Tool Support for Social Risk Mitigation in Agile Projects

Sherlock Anthony Licorish

A thesis
submitted in partial fulfilment
of the degree of
Master of Computer and Information Sciences (MCIS)

at the

Auckland University of Technology
Auckland

June, 2007

Primary Supervisor: Anne Philpott

Co-supervisor: Professor Stephen MacDonell

Table of Contents

List of Figures

List of Tables

Attestation of Authorship

"I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning."

………………………………………………….

Acknowledgements

The supervisors of this research have done all that is possible to make this a challenging but pleasant experience. A special thank you is hereby extended to Professor Stephen Macdonell and Anne Philpot for their exceptional guidance throughout this study. In addition to satisfying the research goals set out for this project, my research experience has been one of enjoyment, and my expectations were exceeded.

Thank you to the lecturers of AUT who instilled in me a drive for rigor and a spirit of critical inquisition; in particular, Dr. Andy Connor for his kind words, direction, and early interest in supervising me, and Krassie Petrova for being the most dedicated program administrator.

Special thanks to the Government of New Zealand, the New Zealand Agency for International Development (NZAID), the New Zealand Vice Chancellors' Committee (NZVCC), the Commonwealth Scholarship and Fellowship Plan (CSFP), and the Government of Guyana for offering me a special opportunity.

Finally, I would like to recognise my wife Ngosi Mondella Smith-Licorish, and son Isiah Jordon Saul, who left their comforts in Guyana to travel to New Zealand to support me whilst I realised a life-long dream; this milestone would not have been possible without their involvement and support.

Abstract

Software engineering techniques have been employed for many years to guide software product creation. In the last decade the appropriateness of many techniques has been questioned, given unacceptably high rates of software project failure. In light of this, there have emerged a new set of agile software development methodologies aimed at reducing software projects risks, on the basis that this will improve the likelihood of achieving software project success. Recent studies show that agile methods have been gaining increasing industry attention. However, while the practices recommended by agile methodologies are said to reduce risks, there exists little evidence to verify this position. In addition, it is posited that the very processes recommended by agile methodologies may themselves introduce other risks.

Consequently, this study addresses the risks inherent in the human collaboration practices that are central to agile methods. An analysis of the risk management literature reveals that personality conflicts and customer-developer disagreements are social risks that occur through human collaboration. These risks negatively affect team cohesion and software project success. Personality conflicts are said to be mostly influenced through poor team formation, whereas customer-developer disagreements are induced through excessive customer direct interaction. However, these risks are not adequately addressed by standard risk management theories. Furthermore, an evaluation reveals that these risks are also not considered by existing software tools.

This study therefore designs and implements a web-based solution to lessen the social risks that may arise in agile projects. The Agile Social-Risk Mitigation Tool (ASRMT) offers support for personnel capability assessment and management and for remote customer feature management, extending the customer's access through an interface. Using software engineering experts to evaluate ASRMT, the tool is shown to effectively address social risk management theories, and is considered likely to assist agile developers in their handling of social risks. In addition, above and beyond its intended purpose, ASRMT is also likely to assist agile teams with general project management. The findings of the ASRMT user evaluations demonstrate sufficient proof of concept to suggest that such a tool could have value in live software projects.

1. Introduction

As the scale and complexity of software systems have grown over time, the processes used in the creation of software have tended to become similarly complex. Out of this complexity emerged a range of software development methodologies, each created to guide the software development process and/or its project management. Early methodologies (started in the 1950's and remaining popular up until the late 1990's) included the waterfall model, prototype model, iterative model, rapid application development model, and the spiral model. Nerur, Mahapatra, & Mangalaraj (2005) state that such conventional software models, which are also called heavyweight methodologies, follow a linear, heavily documented, pre-planned, process-centric, and very rule based process.

Project management has similarly employed conventional sequential models in the software project life cycle. Abrahamsson & Koskela (2004) assert that conventionally, software engineers try to conform to project plans. In spite of this, software project success rates using conventional software development methodologies over the last 20 years have not been promising. The well known 2001 report "Extreme Chaos" by the Standish Group reveals that more than 50% of all software projects either fail or overrun. In addition, the authors found correlation between underestimated project complexity and ignored changing requirements, and software failure. While there may be some questions over the scale of the problems reported by the Standish Group (Jorgensen & Molokken-Ostvold, 2006), there is little doubt that software development remains a very challenging activity.

In light of software project failures, a more recently promoted approach to software development called 'agile software development' emerged. In agile methodologies such as Extreme Programming (XP), Scrum, the Crystal Families of Methodologies, and Feature-Driven Development (FDD), there is a gradual surfacing of the software design and requirements, which promotes a more humanistic environment, having persons interacting in a common space, employing a 'speculate-collaborate-learn' approach (Highsmith, 2000). Beznosov & Kruchten (2004) contend that the aims of this new approach are to reduce failure in software projects, and reduce the cost of software development.

Abrahamsson, Warsta, Siponen, & Ronkainen (2003) and Kuppuswami, Vivekanandan, Ramaswamy, & Rodrigues (2003) claim that agile methodologies can reduce failure in software projects by mitigating software project risks. However, these authors do not present any empirical evidence to support their claim. Their view is in contrast to Kirk & Tempero (2006) and Sharp, Robinson, & Segal (2004), who argue that agile methodologies may present complexities which may result in additional risks in the software process. Added risks may be associated with minimal upfront planning which can sometimes result in rework due to oversight, regular customer involvement which may influence disagreements and increase project cost, the need to manage a diversity of skills within highly interactive project teams, a lack of shared vision and domain knowledge, and a lack of documentation which results in poor communication during the project life cycle (Kirk & Tempero, 2006; Sharp et al., 2004; Nord & Tomayko, 2006; Hulkko & Abrahamsson, 2005).

In implementing agile methodologies, developers therefore face a set of circumstances in which risk continues to be evident; perhaps new risks are introduced by the very use of the agile methodologies that have been promoted to reduce (conventional) risk. In light of the fact that there is steady adoption of agile methodologies (Behrens, 2006; VersionOne, 2006b), but little empirical evidence to verify the effectiveness of these methodologies in (new) risk analysis and mitigation, there emerges a research opportunity. This research project therefore addresses theories in support of risk analysis and mitigation in agile contexts, leading to the development of a tool that is directed towards mitigating risks during agile software development projects.

The next section of this chapter highlights the intended contributions and outlines the research objectives of this project; this is followed by a discussion of the selected research design, and the structure of this thesis.

1.1. Intended Contributions and Research Objectives

Research examining agile methodologies and the practices recommended thereby seems to be mostly descriptive (Abrahamsson et al., 2003). While several authors have made recommendations regarding ways for improving agile practices (for example Augustine, Payne, Sencindiver, & Woodcock (2005), Kontio, Hoglund, Ryden, & Abrahamsson (2004), and Williams & Cockburn (2003)), it appears that such recommendations have been founded on the basis of little empirical evidence. As stated above, agile methodologies have gained significant industry attention, and research shows that these methodologies are being increasingly adopted (Behrens, 2006). Therefore, studies aimed at evaluating agile methodologies, and offering tangible support (or otherwise) for process improvement in agile projects, have the potential to provide both theoretical and practical value. This study seeks to offer a concrete, evidence-based way for improving agile practices by evaluating risk management in agile methodologies, and offering a toolset to assist with social risk administration.

Correspondingly, the objectives of this research project are ‘to evaluate risks in agile software development methodologies, and to provide a tool to assist with risk management in software projects adopting agile methodologies’. In order to realise the objectives of this research project, this study aims to answer the following research questions:

1. What risks are induced through human collaboration?
2. Can a process tool be implemented that effectively addresses social risk management theories?
3. Will such a tool be useful to agile software teams in terms of improving project risk management?

1.2. Research Design

Embedded in this research is a set of expectations regarding best practice software development. Existing methods and tools have been and will continue to be evaluated against best practice expectations regarding risk management. This work provides the basis for the generation of a set of criteria against which to assess agile project management tools, and that guides the development of a new tool. This research is therefore conducted using the constructive method, in the positivist paradigm. Atsuta & Matsuura (2004), Ceravolo, Damiani, Marchesi, Pinna, & Zavatarelli (2003), Kaariainen, Koskela, Abrahamsson, & Takalo (2004), Maurer & Martel (2002), and Rees (2002) have all employed this approach in their similar studies. This method is aimed at building innovative prototype systems, to present proof of concepts for improving effectiveness and efficiency in organisations.

The constructive paradigm is alternatively referred to as design science (Hevner, March, Park, & Ram, 2004). The design science paradigm is in fact made up of two disciplines: behavioral science and design science. Silver, Markus, & Beath (1995) assert that in the design science paradigm, building applications to solve problems uncovers knowledge and understanding. Zmud (1997) also argues that the constructive method is most suited toward the construction of IT solutions. It is therefore appropriate for the research proposed here.

According to Highsmith (2004), risk assessment by project managers in agile software projects is predominantly qualitative in nature. This is because the project manager must always be reflecting on previous experiences, constantly assessing the relationship between engineers and customers, and its impact on the software project. Thus, any effective project management tool should address this and other aspects of the development context. Kelter, Monecke, & Schild (2003) affirm that agile project management tools may reduce risk in agile projects by extending the scope of agile methodologies. These tools are created through the design science paradigm. Denning (1997) asserts that the only way to fulfill these requirements in design science is to apply theories that build, test, and modify artifacts through the experience and creativity of researchers. Thus, the principles appropriate for agile risk management, and the theories governing the development of effective project management tools are considered during this study.

Glass (1999) stresses that the relevance of IS research is directly linked to its application. In addition, he argues that the outcomes of IS research should be implementable. He further contends that the rigor involved in the development of artifacts promotes critical thinking; sometimes existing theories are insufficient; thus, new theories are made. Markus, Majchrzak, & Gasser (2002) emphasise that innovative artifacts extend the boundaries of human problem solving capability through the provision of intellectual and computational tools, and these artifacts also add to existing theories once they are implemented. These insights direct the development of the tool that is created in this research.

Therefore, in accordance with the principles of the constructive method, relevant risk theories underpin the conceptualisation of a toolset to support those undertaking agile software development projects. The toolset is evaluated against criteria selected and/or developed during the literature review and from an evaluation of existing tools. This provides the primary basis for the assessment of the outcomes of the research. The views of a small number of experts are also informally sought to provide additional verification of the toolset's potential effectiveness in use.

