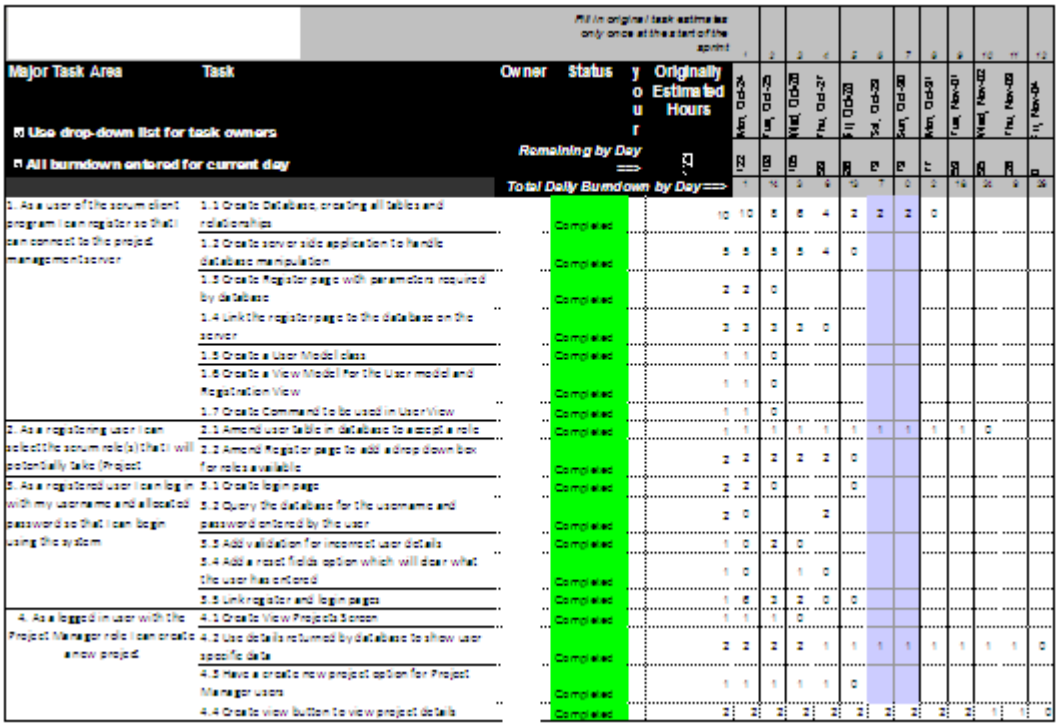


Figure D2: Hartmann-Orona Spreadsheets - Team Planning