1.3. Thesis Structure

The remaining sections of this research are structured as follows; Chapter 2 examines the evolution of software development practice in order to provide the appropriate context for the work subsequently performed. Chapter 3 presents an introduction to software project risk management along with an analysis of existing risk management approaches, and highlights inefficiencies in these approaches. In Chapter 4 a study of the effect of one of agile methods’ most favoured processes: human collaboration - the effect of stakeholders’ interaction is presented. In Chapter 5 consideration is given to existing project management tools in order to verify what the different agile tools are, and how these tools are used to support agile teams. In Chapter 6 an introduction to the design and implementation of a tool called the agile social-risk mitigation tool (ASRMT) is provided. Chapter 7 explains the ASRMT evaluation process, discusses the benefits of using ASRMT, and revisits the research questions and highlights this study’s contribution. Finally, Chapter 8 summarises this project, presents the conclusions for this research, considers the limitations of this study, and provides recommendations for future research.

2. Evolution of Software Development Practice

Software engineering is seen as the process by which software developers apply a systematic and controlled approach to software development. Boehm (2006), Coram & Bohner (2005), and Kelter et al. (2003) contend that, through software engineering, useful software products are manufactured for customers. According to Coram & Bohner (2005), software development, as a discipline, is said to confront two vital challenges that distinguish this discipline from the realm of other engineering disciplines. The product, software, which can be seen as conceptual and intangible, is adopted in evolving and changing environments, much in contrast to the development of tangible products, such as integrated circuits. In addition, even though change is often allowed, the cost of change in software development is said to increase exponentially as the project progresses (Kim, 2006).

While the previous assertions might be generally true, in contrast to Kim (2006), Beck (2000) claims that in the production of a tangible software artifact, software processes that possess the ability to respond rapidly to change allow early identification of defects which are likely to reduce cost throughout the project (reducing the cost of change as the project progresses). The varying opinions, and the risk involved in software development have led to the emergence of a new set of software development methodologies (Chin, 2004). Augustine et al. (2005) argue that the new methodologies, Agile Software Development Methodologies, may be substituted for process centered software development methodologies in order to avoid the risks inherent in existing approaches. In contrast, Coram & Bohner (2005) and VanDeursen (2001) express reservations regarding this view, and assert that agile methodologies might introduce several complexities leading to software risks.

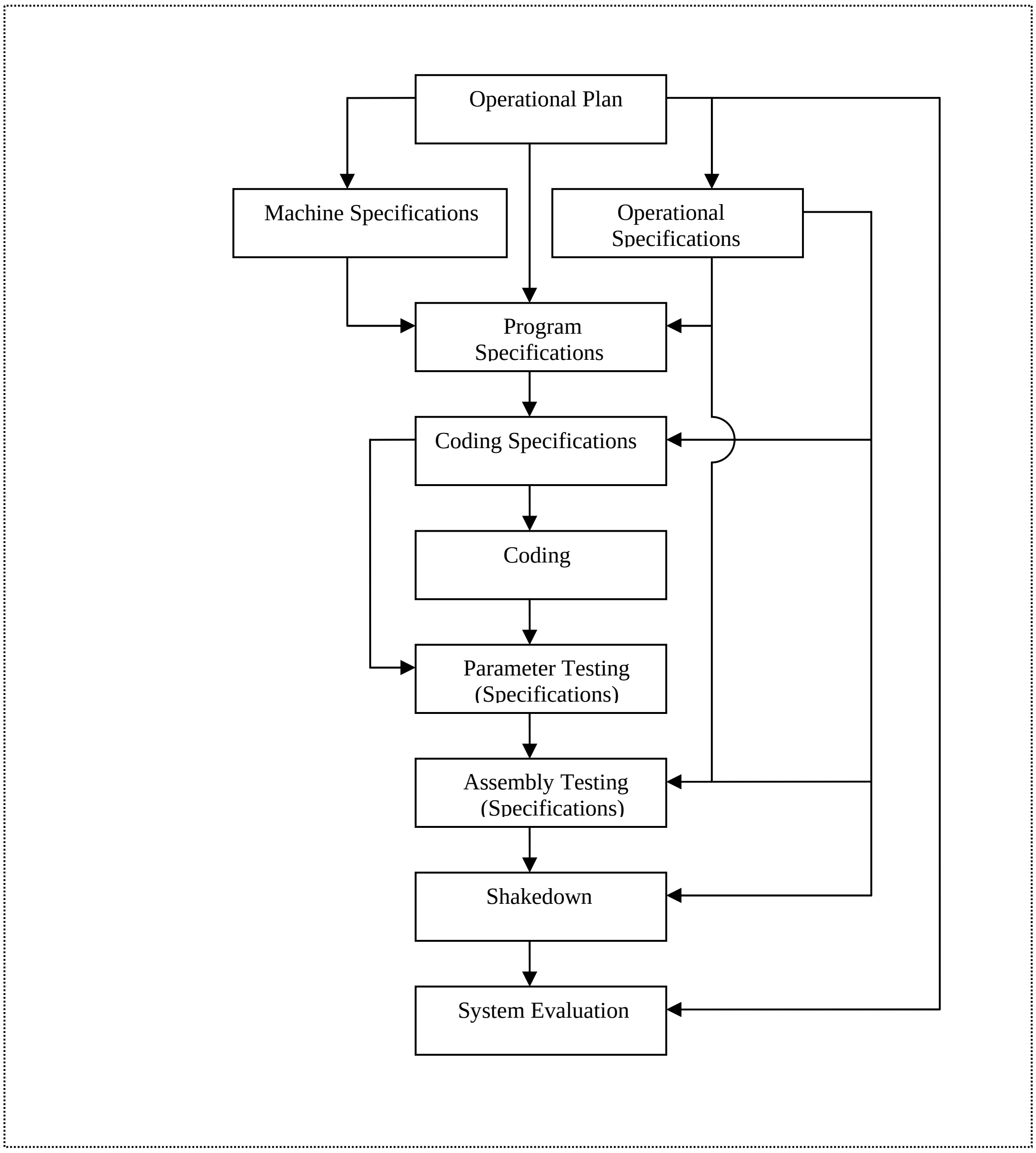
To understand the methodologies used for software engineering and place the proposed work on risk management in the wider context, this chapter presents a brief overview of the history of software engineering and its evolution. This chapter also examines the conventional software development methodologies and agile software development methodologies, and compares and contrasts these methodologies.

2.1. History of Software Development

To understand where software engineering is today, it is essential to verify where this discipline started and the reasons for its formation and transformation. Therefore, in this section the history and evolution of software development is examined.

Early use of the term software engineering emerged in the 1950's. During this era, software engineering was closely linked to hardware engineers and mathematicians (Boehm, 2006). In addition, the software developed during this period often supported aircraft or rocket engineering. Evidence of this is reflected in the Semi-Automated Ground Environment (SAGE) project, which was developed for air defense boundaries of United States of America and Canada. The software development process followed in this project (see Figure 2.1) (Boehm, 2006), even though waterfall in nature, was quite different to the processes that were formally recommended during the 1960's. In Figure 2.1 the software processes were particularly aimed at supporting hardware functionality, and this model (overall) bears a resemblance to those used in the production of hardware. In addition, there is less emphasis on the software requirements and design phases (leading to coding) when compared to the subsequent model (see Figure 2.2).

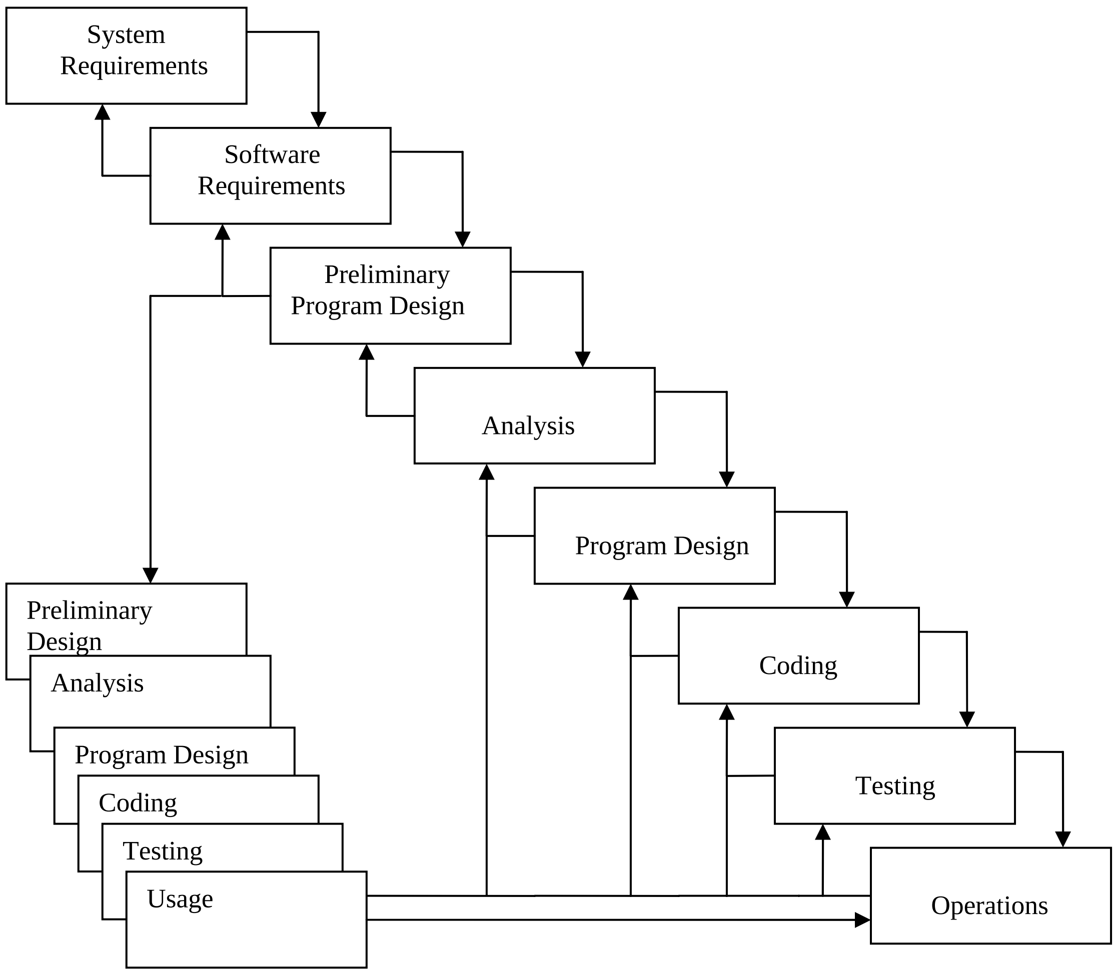
Figure 2.1. The SAGE Software Development Process (1956) (Boehm, 2006)



During the 1960's, efforts made by Peter Naur and Brian Randell to formalise the software engineering process at the first ever software engineering conference were greeted with mixed reaction (Boehm, 2006; Cockburn, 2004). This was because the term engineering and its practices were more akin to the hardware perception of engineering than to software development. In addition, the software engineering process possessed many more states, modes, and paths to test than the former, often complicating specifications (Naur & Randell, 1968). However, with subsequent work by Brian Randell and John Buxton, Software Engineering as a branch of engineering became partially accepted (Randell & Buxton, 1969). This initial reluctance by some may have re-emerged in recent years in the form of growing dissatisfaction with heavily prescriptive methods.

During the 1970's software development projects followed an amended SAGE waterfall model (Boehm, 2006). The SAGE model was revised by Royce; this new model possessed several additional stages when compared to the preceding model (see Figure 2.2.). Royce recommended a more carefully organised coding phase, which was preceded by the design, which was in turn preceded by the requirements engineering phase (Boehm, 2006). These activities were most often sequential, as design did not start until there was an exhaustive set of requirements, and coding was delayed until there was extensive design review. This software process was reinforced by the US Government, and at this time they began to instantiate software development standards (Boehm, 1976, 2006).

Figure 2.2. The Royce Waterfall Model (1970) (Boehm, 2006)



Though IBM and the US military reported that small software projects were being developed in a stable environment using Royce and SAGE models in the 1970's (Boehm, 1976), the early 1980's saw a reverse of these assertions. The US Department of Defense (DoD) reported that software projects were over budget and often late, lacking the required functionality (Boehm, 2006). This led to the implementation of the DoD's DoD-STD-2167 and MIL-STD-1521B contractual standards in 1985. These standards were recommended for the development of future software projects. The DoD-STD-2167 and MIL-STD-1521B standards strongly reinforced previous waterfall models, and stressed thorough management reviews in every project milestone. However, even with the new standards, software projects still continued to fail (Standish Group, 1995).

Due to these ongoing failures, the DoD commissioned the Software Engineering Institute (SEI) formed in 1984, to develop a software capability maturity model (SW-CMM) to assess organisations' software process maturity (Boehm, 2006). The SW-CMM model was based on IBM's software practices, and was partially effective at assessing organisations' capability (Humphrey, 1989), though it was still reinforcing the previous waterfall models. In addition, around this time the International Standard Organisation (ISO) standard (ISO-9001) was also developed under European leadership to bring forward complementary standards (Boehm, 2006).

Subsequent years for software development were relatively successful (Standish Group, 1995). Software contractors who complied with the standards (SW-CMM and ISO-9001) in order to secure contracts reported some success regarding a reduction in software rework in the early 1990's. In addition, research into the use of tools in the software development process, and improvement to the waterfall software process was reported (Ryder, Soffa, & Burnett, 2005). Further, Brooks (1987) presented several fervent ideas in his famous paper 'No Silver Bullet', which stressed that new software development technology such as process tools, code reuse, high level languages, and powerful workstations might reduce the cost and time to develop software.

Around the time of Brooks's suggestions, OOP, and software architecture and description languages where beginning to support the expansion of the World Wide Web (WWW) (Gamma, Helm, Johnson, & Vlissides, 1995; Perry & Wolf, 1992). The emphasis on software development had now shifted with pressures from time-to-market and open source development, and the waterfall models were not suited towards concurrent software engineering (Boehm, 2006). Thus, the view of software as a competitive discriminator and the reduction in the time-to-market gave rise to the agile methodologies in the late 1990's. Agile methodologies stress a software development environment in which requirements emerge over time, emphasising adherence to changing priorities and the use of different processes and tools.

Since the emergence of agile software development methodologies, there has been speculation on their supremacy over conventional software development methodologies (Augustine et al., 2005). However, there seems to be little empirical evidence to confirm this assumption. The following sections (sections 2.2 and 2.3) of this chapter are dedicated to the examination of conventional software development methodologies and the newer agile software development methodologies, with a view to considering their impact on software development risk management.

2.2. Conventional Software Development Methodologies

Due to a desire for formality and an emphasis on control, from their inception conventional software development methodologies emphasised planning, and processes were most often predefined, utilising the rule of division of labor (Boehm, 1976, 2006). Boehm (2006) notes that plans and processes were often used to agree on system architecture, project time lines, and phases in the software development life cycle well in advance of the start of the software project. In addition, fulfillment of customers' requirements, which were agreed upon prior to system development, determined the projects' success. This train of thought is also evident in comments of Kelter et al. (2003); they claim that in conventional software development environments, changes and volatility in the software development process were often greeted with additional plans and processes. Consequently, conventional software methodologies were often called process centric, plan-oriented, or (rather negatively) heavyweight.

Due to the nature of the processes recommended by conventional software development methodologies, there has been controversy over the early adoption of such methodologies and their effectiveness (Beznosov & Kruchten, 2004; Chin, 2004; Kelter et al., 2003). Chin (2004) argues that conventional software development methodologies were constructed to manage large US government projects for the military, construction companies, and space industries; as such they are not suitable for software development. A similar line of thought was put forward by Beznosov & Kruchten (2004), suggesting that software development that conforms to previous methodologies that were developed for construction or manufacturing may be subject to failure, because unlike construction, software development is not a linear activity.

While the literature presented in this chapter shows that the arguments of Beznosov & Kruchten (2004) and Chin (2004) have some degree of support, software development, regardless of the scale or method used, is a complex phenomenon. Hence, simply replacing one software methodology with another will not in itself eliminate software risks (Nerur et al., 2005). The well-known reports "Chaos" and "Extreme Chaos" published in 1995 and 2001 by The Standish Group reveal that only about 16% of the sample studied achieved software project success (Standish Group, 1995, 2001). These reports show that about 31 % of those projects sampled failed, while 53% were over-cost and exceeded schedule (Standish Group, 2001). Further, these authors found correlation between "underestimated project complexity" and "ignored changing requirements", and "software failure". Although there may be some questions over the scale of the problems reported by the Standish Group (Jorgensen & Molokken-Ostvold, 2006), there is little doubt that software development remains a very challenging and risky activity.

2.2.1. Representative Conventional Methodologies

In software development, the term waterfall categorically defines all software development methodologies whose phases during the development process are sequential in nature. Projects using the original waterfall model proceed sequentially through phases such as requirements analysis, design, coding, testing, release, and support. Each of these phases requires contractual sign-off prior to the commencement of another phase (Pressman, 2001). This contractual control or documentation is said to form the basis for predicting, measuring, and controlling problems (risks) throughout the software development life cycle (Highsmith & Cockburn, 2001). In addition, there are specific individual(s) assigned to software roles who are responsible for each phase during the software development life cycle (Nerur et al., 2005). Further, individual roles use documentation as a guide for their development and communication efforts.

Even though the waterfall model is popular, Siddiqui & Hussain (2006) establish that present-day software engineers are not likely to implement this model in their software development projects. This is because of the overhead required in implementing the processes recommended by this model. In addition, the waterfall model's presumption that requirements should be stable at a certain (early) point in time encourages several variations to this model (Highsmith & Cockburn, 2001). In the following discussions, three of the most popular variations of the waterfall model are explained.

The rapid-prototyping model was developed to recognise the dynamic nature of software engineering. According to Siddiqui & Hussain (2006), in this model, developers aim to deliver an early prototype of the accepted customer requirements. This prototype is expected to be a simple version on the proposed system, and is aimed at gathering feedback from the parties involved in the software project in order to produce an enhanced version of this prototype (Connell & Shafer, 1989). If the enhanced version does not meet the customer requirements, this prototyping process is repeated until it is accepted or until an agreed end point is accomplished.

Likewise, the spiral model implements a process of prototyping in a framework that combines the rapid-prototyping model and the waterfall model (Siddiqui & Hussain, 2006). In addition to the delivery of a prototype which is continually refined in the rapid-prototyping model, the spiral model emphasises risk assessment which is assisted through repeatedly implementing the waterfall model in the software development life cycle (SDLC). Oriogun (1999) explains that the purpose of risk assessment in the spiral model is to verify the feasibility of continued development. The prototype developed in both the rapid-prototyping and spiral models are used as guides for the original system after the parties involved in the software project agree that the prototype resembles what is required.

A third variation to the waterfall model is the evolutionary prototyping model. Sommerville (1997) says that in this model prototypes are similarly developed as in the preceding models. However, the prototype itself evolves into the final system after acceptance that the prototype adequately implements the initial customer requirements. Divergent to the preceding models, the final version of the prototype in the evolutionary prototyping model is completed using waterfall techniques and delivered as the final product to the customer.

2.2.2. Characteristics of Conventional Methodologies

Previous research seems to associate conventional software development methodologies with pre-planning, formal processes, and documentation (Connell & Shafer, 1989; Oriogun, 1999; Siddiqui & Hussain, 2006). Though there are several variations to the initial waterfall methodology, these deviations retain emphasis on a plan driven and process oriented approach. Hence, Kelter et al. (2003), Nerur et al. (2005), and Oriogun (1999) identify the following characteristics which are consistent with the conventional software development methodologies:

Plan Oriented

The goal of conventional software development methodologies is to capture the requirements of the software project prior to the SDLC, through requirements engineering and analysis. In the planning phase, contractual estimates specify the cost of the project, project requirements, and project time. Changes in the requirements are often characterised by re-planning which also influences additional contractual estimates and documentation.

All-inclusive Documentation

The role of documentation in conventional software development methodologies is to provide a representative snap shot of the project at any given time during the software development life cycle. In addition, the documentation forms the basis for future maintenance of the system.

Predictive

The predictive approach in conventional methodologies is as a direct consequence of the planning framework implemented by such methodologies. The plans assume software to be predictive and repeatable, similar to other engineering disciplines.

Task Oriented

The emphasis of conventional software development methodologies on role-base development activity (example: the systems analyst must analyse requirements) encourages a process oriented framework. In this setting, participants are seen as roles which can be exchanged arbitrarily. Therefore, the project manager may decide on a task which is communicated to any programmer via documentation. These processes have been the subject of criticisms since their inception (Boehm, 2006; Brooks, 1987; Tolvanen, 1998), the following section highlights a few of the criticisms commonly echoed in the literature that have led to the emergence of agile alternatives.

2.2.3. Criticisms of Conventional Methodologies

Issues associated with changing requirements, difficulties matching developers to roles, and too little focus on individual interaction seem to be the most common criticisms in the literature against the conventional waterfall model (see Kelter et al. (2003) and Nerur et al. (2005) for example). However, even when there are variations to this model (see section 2.2.1), there seem to be additional shortfalls to conventional methodologies.

The prototype developed in the rapid prototyping model is used to develop the real system. Gordon & Bieman (1993) assert that the provision of such a model leads customers to false expectations, believing that the prototype being developed is the system. In addition, Serich (2005) and Siddiqui & Hussain (2006) argue that the rapid prototyping and evolutionary prototyping models may induce risks associated with bad design, and that scalability can be a challenge. They explain that the technologies used in the prototype may work inadequately in the real system, which can also influence the cost of rework.

Though the spiral model is said to adequately deal with risk (Oriogun, 1999), Boehm (1988) affirms that adoption of this model may also result in risks associated with project over-runs. This is influenced by the demand of implementing the waterfall model in each prototype, and the extensive reliance on risk assessment expertise. Additionally, if the spiral model is used in a low risk software project, the overhead involved in implementing this model would be wasteful.