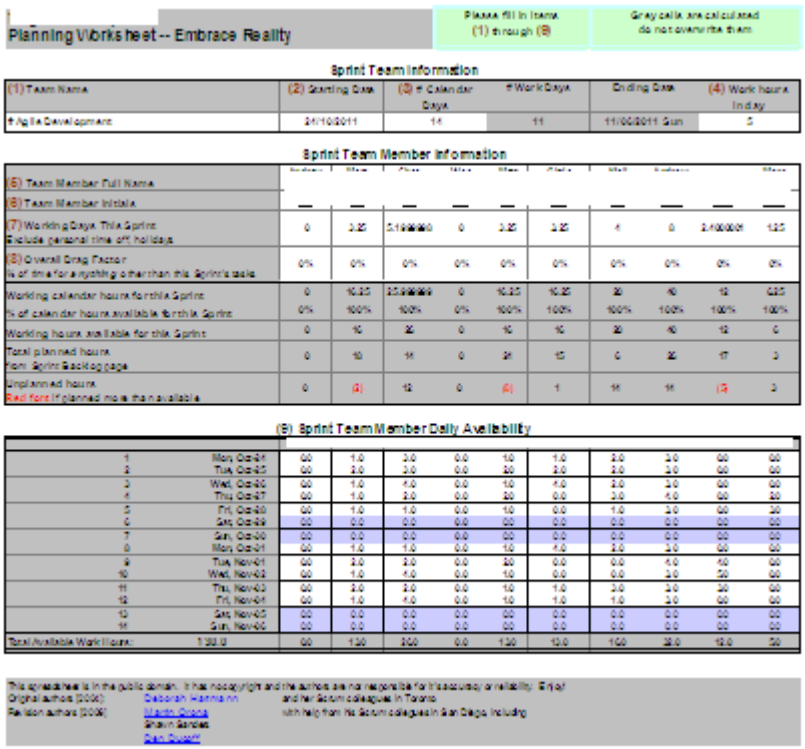
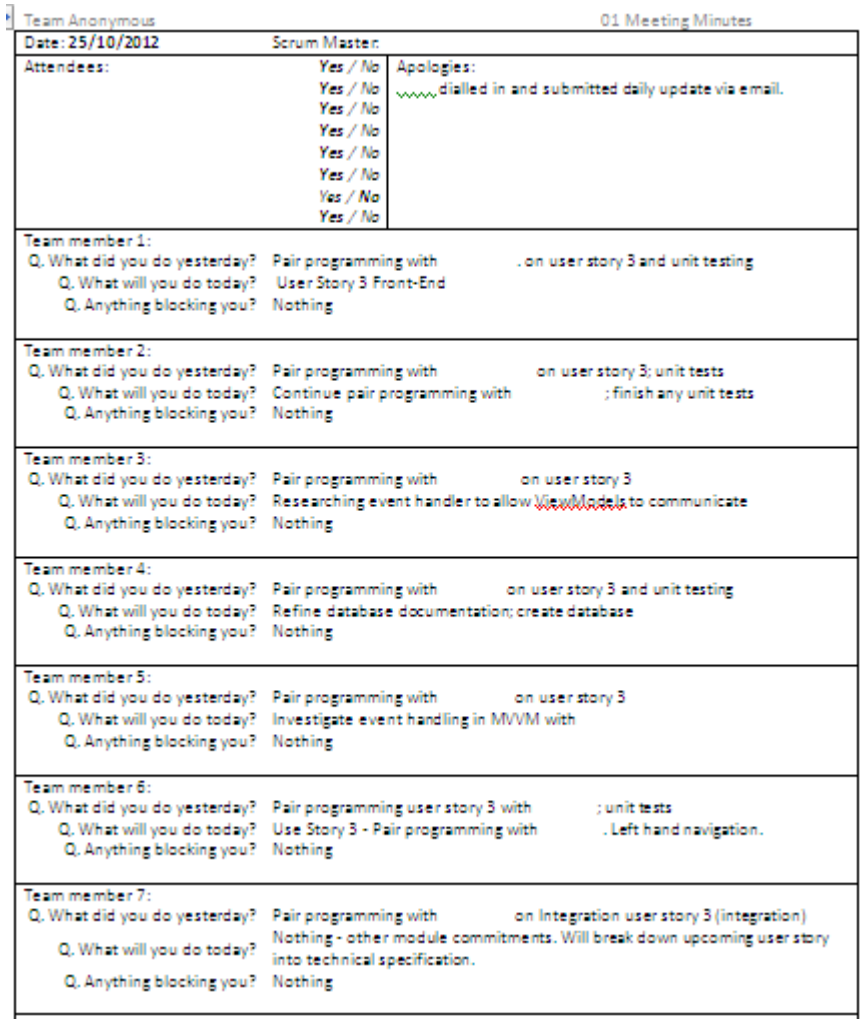