2.3. Agile Software Development Methodologies

The concept of agility in software development has been in existence for over two decades (Koch, 2005). However, in February, 2001, a group of 17 authors and developers met at a conference in the United States of America to officially forge an alliance (agile alliance) and formally agreed to the Agile Manifesto (http://www.AgileAlliance.org). The agile manifesto emphasised four values

(Coram & Bohner, 2005): individual and interaction over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan. In addition, members of the alliance sanctioned 12 principles (see Koch (2005) for details). The values and principles recommended by the agile alliance were said to deliver value to software organisations and customers, and improve software quality and reduce risk.

In agile software development methodologies such as Extreme Programming (XP), Scrum, the Crystal Families of Methodologies, Feature-Driven Development, Adaptive Software Development, and Agile Modeling, there is a gradual surfacing of the software design and requirements, which promotes a more humanistic environment, having persons interacting in a common space, employing a 'speculate-collaborate-learn' approach (Abrahamsson et al., 2003). This approach to software development is often termed 'lightweight'. The agile processes are implemented in contrast to the previous waterfall life cycle, which is often documentation driven, in a continuous 'plan-build-implement framework' (Siddiqui & Hussain, 2006).

There were several drivers behind the formation of the agile alliance, among these the rapid pace of change in the software industry, the pressure of time to market, users uncertainties when specifying requirements, and the requirements of commercial off the shelf (COTS) software were emphasised (Boehm, 2006). Additionally, conventional software development methodologies and predictable process management techniques were criticised as not being suited for modern software development (Chin, 2004). The fact that software projects success rate using conventional methodologies were not very satisfying (see (Standish Group, 1995, 2001)), was taken as clear evidence of their unsuitability.

All of these drivers highlighted a common variable, 'changing requirements'. Thus, agile software development methodologies were expected to deliver two sets of benefits, firstly, to meet the needs of the software industry, and secondly, to reduce software risk thereby improving software success rates. However, though there have been some attempts to verify the effectiveness of agile methodologies (Kuppuswami et al., 2003; Law & Charron, 2005; Misic, 2006), there exists little empirical evidence to demonstrate their success.

2.3.1. Representative Agile Methodologies

According to Collins dictionary and thesaurus (2006), agility denotes the quality of being “ready for motion, or being nimble”. Highsmith & Cockburn (2001) say that the focus in agile software development methodologies is on simplicity and speed. This, they maintain, is achieved through agile software development groups concentrating on specific requirements, and rapidly delivering solutions to those requirements. Thus, agile methodologies are said to adhere to the definition of agility by recommending a software process that is incremental, cooperative, straightforward, and adaptive (Abrahamsson et al., 2003). This is achieved through an emphasis on minimum documentation, regular customer involvement, iterative development, small collaborative teams, small releases, and continuous software product testing. To illustrate these principles, three of the most popularly reported agile software development methodologies are examined below.

Extreme Programming (XP), one of the agile software development models, brings together some new software principles to develop software based on ‘user stories’ in a vague and constantly changing environment (Kuppuswami et al., 2003). In XP projects, the software development life cycle is characterised by small iterations. Iterations each address a collection of stories, for each story an acceptance test is written, then a solution is designed and coded (Beck, 2000). Abrahamsson et al. (2003) advise that XP centres on small teams realising four values (communication, simplicity, feedback, and courage) and implementing twelve practices. These practices are: the planning game, small releases, metaphor, simple design, test first, re-factoring, pair programming, collective ownership, continuous integration, 40 hour week, on-site customer, and coding standards; see Abrahamsson et al. (2003) and Theunissen, Kourie, & Watson (2003) for further details.

In Scrum, development assumes and addresses changing environmental and technical variables (Schwaber & Beedle, 2002). Its main focus is on the organisation of a software team to influence software projects' success in a changing environment. Coram & Bohner (2005) say that Scrum's software project life cycle mirrors a rugby game, where there are three phases: Pre-game, Development, and Post-game. The pre-game is characterised by planning; early plans produce prioritised requirements, the system architecture, and a high-level design. Development follows in iterative cycles called 'sprints' (Schwaber & Beedle, 2002). In each sprint a part of the system is expected to be delivered. The post-game follows once there is full customer-team agreement on the system's functionality. In this phase, no additional functionality is presented (Coram & Bohner, 2005; Koch, 2005). Scrum projects involve regular project management activities aimed at removing obstacles which can influence deficiencies.

Similarly, Feature-Driven Development (FDD) emphasises software process management in the early stages of development, focusing its agility primarily on the design and building phases (Palmer & Felsing, 2002). Abrahamsson et al. (2003) affirm that the main focus in FDD software development is on quality and rapid delivery. This is rarely achieved without careful monitoring of project processes (Boehm & Turner, 2005). Koch (2005) and Palmer & Felsing (2002) explain that the FDD project life cycle has five phases: overall development planning phase, building features list phase, features planning phase, design by feature phase, and build by feature phase. Planning and process management are particularly evident in the first three phases, agile development then follows in the remaining phases, implementing a life cycle resembling that of XP.

2.3.2. Characteristics of Agile Methodologies

Proponents of agile methodologies agree that the term 'agile' denotes a way of thinking (Highsmith, 2004; Koch, 2005). Thus, though the preceding discussion categorises several software development methodologies as 'agile', the implementation of agile methodologies in real software projects may not always fit the agile framework. An example of this occurrence is demonstrated in Palmer & Felsing (2002), who posit that their first real use of the FDD method was in the process centered and task driven implementation of a critical banking system.

However, they explain that their implementation *environment* was agile. There may be deviations to this view; see (Abrahamsson et al., 2003) for example. Nonetheless, Verner & Evanco (2005) list the factors of agility as follows:

People Oriented

Agile software development methodologies stress the need for face-to-face communication among stakeholders (customers, developers, and end users). As a consequence of this phenomenon, documentation is accentuated only when it offers direct support to the stakeholders. Such methodologies favor daily communication among team members; for example: XP promotes the presence of an on-site customer during development.

Adaptive

Supporters of agile methodologies give emphasis to change in the software project. Agile methodologies welcome change, as it is often argued that change introduces new information that may improve software quality, and project success. The intent here is that a customer can request changes to the requirements without re-negotiating previous contracts.

Decentralised Planning and Decision Making

Decisions are made by any team member involved with agile projects. Chin (2004) asserts that the project manager’s role becomes that of a shaper and motivator in such projects (remover of obstacles). Accordingly, management is often responsible for general decision making and project coordination, and sometimes the project manager may even function in a developer capacity.

Small Teams

It is often posited that small teams allow for effective communication among agile team members (Cockburn, 2004; Nerur et al., 2005). This is because, as software teams become larger, face-to-face communication becomes restricted, and documentation becomes the main facilitator. Agile methodologies are said to reduce reliance on documentation, therefore agile methodologies may not be suited for implementing large software projects which require large teams.

2.3.3. Criticisms of Agile Methodologies

Augustine et al. (2005) explain that changing requirements are characteristic of all real life software projects, and agile software development methodologies are change-focused. If their assertion is true, then it might be said that agile methodologies will handle change effectively. However, the techniques employed in agile software development methodologies to address changing requirements, and to allow for a more participative software environment to reduce risks, are themselves risky (Agarwal, Prasad, Tanniru, & Lynch, 2000; Boehm & Turner, 2003). Similar sentiments are expressed by several other authors.

Chin (2004) asserts that agile software development methodologies, especially XP, rely on practices that require experienced and versatile team players. Thus, any deviation from this, which is inevitable, introduces risks related to software quality and delivery. In addition, documentation of the software development process in agile approaches is often replaced by collaboration among team players. However, De Souza, Anquetil, & De Oliveira (2005) and Nord & Tomayko (2006) state that in large projects documentation becomes a necessary factor for project success, and that in such scenarios human interaction cannot replace documentation. Similarly, Boehm & Turner (2003) say that the loss of team players in agile teams may result in the loss of tacit knowledge, which introduces further project risks.

McKinney & Denton (2005) assert that agile software development methodologies' practice of team collaboration might not always positively influence team cohesion, and therefore success, in agile projects. Likewise, Hulkko & Abrahamsson (2005) assert that agile methodologies emphasis on iterative feature development may influence project re-works and over-runs if their efforts are not coordinated. The negative effects of collaboration can also be found in the literature on role theories. A study by Rajendran (2005) finds that an individual's personal orientation (personality preferences) significantly impacts on their team’s performance. Consequently, agile methodologies’ emphasis on team cohesion may initiate risks associated with social issues.

The anecdotal nature of the evidence presented by the studies above may affect the validity of the authors’ assertions. However, since software is produced primarily by humans, arguments suggesting that individuals’ personality preferences determine overall team performance are subtly valid. Theories on the risks that may arise through agile methods’ stakeholders’ interaction practices are further explored in Chapter 4.

2.4. Conventional Vs Agile Methodologies

Software development methodologies were derived to help software teams manage the software development process. Although there are several predefined software development methodologies documented, software projects over the years have often been characterised by processes which may follow one methodology, a mix of methodologies, or sometimes none at all (Cockburn, 2004). Underlying this varied usage seem to be mixed opinions on each methodology’s strengths, and suitability for any given project (Abrahamsson & Koskela, 2004; Abrahamsson et al., 2003; Boehm & Turner, 2003).

According to Williams (2005), software projects that typify an environment of complexity, uncertainty, and are time limited, should adopt agile methodologies. On the other hand, Boehm & Turner (2003) assert that software projects that are highly predictable, stable, require few changes, and mission critical, should be developed using conventional methodologies. These two views seem to be common in the literature. Nonetheless, the previous authors present very little rationale for their assertions. In addition, Boehm & Turner (2003) contradict the general body of knowledge which has established that software product development is on no account predictable and stable (Cockburn, 2004; Coram & Bohner, 2005).

While each methodology may yield benefits in their allied environment, the appropriateness of the generic models most common in the literature are presented in Table 2.1. In Table 2.1 it is illustrated that these methodologies differ on the dimensions of: people, processes, and tools. There are also novel arguments in the literature which propose that agile methodologies can be adapted to suit large software projects (Cockburn, 2004; Schwaber & Beedle, 2002). Accordingly, agile methodologies are said to possess strength in either context. However, no evidence exists to confirm this assertion. This highlights the need for research examining and improving agile processes and their effectiveness and suitability for software development.

Table 2.1. Conventional versus Agile methodologies (Boehm & Turner, 2003; Nerur et al., 2005)

Project Characteristics	Conventional Methodologies	Agile Methodologies
Fundamental Assumption	Systems are fully specifiable, predictable, and can be built through meticulous and extensive planning	High-quality, adaptive software can be developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change.
Control	Process centric	People centric
Management style	Command and control	Leadership and collaboration
Knowledge Management	Explicit	Tacit
Role Assignment	Individual favors specialisation	Self organising teams, encourages role interchangeability
Communication	Formal	Informal
Customer’s Role	Important	Critical
Project Cycle	Guided by tasks or activities	Guided by product features
Development Model	Life cycle model (waterfall, spiral, or some variation)	The evolutionary-delivery model
Desired Organisational Form/Structure	Mechanistic (bureaucratic with high formalisation)	Organic (flexible and participative encouraging cooperative social action)
Technology	No restriction	Favors object-oriented technology
Size	Larger teams and projects	Smaller teams and projects

2.5. Summary

Software engineering techniques have now been employed for many years to guide software product creation. Early waterfall techniques used in the software development process were often criticised as not appropriate for software process management. Thus, the failure of software projects has often been (unfairly) linked to these waterfall processes. Though there have been some variations created to the conventional waterfall process, these too are often assessed as inappropriate to manage software production.

In light of this, there has emerged a new set of agile software development methodologies aimed at reducing software projects risk and thereby inducing software production success. The new methodologies are implemented using processes contrasting to the previous models. However, the new processes also present complications and risks in software engineering practice. Consequently, there seem to be arguments in support of each methodology’s appropriateness in software projects. Conventional methodologies are said to be suited to software projects that are highly predictable, stable, require little change, and are mission critical; whereas agile methodologies are recommended for use in software projects that are implemented in an environment of complexity, uncertainty, and time limitation. Since the practices recommended by agile methodologies to deal with risk are also risky, it is worth considering how risk management can be implemented to reduce risk in both development contexts. The following chapter examines software project risk management.

3. Software Project Risk Management

According to cases reported by Charette (2005), software project failures account for billions of dollars. In 2001, Nike Inc reported losses of about $100 million due to their supply-chain management system failure. The Sydney Water Corporation cancelled their billing system software project in 2002 after investing $33.2 million. In 2003, AT&T Wireless reported losses of around $100 million due to glitches in upgrading to a new customer relationship management software system. Further, Avis Europe PLC in the United Kingdom reported $54.5 million losses after cancelling the development of their enterprise resource planning system. Failure in these projects was linked to a variety of reasons: unrealistic project goals, inaccurate estimates, poor system requirements engineering, inadequate reporting of the project status, unmanaged risks, poor communications among projects' stakeholders, use of immature technologies, unmanageable projects' complexities, inadequate development practices, inadequate project management, stakeholder politics and commercial pressure (Charette, 2005).

Even though 'unmanaged risks' is explicitly stated as one of the reasons for failure in the above projects, the other reasons cited for failure can also be classified as possible risks areas. According to Morski & Miler (2002), effective risk management techniques often allow project managers the opportunity to identify potential risk, thereby permitting them to change their course of action in order to mitigate those risks. In addition, Roy (2004) argues that risks are often mitigated if risk management is an evolving and learning process. Thus, to avoid or lessen the impact of failure, risk mitigation techniques should be implemented through careful qualitative and quantitative examination of *all* the failure areas above.

Since the lack of effective risk management may impact negatively on software project outcomes, of relevance to this research are the ways that project risk management can be implemented in order to reduce the likelihood of failure. Hence, in this chapter the fundamental notions of projects and software are introduced, conventional project risk management techniques are described, agile project risk management techniques are then examined, followed by a summary of the theories explored throughout this chapter.

3.1. Introduction to Projects and Software

According to the international Project Management Institute (PMI), a project is a venture which is undertaken to produce a product or service (P.M.Institute, 2000). In software development, this product or service is the software system. Software project management is aimed at organising the responsibilities of the software team in order to achieve specific goals. This discipline dates back to the commencement of software engineering in the 1950’s (Williams, 2005). Prior to the commencement of software engineering, the project management profession comprised of around 130,000 professionals represented by the PMI and the International Project Management Association (IPMA).

The main reasons cited for the establishment of the PMI and the IPMA were: projects’ potentially complex nature due to their being time-limited, evolutionary, possessing transient teams, and risky (Williams, 2005). Consequently, the PMI and IPMA formalised agreed standards for project management which were later documented in the project management body of knowledge (PMBOK), and the body of knowledge (BoK) respectively (Dixon, 2000; P.M.Institute, 2000). The PMBOK and BOK identify five processes which are common in a project’s life cycle, and nine knowledge areas suitable for guiding management processes (P.M.Institute, 2000). The guide presented in the PMBOK offers potential for supporting project management in a generic sense. However, there are many criticisms in the literature with regard to its effectiveness when used as a guide for projects with rapid change (Chin, 2004; Hodgson, 2004), although change is said to be a main characteristic of software development.

Among its recommendations, the PMBOK guide states that if the guidelines presented are followed, then project success becomes inevitable (P.M.Institute, 2000). In addition, this guide notes that the project management process bears validity across all disciplines. Nonetheless, there are contradictions to these statements in the literature (David, 1991; Williams, 2005). Arguments presented by David (1991) and Williams (2005) claim that there is evidence of widespread failure across all disciplines in spite of projects having employed the guidelines of the PMBOK and BoK. Failures are often linked to poor risk management (Winch, Millar, & Clifton, 1997). Of particular relevance to this thesis are risks that result from uncertainty in project goals, personality differences, and project complexities which are regularly linked to project failure (Standish Group, 2001; Williams, 2005). Risks associated with poor productivity (and therefore success) often arise due to personality differences among team members (see Bradley & Hebert (1997) for further details).

In software projects, project managers apply project management techniques to guide the software development life cycle (SDLC). Though there are variations to the implementation of the SDLC in software projects (see Chapter 2), most software projects comprise some or all of the following activities (Siddiqui & Hussain, 2006): conceptualisation, requirement and cost benefit analysis, project scoping, specification of software requirements, design of systems (architectural design and detailed design), module development, integration and testing, system testing, installation, meeting acceptance conditions, user and technical training, and maintenance. Thus, the software project manager is expected to link their understandings of the technology with project management principles, and administer their leadership skills to induce success through coordinating and implementing the various activities of the SDLC (Brewer, 2005).

Consequently, selecting the most appropriate project manager is also a critical risk factor in relation to project success (Standish Group, 2001). While being versed in the areas identified by the PMI such as time management, risk management, scope management, and costing and budgeting (P.M.Institute, 2000), Melymuka (2000) stresses that soft skills are also critical for project success. This view coincides with the line of thought put forward by Bradley & Hebert (1997) and Brewer (2005) who claim that relationship management is an essential ingredient for team coordination and for successful customer interactions, leading to effective project management.

The idea presented by Melymuka (2000) appears to be minimally considered in the literature on risk management. According to Chapman & Ward (2004), risk is an uncertain event or set of circumstances that can potentially hinder project objectives. This seems to suggest that uncertainty management does not necessarily describe only perceived threats, opportunities, and their implications in a software project, but the identification and management of all potential sources that give rise to threats throughout projects. To this end, the PMBOK identifies six key risk processes (P.M.Institute, 2000): risk management planning, risk identification, qualitative risk analysis, quantitative risk analysis, risk response planning, and risk monitoring and control. Accordingly, the PMBOK guide suggests brainstorming and assumption analysis as useful mechanisms for risk identification, coupled with a survey of risk categories such as technical and organisational risks.

Similar to the PMBOK, the US-based Software Engineering Institute (SEI) uses a classification-based model to identify software risks (Carr, Konda, Monarch, Ulrich, & Walker, 1993). In their model, categorisations are based on product engineering, development environment, and program constraints. Risks are identified through the answering of questionnaires, each category relating to a specific list of factors associated with the questions. The techniques for risk identification presented in the models by the PMBOK and SEI are often criticised as technically biased and process centered (Kirk & Tempero, 2006; Verner & Evanco, 2005; Williams, 2005). For this reason, other variations to these models warrant examination.

Further use of a classification model is relation to project risks is described in Klein (1999). He recommends answering the following questions to assist with risk identification:

1. Can the developers describe clearly and in detail the objective and requirements for the proposed system?
2. Are the requirements consistent and feasible?
3. Is there a clear approach or plan for developing the system?
4. Have adequate time and resources been allocated for this project?
5. Is there certainty that the off-the-shelf items being used are suitable for the problem at hand?

6. Are the relevant cultural communities working together properly?
7. Are all the relevant ancillary issues being addressed?
8. Is there a well defined process with clear criteria for determining completion?
9. Is there a significant ongoing effort to identify and mitigate project risks?
10. Is the status of the project clear to management and other relevant stakeholders?

To ensure a clear view of the project prior to answering the recommended questions, Murthi (2002) also suggests doing the following generic risk-related tasks: establishing potential impact of each risk, ranking risks according to potential impact, calculating the probability that the risk will occur, ranking risks by their combined impact and probability of occurrence, developing contingency plans for major risks, determining the resource requirements for the contingency plans, putting risk information in the review plan, tracking risks as the project progresses, and periodically evaluating and modifying the risk evaluation approach.

While the approaches mentioned above might help in the evaluation of technological project conditions that might equate to risks (Kirk & Tempero, 2006), there exists literature which asserts that a more behavioral approach to risk analysis is needed (Verner & Evanco, 2005). Early studies by Curtis (1989) and Curtis, Krasner, & Iscoe (1988) found that software projects were often affected by a small number of human related issues. They affirm that such issues were often critical to project success, and directly related to lack of shared vision and domain knowledge, requirements uncertainty, and poor communication. These issues are long-standing - a recent study by Verner & Evanco (2005) shows that excessively focusing on the technological aspects of project risks may be in itself risky, and that the greatest project risks are influenced through social factors.

Therefore, considering the previous literature on project risk, it might be said that the best project management techniques should take an all encompassing view regarding project risk analysis, covering both technological and behavioral dimensions. Conventional systems development procedures are characterised by process-centered risk management (Williams, 2005). In contrast, agile software development approaches are said to ignore risk management. These approaches are alleged to employ light techniques (see Chapter 2 for agile practices), which are said to eliminate uncertainty (Chin, 2004). However, in both conventional and agile environments, project risks are inevitable. Additionally, if the theories presented by Curtis (1989), Curtis et al. (1988), and Verner & Evanco (2005) above hold true, the processes employed in the latter project environment offer significant potential for social risks to occur. In the remaining sections of this chapter, consideration is therefore given to techniques presented in the literature that are said to assist with conventional project risk management, and agile project risk management. A summary of the theories presented throughout this chapter then follows.

3.2. Conventional Project Risk Management Techniques

Conventional project management techniques mirror the systems analysis approach of the traditional software development methodologies (Murthi, 2002). This heavily process oriented approach, while adding the impression of control, does not automatically eliminate risk from software development. In fact, Murthi (2002) asserts that these methods of project management *induce* risks into the software development process through delays and unnecessary overhead. Nevertheless, there exists a dominant discourse that these methods, such as those recommended by the PMI and SEI, are implicitly correct (Williams, 2005).