Figure D3: Sprint Backlog Target

Sprint Backlog Target

		Points	Dependencies
1	User Stories	2	-
2	As a user of the scrum client program I can register so that I can connect to the project management server	0.5	1
3	As a registering user I can select the scrum role(s) that I will potentially take (Project Manager, Product Owner, Scrum Master, Developer) so that I will be able to carry out those roles in projects	3	1
4	As a newly registered user I will receive an email with a newly generated password so that I can log in to the system	0.5	1,3
5	As a registered user I can log in with my username and allocated password so that I can begin using the system	5	2,4
6	As a logged in user with the Project Manager role I can create a new project	3	5
7	As a Project Manager I can search for a specific registered Product Owner and assign them as a Product Owner for a project so they can begin to add stories to the Product Backlog	3	5
8	As a Project Manager I can search for specific registered Scrum Masters who are not already assigned to an unfinished project and add them to the Project so that they can begin to organize Sprints	5	6,7
9	As a logged in user I can see a list of the projects to which I have been assigned and open one so that I can begin to	8	5
10	As a Product Owner I can Add new stories / Edit currently unassigned stories/ Reorder stories on the Product Backlog so that they reflect the current requirements, real priorities and likely ROI	2	8
11	As a Product Owner I can view all other details of the project but I am not able to edit anything	5	7
12	As a Project Manager I can create a new sprint and search for a Scrum Master who is available during the period of a	5	11
13	As a Scrum Master I can search for registered Developers who are available during the sprint dates so that I can add	40	12
14	As a scrum master I can use the system to conduct a planning poker session with my team (who may be remotely located) so that the estimates for stories on the product backlog can be decided and set prior to the sprint backlog	5	11
15	As a Scrum Master I can allocate user stories from the Product Backlog to the Sprint Backlog so that they can be implemented by the team and they are shown as assigned and uneditable on the product backlog until the sprint is	5	13
16	As a Scrum Master I can add tasks with estimated hours of work to user stories on the sprint backlog so that a team	8	15
17	As a Developer I can easily update the hours remaining for any tasks for which I am assigned so that a burn down chart created using the values would reflect reality	3	14
18	As a Developer I can add new tasks to stories on the sprint backlog so that they can be allocated a time and person	1	15
19	As a Developer I can indicate that I am blocked on a certain task so that the problem is highlighted for the Scrum	20	16,17,18
20	As a user I can save my open project to a local XML file which I can reopen later to view project details so that I can see the details offline (in stand-alone mode)	3	16
21	As a user I can see the team burn down chart for each sprint so that I can see the progress being made	2	20
22	As a user I can see the projected burndown line on the burn down chart	2	21
	As a user I can print the Burn Down chart so I can pin it up on the wall		
19.1	Breakdown of Task 19	20	19
19.2	As a Scrum Master I can create a planning poker session so user stories can be estimated even from a remote location	8	19.1
19.3	As a Scrum Master I can invite required people to the planning poker session so they can provide their estimates	5	19.2

Figure D4: Scrum Minutes of Meeting



Team Anonymous 01 Meeting Minutes

Date: 25/10/2012	Scrum Master:	
Attendees:	*Yes / No* *Yes / No* *Yes / No* *Yes / No* *Yes / No* *Yes / No* *Yes / No* *Yes / No*	Apologies: dialled in and submitted daily update via email.
Team member 1: Q. What did you do yesterday? Q. What will you do today? Q. Anything blocking you?		Pair programming with . on user story 3 and unit testing User Story 3 Front-End Nothing
Team member 2: Q. What did you do yesterday? Q. What will you do today? Q. Anything blocking you?		Pair programming with on user story 3; unit tests Continue pair programming with ; finish any unit tests Nothing
Team member 3: Q. What did you do yesterday? Q. What will you do today? Q. Anything blocking you?		Pair programming with on user story 3 Researching event handler to allow ViewModels to communicate Nothing
Team member 4: Q. What did you do yesterday? Q. What will you do today? Q. Anything blocking you?		Pair programming with on user story 3 and unit testing Refine database documentation; create database Nothing
Team member 5: Q. What did you do yesterday? Q. What will you do today? Q. Anything blocking you?		Pair programming with on user story 3 Investigate event handling in MVVM with Nothing
Team member 6: Q. What did you do yesterday? Q. What will you do today? Q. Anything blocking you?		Pair programming user story 3 with ; unit tests Use Story 3 - Pair programming with . Left hand navigation. Nothing
Team member 7: Q. What did you do yesterday? Q. What will you do today? Q. Anything blocking you?		Pair programming with on Integration user story 3 (integration) Nothing - other module commitments. Will break down upcoming user story into technical specification. Nothing

Appendix E: Investigation Notes

Table E1: Investigation Notes based on outcome from CSA

Investigation Notes (based on previous project investigated - CSA)	
No.	**Description**
1	User stories ID should use a unique ID that start with US1.. US2.. Usn
2	Task ID should use a unique ID that start with TS001… TS002..TSn
3	Team ID should use a unique ID that start with TM001..TM002..TMn
4	Missing information on ScrumMaster name - to make sure that this detail in included
5	Each US should be indicated its priority based on value and effort for each of them (for example: High/Medium/Low)
6	Breakdown of the US into no. of task - to make sure that each task has +/- the same size of effort or hours
7	Record the unit test and acceptance test based on each Task ID and file name commits in SVN
8	Record the list of bugs based on each Task ID and the file name commits in SVN
9	Spread all the task evenly to all team members - if US are broken down into too many tasks there are possiblity that team member will take too many tasks at one time
10	Consecutive task - one task can be started only when another task has completed OR depending on hours of availability to work e.g. 3 hrs availability can commit (Task1(1hr)+Task2(2hrs)
11	Each student should specify the skill level and agile experience at the start of the project - this is based on if they have involved with industrial placement etc
12	Emphasize student to do pair programming especially if the size of task is large
13	Due to inconsistencies of estimated hours for tasks in each group, the pair programming rule needs to be amended to detect risk for a bigger size of the task
14	During the standup meeting, each team member should be prepared to report the 3 items. Each reported work should use a unique ID e.g. TaskID for task and TeamID who responsible for the work.