Thus, two trains of thoughts are presented in the literature. Packendorff (1995) contends that in conventional project organisations, planning processes are excessively heavyweight and these often delay execution processes. Similarly, Koskela & Howell (2002) argue that according to the PMBOK, management is seen as planning, and in this framework one execution process is often preceded by ten planning processes. On the other hand, Boehm & DeMarco (1997), Jiang & Chen (2004), and Schoenthaler (2002) assert that thorough risk management techniques recommended by risk theories reduce failure. They say that the cost of plans may be insignificant when compared to the cost of corrective actions after project failure.

In addition, they affirm that project risks cannot be reduced or eliminated without implementing risk management techniques. Taking the previous views into consideration, regardless of the software project, the goal of software project risk management should be to ensure the reliability of software development. Thus, the role of project managers should be to identify possible threats to the software project development and deployment atmosphere, and devise strategies and plans to mitigate against those threats.

Accordingly, risk management (listed in the PMBOK among nine areas of project management) is said to be one of the most crucial areas of project management (P.M.Institute, 2000). Similarly, the SEI continually stresses the importance of risk management in software projects (Gallagher, 1999; Gallagher, Alberts, & Barbour, 1997). While there are some controversies over the SEI's approaches to risk management techniques (Kontio et al., 2004), there exists empirical evidence which supports the necessity of risk management in all software projects (Charette, 2005; Standish Group, 2001).

Previous literature shows that there are several variations to conventional risk management techniques. In addition, previous research stresses the importance of implementing appropriate risk management in software projects. The SEI presents a wide-ranging risk taxonomy (Gallagher, 1999; Gallagher et al., 1997). In this taxonomy, 194 possible areas of risks are identified, and corresponding questions are provided in order to assist with risk identification in specific areas. Similarly, Karolak (1998) proposes a quantitative approach examining 81 risks factors or perspectives. Questions are also presented by Karolak (1998) to enable risk identification. Roy (2004) and Siddiqui & Hussain (2006) also propose a quantitative project risk management approach that is designed to monitor the processes involved in the waterfall and incremental models. Additionally, Serich (2005) recommends a model which uses the constructive cost model (COCOMO) representation and places specific emphasis on risk management.

The models proposed by Gallagher et al. (1997), Karolak (1998), and Roy (2004), and their associated data collection and analysis processes, may be overly time consuming when implemented in time-limited software projects. In addition, these authors seem to give scant consideration to behavioral risks such as those resulting from personality differences through team members' involvement, or risks associated with organisation politics, and changing business requirements. This could be problematic if it is accepted that, as mentioned earlier (Murthi, 2002; Verner & Evanco, 2005), the most critical software development risks are linked to social issues.

Further, there exists empirical evidence which shows that project managers rarely apply the processes defined in the associated theories when executing their real life projects (Moynihan, 1997; Verner & Evanco, 2005). In addition, Moynihan (1997) found that in such project environments successful project managers quite often substituted formal risk management processes with expert judgment. These findings could be taken to imply that the models recommended in the literature do not adequately address risk management approaches suitable for at least some software projects. In particular, their applicability to agile development has been questioned. In light of this, the following section examines popular techniques used for agile project risk management.

3.3. Agile Project Risk Management Techniques

Agile project management practices are intended to support the processes enacted by agile software development methodologies. Thus, in a similar manner to the way agile methodologies are said to deal with project uncertainty (see Chapter 2), the associated project management techniques are also meant to support emerging projects (Williams, 2005). Augustine et al. (2005) stress that in an agile project environment there is minimal control and projects are developed in an adapting atmosphere where plans are often short and iterative. While an adapting and iterative form of project management may prove suitable to a software production environment where change is expected (Bostrom, Wayrynen, Boden, Beznosov, & Kruchten, 2006; Lehman & Ramil, 2001), some have expressed reservations over the negative impact that too much change can have on software development (VanDeursen, 2001). That is, it is asserted that there is a point at which even methods adaptable to change can be overwhelmed.

According to early work by Gilb (1988), software project management techniques are coupled closely with planning and documentation – but these activities are not predominant in agile projects. It is also important to note that planning by itself does not necessarily eliminate risk, though the systematic gathering of information to assist with risk assessment may aid in risk reduction (Highsmith, 2004). In any case, there still seems to be some uncertainty as to whether risks are introduced through the lack of planning in agile projects. In addition, there exists little empirical evidence to ascertain whether the management techniques that are (meant to be) employed by agile project teams are actually effective in reducing risk.

Proponents of agile methodologies do not support the employment of 'heavy' plans often common in more conventional software development settings. In addition, they argue that the current framework for software risk management is not appropriate for agile project management (Highsmith, 2004; Kontio et al., 2004). This view is expounded by Kontio (2001), who added that a formula driven approach to calculate risk is not suited to fast paced development, and such a method is mathematically unsound. That is not to say that there should be *no* planning. It is important to note that agile allies support the development and use of the appropriate amount of plans that are necessary for software development, still allowing for agility.

To that end, risk management is continually stressed in the agile project management literature (Chin, 2004; Highsmith, 2004). According to Highsmith (2004), until a final product emerges, the life cycle should be one of constant information gathering. In addition, he says that each question answered, and each new piece of information uncovered, throughout the software development life cycle should be aimed at reducing risk (agile methodologies being well suited to assist with this). For those reasons, Highsmith (2004) goes on to identify five risks that are most likely to be reduced in agile software projects: inherent schedule flaws, requirements inflation, employee turnover, specification breakdown, and poor productivity. To understand how agile methodologies are said to reduce the five risks previously mentioned, they are further considered in detail.

3.3.1. Inherent Schedule Flaws

It has been stated that the most resilient risk mitigation technique in agile development projects is incremental delivery (VanDeursen, 2001). Inherent schedule flaws result when product size is misestimated or there are unrealistic timelines set in the software project plan (Highsmith, 2004). Highsmith (2004) states that in uncertain software development projects, scheduling the unknown may result in a project failing to meet the schedule. Hence, the process of exploration, common to most agile software methodologies, may reduce this risk. Most agile practices address schedule risk in some way (Williams, 2005) (see Chapter 2 section 2.3.1 for agile practices). According to Chin (2004), the agile project teams' direct involvement in planning and estimating, the emphasis on early feedback regarding delivery speed, the constant pressure to balance system features with schedule constraints, the tightly coupled team (engineers and customer), and the principle of early error alleviation potentially reduce schedule risk in these projects.

3.3.2. Requirements Inflation

Requirements evolution is popular (and is in fact welcomed) in agile projects (Abrahamsson et al., 2003). Under this approach, engineers and customers are constantly considering cost and time while evolving the projects' requirements. This approach may reduce unanticipated (and therefore unmanaged) requirements inflation in agile projects, as both the developer and customer are involved in any changes to be made to the project requirements.

3.3.3. Employee Turnover

Employee turnover is an inherent risk factor in all software projects. Highsmith (2000, 2004) establishes that the impact of turnover can be reduced through cross training and appropriate amounts of documentation. Williams & Kessler (2003) assert that the agile practice of coupled teams induces a great degree of knowledge sharing, which is likely to reduce the risk of turnovers. The coupled team practice employed in agile project environments may also induce extensive tacit knowledge gain, which may also reduce risks.

3.3.4. Specification Breakdown

This occurs when the stakeholders fail to agree on a specification, due to conflicting intentions between the customer and the developers. In an agile project, the project manager and the customer are responsible for managing specification and workflow. There are arguments in the literature that seem to suggest that agile methodologies' practice of heavy customer involvement may reduce risks associated with specification breakdown (Grisham & Perry, 2005).

3.3.5. Poor Productivity

According to Highsmith (2004), the risk of poor productivity arises from three sources: having an inefficient team, having a team that is not cohesive, and having a team with poor morale. Several agile practices may potentially reduce these risks. Highsmith (2004) explains that agile approaches of having the right people on the team, and coaching the team and promoting team development help to offset the risk of poor productivity. Further, some agile methodologies such as XP and SCRUM focus on short iterations. Chin (2004) asserts that this may also mitigate risks associated with poor productivity.

While the practices presented by the authors above may help with risk mitigation (in principle) in agile projects, there is a lack of empirical evidence to support this position. Nonetheless, agile supporters seem to agree that, as with all software projects, risk management techniques should be considered throughout the agile software development life cycle. Through effective risk consideration, agile project managers may be better able to set realistic project expectations, recover quickly from problems through earlier conceived contingencies, and instrument preventative actions to lower the impacts of problems. Accordingly, Chin (2004) identifies four parts to the agile risk management process: identify potential risk, assess the risk, make plans to address the risk, and reassess the risks throughout the project. This process largely mirrors the thinking in conventional project management, even though the techniques recommended by conventional project management are often criticised by agile proponents.

Several steps are followed to identify risks: these include reviewing project planning outputs, reviewing project dependencies, quantifying unknowns, and reviewing previous lessons of similar projects. These techniques are both quantitative and qualitative. Therefore, the project manager is faced with the task of applying quantitative methods to calculate risk, or alternatively reflecting on previous experiences and observing agile projects' dimensions to mitigate risk. Chin (2004) explains that risk assessment is supported by describing each potential risk, and estimating risk outcome and impacts. He says that the risk plan should consider risks that are going to be managed by the project, and once these are considered, a mitigation plan should be constructed. For every iterative release, the agile project manager should reassess risk in the new iteration.

The literature seems to suggest that agile practices (see Chapter 2 section 2.3.1 for common agile practices) which are said to eliminate project risks, are themselves risky. In addition, in the absence of plans and a quantitative framework for risk identification, the agile risk management process appears to be more qualitative in nature. Irrespective of whether qualitative or quantitative approaches are used, software projects that lack risk management techniques may be perilous. Since there is great emphasis on team members' interaction in agile methods and this practice is said to attract the most critical risks, the following chapter presents a review of the theories around the particular risks induced through human collaboration.

3.4. Summary

From its inception software project management was a knowledge area created to assist project teams. In particular, this discipline serves as a guide to the project manager to help with organising the responsibilities of the software team in order to achieve specific goals. Software project risk management is listed as one of the most critical areas in software project management literature. That said, the recommendations made by the early bodies of general project management to assist with risk management in software projects were not quite suited to the management of risks in these projects. Previous recommendations were often technically focused, while the most critical risks in software projects are said to be induced through social factors.

While the present risk models, such as the SEI's risk analysis framework, are more applicable to software risk analysis than these earlier efforts, questions remain over their effectiveness. Nonetheless, there is evidence that supports the importance of risk management in software development projects. Thus, there are several variations to the model presented by the SEI in the literature. However, research has shown that project managers rarely adhere to strict employment of risk management in real life software projects.