Appendix F: New Minutes of Meeting format

Figure F1: New Minutes of Meeting Template

AGILE RISK PROJECT MINUTES TEMPLATE

Day:		**Absent in Daily Stand-up Meeting:**	**Progress Report? (Y/N)**	*Please use standard pattern of ID as below:	
Date:		1-		Task ID: TS001...n	
S.Mast		2-		Team ID: TM001...n	
		3-			
		4-			

What is done yesterday

Task ID	Description (detail on what was done)	Actual hrs spent	Hrs remaining	Task Status (In Progress / Completed)	Team ID	Paired By (ID)	File Name in SVN (Commit file)

What is the plan for today

Task ID	Description (detail on what is the plan for today)	Estimated hrs	New Task (Y/N)	Team ID	Paired By (ID)	File Name in SVN (Commit)

Impediments

Task ID	Description an impediment

Appendix G: Tables of Product Evaluation for CSA and CSB

Table G1: Deliverables for CSA

CSA - Product Deliverables for Each Team for Sprint 1 and 2						
Team	Sprint	Volume: Acceptance test and completed US (%)	Quality: Code style, System Design & Pattern, Unit Testing (%)	Project Management : Project files, Minutes meeting (%)	Average	Product Quality
Alp1	SP1	55	55	58	56%	Poor
	SP2	62	60	60	61%	Fair
Alp2	SP1	80	80	68	76%	Good
	SP2	87	80	87	85%	Very Good
Alp3	SP1	68	80	65	71%	Good
	SP2	80	80	75	78%	Good
Alp4	SP1	60	58	58	59%	Poor
	SP2	68	65	75	69%	Fair
Alp5	SP1	75	70	65	70%	Good
	SP2	70	67	75	71%	Good
Alp6	SP1	75	70	65	70%	Good
	SP2	75	75	65	72%	Good

Category:	80-89%	Very Good deliverable
	70-79%	Good deliverable
	60-69%	Fair deliverable
	50-59%	Poor deliverable

Table G2: Deliverables for CSB

CSB - Product Deliverables for Each Team for Sprint 1 and 2						
Team	Sprint	Volume: Acceptance test and completed US (%)	Quality: Code style, System Design & Pattern, Unit Testing (%)	Project Management: Project files, Minutes meeting (%)	Average	Product Quality
Bet1	SP1	58	62	62	61%	Fair
	SP2	58	65	65	63%	Fair
Bet2	SP1	58	78	55	64%	Fair
	SP2	72	85	60	72%	Good
Bet3	SP1	58	62	62	61%	Fair
	SP2	70	65	70	68%	Fair
Bet4	SP1	68	72	75	72%	Good
	SP2	78	78	78	78%	Fair
Bet5	SP1	58	65	65	63%	Fair
	SP2	62	65	68	65%	Fair
Bet6	SP1	62	58	62	61%	Fair
	SP2	62	68	68	66%	Fair
Bet7	SP1	78	82	72	77%	Good
	SP2	78	82	72	77%	Good

Category:	80-89%	Very Good deliverable
	70-79%	Good deliverable
	60-69%	Fair deliverable
	50-59%	Poor deliverable

Table G3: Pair programming rate for CSA and CSB

Team	Sprint	Pair programming rate (%)
Alp1	SP1	0
	SP2	5
Alp2	SP1	29
	SP2	11
Alp3	SP1	0
	SP2	0
Alp4	SP1	0
	SP2	17
Alp5	SP1	44
	SP2	85
Alp6	SP1	12
	SP2	15
Bet1	SP1	16
	SP2	4
Bet2	SP1	10
	SP2	4
Bet3	SP1	10
	SP2	9
Bet4	SP1	33
	SP2	6
Bet5	SP1	30
	SP2	1
Bet6	SP1	6
	SP2	0
Bet7	SP1	31
	SP2	20