Recent approaches to risk management in agile projects seem to be mostly qualitative in nature, dependent on expert judgment. In addition, the practice adopted by agile methodologies does not allow for a heavy process-centered approach to risk management. Besides, agile methodologies are said to mitigate most of the risks inherent in the conventional software development methodologies. While some of the practices recommended by agile project management may mitigate several project risks, it is doubtful that all agile software projects are entirely 'safe'. Further, the most critical project risks are said to be social in nature. Thus, several agile practices may actually introduce serious risks to the software project environment. Therefore, the following chapter considers the risks that may become prevalent in agile projects through human collaboration.

4. Agile Processes: The Effects of Stakeholders' Interaction

Since significant software failures still occur (Charette, 2005; Standish Group, 2001), and software plays a major role in our lives today, activities aimed at assessing and improving the processes employed in successful software creation are of vital importance. Academic research should continue to be aimed at evaluating the reasons for software failures (and successes), and should present recommendations regarding ways to reduce such failures. In this chapter, some of the processes recommended by agile methods are assessed, with the intent of identifying ways to improve agile practices. To this end, agile process social risks are examined. Consideration is given to four subject areas: psychology of human collaboration, management and role theories, role theories and IS research, and risk of customer involvement. The chapter then closes with reflections on the literature explored throughout the chapter, and possible directions through which the risks of agile methods social processes could be addressed.

4.1. Human Collaboration

Evident among the arguments presented in support of agile processes are two lines of thought regarding the social process improvements made possible through the use of agile methods. One line of thought argues for extensive stakeholder interaction in software projects without condition (Murthi, 2002), as against another position which supports stakeholder interaction *after* the critical process of team formation (Acuna, Juristo, & Moreno, 2006). Since the research literature reveals mixed findings regarding the former argument, the objective and reliable literature examining the latter is worth considering. To that end, this section considers social risks (which are also referred to as people-related risks) that may affect software development projects employing agile processes.

In Cockburn & Highsmith (2001), the authors emphasise the importance of the people factor in software projects. They go on to mention the ingredients for project success; these, they posit, are talent, skill, and communication. Many authors share similar views (see Clavadetscher (1998) for example), and the activities recommended by agile processes such as those bringing the team and the customer together, and others emphasising ongoing oral communication among team members and customers, reinforce the ideas embraced by agile proponents.

Empirical evidence verifying the benefits of bringing people together to employ agile practices has reported mixed findings. For example, Nosek (1998) observed 15 programmers working on a 45 minutes long task in live settings to verify the benefits of collaboration. He found that collaboration improved the programmers' performance (software development took less time, and there was also improvement in software quality) and enhanced appreciation of the problem solving process. Williams, Kessler, Cunningham, & Jeffries (2000) experimented with small groups of students working on class assignments to verify whether XP's pair programming yielded benefits over solo programming. Their findings revealed that software quality was enhanced for teams using pair programming, and software development by pairs took slightly less time than by single individuals.

McKinney & Denton (2005) and Wellington, Briggs, & Girard (2005) also experimented with small groups of students working on class projects to investigate the effectiveness of collaboration and team cohesion using agile practices such as pair programming, collective code ownership, and on-site customer interaction. They reported mixed findings; through observation they revealed that teams were not cohesive initially, but fared better as the projects studied moved close to completion. While studies by McKinney & Denton (2005), Wellington et al. (2005), and Williams et al. (2000) were conducted to support an area that lacks theories, questions also arise over the validity of studies employing students as subjects (Collis & Hussey, 2003).

Additionally, the studies above all shared similar characteristics with regard to task size. Research exists that shows that task size and complexity affect the outcomes of problem solving practices (Barki & Rivard, 1993; Chang & Ho, 1997). Barki & Rivard (1993) and Chang & Ho (1997) assert that the significance of development practices is rarely observed in small projects. Therefore, the validity of the findings of studies by McKinney & Denton (2005), Nosek (1998), Wellington et al. (2005), and Williams et al. (2000) might also be affected by the task sizes observed by these authors.

A study by Lan & Peng (2005) employing students as subjects considered the benefits of collaboration. In Lan & Peng (2005) study, which controlled task size, they reported differing findings to the previous body of knowledge put forward by (Nosek, 1998; Williams et al., 2000). Their findings revealed that some teams working in pairs divided due to personality conflicts and communication problems. Additionally, they found that persons with high competence performed well with low competence persons as they were able to exert control; and persons working alone produced superior quality work. Lan & Peng (2005) observed that the effect of collaboration hinged heavily on team members' personality orientation. Findings opposing the benefits of pair programming are similarly reported by Hulkko & Abrahamsson (2005) in a case study observing four software teams in real software development settings. They found that pair programming offered no clear benefit over solo programming.

While the negative impact of collaboration might have not been severe for the projects observed by Hulkko & Abrahamsson (2005) and Lan & Peng (2005), failure in software projects such as the FoxMeyer Drug Company in 1996 which resulted in bankruptcy, and the US Federal aviation software project which was cancelled after a $2.6 billion US dollars investment (see Charette (2005) for further details) can be catastrophic. Thus, collaboration should be closely examined, as team cohesion and the management of interpersonal skills are most critical to project success (Verner & Evanco, 2005). Consequently, the concepts and evidence underpinning personality and role theories are examined in the following sections.

4.1.1. Psychology of Human Collaboration

Research proposing or evaluating psychology theories considers the impact of personality type and behaviour (Jung, 1971; Montgomery, 2002; Myers & McCaulley, 1985). This discipline seeks to understand personality type and its influence on individuals' strengths and qualities, and their ability to communicate and form/sustain relationships in teams. Additionally, some studies are primarily concerned with the effect of the mix of personality types in groups, how personalities interact to influence team performance, and the impact of behavioural difference on team work; for example: Belbin (2002) and McCrae & Costa (1990).

Accordingly, Myers & McCaulley (1985) used Jung’s theories (Jung, 1971) to develop the psychometric instrument called Myers-Briggs Type Indicator (MBTI) after controlled experiments during the 1940’s, 1950’s, and 1960’s. This indicator is intended to reflect an individual’s basic preferences. Similarly, Keirsey (1998) also used the work of Carl Jung and Isabel Briggs Myers to develop a psychometric instrument to identify an individual’s most dominant personality trait. A variation to Myers and McCaulley’s and Kirsey’s studies is Hofstede’s work. He conducted several experiments to determine cultural differences along five dimensions (see (Hofstede, Neuijen, Ohayv, & Sanders, 1990) for details). His findings revealed that your world view, behaviour and preferences, and the decisions you make are heavily linked to your experiences and cultural background.

The Hofstede et al. (1990) findings are consistent with the ideas put forward previously by Keirsey (1998) and Myers & McCaulley (1985). Myers & McCaulley (1985) studies reveal that individuals are expected to possess one of four preferences in their behaviour. For a person’s energetic preference: they are either extrovert or introvert; for what they perceive: sensing or intuition; for their decisions: thinking or feeling; and their lifestyle: judging or perceiving. Myers & McCaulley (1985) describe extroverts as often favouring interacting with people while introverts are complete by being alone. Sensing individuals prefer evidence that is factual or concrete whereas intuitive individuals favour exploration and alternative explanation. Thinking individuals are rational and favour formal methods of reasoning whereas feeling individuals focus their judgement on subjective values and their views. Judgers like pre-planning and order whereas perceivers live through adoption and spontaneous decision making. The validity and reliability of the MBTI instrument have been continually verified in studies done by Carlson (1985), Johnson (1992), and McCarley & Carskadon (1983).

Keirsey (1998) suggests that human personality belongs to one of sixteen types (see www.keirsey,com for more information). Like Myers & McCaulley (1985), he asserts that your dominant or preferred personality exists among one of four types (similar to MBTI above). However, there are additional details within each of these types which further classify individuals to one of sixteen types.

Even though people are often vulnerable to imprecise suggestions and misunderstanding when completing questionnaires (Kitchenham & Pfleeger, 2002), psychometric testing offers a way to at least partially understand the difference in individuals' behaviour. The psychometric tests above have been used widely by many organisations for recruitment, team assembly, assessment, and training and development (for example see http://www.knowyourtype.com/google.html, www.keirsey,com, and www.belbin.com). Key among the psychometric tests is Belbin's Self-perception Inventory (Belbin, 2002). Belbin Associates claim that this test has been used by over 40% of UK's top companies and millions of people around the world (www.belbin.com). Since Belbin's instrument has received such wide audience, and it has been verified (see Beranek, Zuser, & Grechenig (2005) for example), Belbin's work is further examined in the following section.

4.1.2. Management Background and Role Theories

An extension of the personality type theories in the psychology literature is found in Belbin's work (Belbin, 2002). Meredith Belbin conducted several studies to ascertain what particular human behaviour or attitudes are essential for team success (Belbin, 2002). Following several years of observation in Norway, the Czech Republic, the United Kingdom, Germany, Denmark, and South Africa, Belbin found that most successful teams possess eight different functional roles. He listed the roles as company worker, chairman, shaper, plant, resource investigator, monitor-evaluator, team worker, and completer-finisher (see Table 4.1. for details).

Table 4.1. Belbin team roles (Belbin, 2002)

Symbol	Role	Positive Qualities	Allowable Weaknesses
CW	Company Worker	Organising ability, practical common sense, hard-working, self-discipline	Lack of flexibility, unresponsiveness to unproven ideas
CH	Chairman	A capacity for treating and welcoming all potential contributors on their merits and without prejudice, A strong sense of objectives	No more than ordinary in terms of intellect or creative ability
SH	Shaper	Drive and a readiness to challenge inertia, ineffectiveness, complacency or self-deception	Proneness to provocation, irritation and impatience
PL	Plant	Genius, imagination, intellect, knowledge	Up in the clouds, inclined to disregard practical details or protocol
RI	Resource Investigator	A capacity for contacting people and exploring anything new. An ability to respond to challenge	Liable to lose interest once the initial fascination has passed
ME	Monitor-Evaluator	Judgment, discretion, hard-headedness	Lacks inspiration or the ability to motivate others
TW	Team Worker	An ability to respond to people and to situations, and to promote team spirit	Indecisiveness at moments of crisis
CF	Completer-Finisher	A capacity for follow-through, Perfectionism	A tendency to worry about small things. A reluctance to let go

Belbin asserts that in successful teams, the eight roles described in Table 4.1 are performed by the team members. Though Belbin's (2002) findings differ slightly from previous role theories which linked individuals to one personality trait, his work built on the MBTI ideas, and also agrees that a person's interaction in a group is influenced by their personality trait. Among his findings he also reported that individuals who possessed premium quality in one respect were often lacking in others, and that combining individuals with similar personality traits reduced performance. Successful teams are heterogeneous; normally possessing a balance of team members occupying all roles. Individuals can possess more than one personality preference, having a primary preference and other secondary preferences. Individuals are most comfortable when they are functioning in roles that are their natural preference. Interaction between different personalities without understanding and managing their differences can be a source of conflict.

During his studies on team roles Belbin developed a psychometric test called the Self–perception Inventory (SPI). The SPI is a questionnaire used as an indicator for determining an individual's personality preference. The test consists of seven sections with eight questions for each section. For each section individuals allocate a total of ten points based of how they feel about the questions. As mentioned earlier, there exist studies which have verified the accuracy of Belbin's theories; Beranek, Zuser, & Grechenig (2005), Jones (1999), and Rajendran (2005) have offered confirmation to Belbin's findings.

4.1.3. Role Theories and IS Research

The consideration of group dynamics and expertise coordination is not new to software development and information system research (Acuna & Juristo, 2004). Gorla & Lam (2004) use the MBTI model and studied 92 software developers from 20 software companies in Hong Kong to verify whether personality type is linked to team performance, and the effect of heterogeneity of personalities on team performance. Their findings confirm those of Myers & McCaulley (1985) and Belbin (2002). They found that personality preference influences the way team members communicate and personality preference also has a significant impact on team output. Additionally, they found that the higher the heterogeneity of software teams, the higher their productivity levels are likely to be.

Similarly, Rajendran (2005) employed the Belbin model for personality assessment to observe three software development teams. Consistent with Belbin (2002), he found that positive and negative qualities are associated with personality preferences. Rajendran (2005) also found that, through personality assessment, teams can be constructed to possess a mix of members with positive personality traits, thereby reducing the risk of personality conflicts and enhancing the likelihood of success in team development tasks.

Previous studies by Bradley & Hebert (1997) and Faraj & Sproul (2000) also substantiate the Myers & McCaulley (1985) findings. In a case study of two software teams, Bradley & Hebert (1997) found that team composition of personality types influences team performance. They also found that a united team is an important variable for team effectiveness and team success. Additionally, their findings indicated that the team with a balance of personality types performed better than the team that was homogenous. In a study involving 69 software development teams, Faraj & Sproul (2000) also reported that managing personality traits has a significant positive impact on software development performance.

Studies by Beranek et al. (2005) and Jones (1999) in student settings also offer insightful support for linking personalities to roles. Using Belbin's model, Beranek et al. (2005) reported that personalities associated with negative attitudes, such as a 'reluctance to share tasks' or 'being overly critical' created a negative impact on success. Beranek et al. (2005) found that such individuals pose a significant threat to the success of the team and other team members. Additionally, Beranek et al. (2005) observed that an individual's willingness to participate in group settings also plays an important role in successful team work. These findings are consistent with those of Jones (1999) who reported that personality traits are significantly correlated with team cohesion, and team cohesion is a necessary ingredient for team success.

The previous studies investigated here have all reported similar findings regarding the importance of personality traits for software teams' success. Cohesion has also been repeatedly acknowledged as a crucial element for team effectiveness (Verner & Evanco, 2005). Recent research shows that the processes recommended by agile methods have been attracting industry attention, and agile processes are increasingly being adopted by software teams (Behrens, 2006; VersionOne, 2006b). Among the processes recommended by agile proponents, human collaboration is heavily emphasised (Cockburn & Highsmith, 2001). Therefore, research examining ways of improving this process should be fruitful for enhancing the performance and success of software developers.

Even though researchers have focused on improving the social process related to software development (Bradley & Hebert, 1997; Faraj & Sproul, 2000; Rajendran, 2005), previous research has failed to offer a complete solution to personnel capabilities management. This study offers a mechanism for solving this problem by implementing Belbin's SPI in a software tool to permit project stakeholders to electronically determine their personality preference or capability, and storing the personality preference information. This can then be used by project leaders in assigning team members to roles, in so doing, reducing risk of conflicts and software project failure. This work is more fully described in Chapter 6.

4.1.4. Risk of Customer Involvement

In this section the risks associated with customer involvement are examined. The terms ‘user’ and ‘customer’ are used interchangeably throughout this discussion, though in other contexts these two terms may be different (the customer is often the group or individual paying for the software project, whereas the user is taken to be the intended user of the system).

Extensive and active customer involvement is often cited as a key to project success (Clavadetscher, 1998; Jiang, Klein, & Balloun, 1996). In this context customer involvement is seen as their active participation and involvement in requirements specification, software testing, and other development practices often recommended by agile models. This assertion is especially dominant in agile processes; an extreme endorsement of user involvement is evident in XP’s onsite customer practice (Beck, 2000). While customer involvement may be beneficial in principle, there is little actual evidence verifying that this practice significantly improves the outcomes of software systems development. Furthermore, some research reports that there are in fact several pitfalls to extensive customer involvement.

Previous studies examining whether customer involvement affects customer satisfaction have reported mixed findings. In a study involving 107 customers in the service industry, Goodman & Fichman (1995) find that high customer involvement increases customers’ overall dissatisfaction, especially when customers are dissatisfied with product performance. They assert that customer involvement is often only functional when firms perform well, and poor performance increases overall dissatisfaction. Similarly, Grisham & Perry (2005) and Tesch, Jiang, & Klein (2003) find that customer satisfaction is influenced by their prior expectations. Grisham & Perry (2005) and Tesch at al. (2003) establish that when team performance is higher than customer expectations there is greater conformity and higher customer satisfaction. However, when the reverse occurs, customers are often dissatisfied.

A longitudinal study by Heinbokel, Sonnentag, Frese, Stolte, & Brodbeck (1996) of 29 software projects also reported negative effects of customer participation. Heinbokel et al. (1996) found that customer involvement lowered developers' overall success and software teams' effectiveness. In addition, they also reported that the projects studied were not innovative and offered little flexibility. In the same way, Sharp et al. (2004) studied 19 software developers in an activity session and found that heavy customer involvement threatened software projects' success in a number of ways. While these findings may not generalise to all software projects, the results indicate that incidents of conflicting views, skills difference, customer exposure to sensitive information such as schedule slippages and technical issues, and customers asserting too much dominance in developers' decisions are major risks that may influence project failure and may lead to customers and developers falling out.

On the contrary, a study by McKeen & Guimaraes (1997) involving 151 software projects found that customer participation induces customer satisfaction. Similarly, Foster & Franz (1999) studied 87 customers and 107 analysts and found that customer participation was correlated with system success and acceptance. Though the findings of these studies support the position of agile methodologies, in these studies user satisfaction was only pronounced in tasks that were simple and offered the customer control (for example: requirement specification). However, agile methodologies place significant emphasis on intensive and ongoing customer involvement, and recommend customer and developer collaboration on development tasks (for example: system testing) in an environment which might not necessarily mirror the settings of the projects considered by Foster & Franz (1999) and McKeen & Guimaraes (1997).

Additionally, customer involvement in software teams is also likely to be affected by the risks associated with personality differences (Barki & Hartwick, 2001; Newman & Robey, 1992). Accordingly, studies examining the impact of customer involvement and ways of gaining the maximum benefit of customer participation in software development projects are likely to offer significant benefit for software development.

The findings of previous studies suggest that customer involvement is most beneficial when teams are successful, when customers and developers share similar views, and when there is minimal skill difference among the customer and the developers. Accordingly, onsite customers may prove to be a threat to project success under circumstances in which the above conditions are not met – for instance, when software teams perform poorly, or where the team lacks balance in skills or personality. Therefore, lessening the direct interaction with customers may reduce potential conflict and the negative impact associated with interpersonal conflict. One such way to lessen direct interaction with the customer while still accommodating their presence is to extend the customer-developer interaction through other mechanisms. This can be achieved by enabling remote customers to use a tool interface. Such an interface should simulate the face-to-face environment, providing the customer with the opportunity to initiate requirements, and participate in the management of requirements.

Previous tools supporting project management have not considered the employment of a customer interface, allowing an extension of the onsite customer (see tool evaluation in Table 5.1). As a result, the second objective of the current study is to bridge this gap by creating a feature management tool which extends the customer by means of a customer interface.

4.2. Summary

The challenges presented by software development influence studies examining the processes employed in this endeavor. Recent studies show that agile methods have been gaining industry attention. However, mixed evidence exists regarding the effectiveness of the practices recommended by agile methods. Therefore, efforts to verify the most effective ways of employing agile practice should be undertaken. Since social risks are deemed critical to software project success, the current study closely examines issues relating to human collaboration.

An individual’s personality preference and cultural background influences how they contribute in group settings. Previous studies show that personality conflicts negatively affect team cohesion and software project success. Personality conflicts are said to be mostly influenced through poor team formation. Therefore, assigning team members to specific roles based on their personality preference and managing personality preference information should reduce risks associated with personality conflicts. Customer involvement is (in principle) beneficial to the software development team, especially when they perform well. However, evidence shows that customers may pose a threat when teams perform poorly. Reducing direct interaction with customers (whether or not software teams performs favorably) by allowing them a tool interface to the development team may also reduce conflict. As detailed in Chapter 6, the current study offers a toolset that allows personality assessment and management, and feature management, extending the customer interaction through an interface. Prior to this, however, consideration is given to the use and effectiveness of tools in general in relation to software development and its management.

5. Software Development Tools

In this chapter, the role of tools used during software development projects is considered. An introduction to software tools is presented, forming the basis and specifying the direction of this chapter. Secondly, an examination of the importance of project management tools follows. Tools associated with the two software paradigms under consideration in this research are then presented in subsequent sections. Finally, this chapter is concluded with a summary of the theories explored.

5.1. Introduction to Software Development Tools

In software engineering, a software tool is a computer program used to enhance the engineering process by supporting the software process or the production of software deliverables (Hunt & Thomas, 1999). Since the inception of software engineering in the 1950's, tools have been used to develop, test, and maintain software products (Coram & Bohner, 2005). At the height of automation in the 1980's such tools became known as Computer Aided Software/System Engineering (CASE) tools. There are two sub-categories of CASE tools (Kelter et al., 2003), Lower-CASE and Upper-CASE tools. Lower-CASE tools are generally used by software engineers to edit and compile (and otherwise manipulate and manage) source code, whereas upper-CASE tools are used to support other (generally earlier) activities in the software development process, such as analysis and design, and process documentation.

In addition to CASE tools, software engineers also use tools to plan, monitor, and control the software development process (Fox & Spence, 2005). These tools are called project/process management tools. Through the use of these tools, engineers are often able to track critical processes in the software development life cycle. Additionally, the tracking of software cost, resource, and time estimations are often supported by such tools (Kelter et al., 2003). It is also common for the different categories of tools (upper-CASE, lower-CASE, and process) to be integrated into one tool environment.

Project management is often posited to be critical in software production (Fox & Spence, 2005). Even though software tools exist to assist software developers, software challenges over the years have frequently been linked to poor project management (Parker, 2002). While the use of software project management tools cannot eliminate software projects' risks, tools can be used to enhance project communication, and allow project members to plan, track, and control projects, thereby enhancing software project management, and thus reducing software risks (Jurgen, 2000). Similar views are also promoted by Meredith & Mantel (2003), who confirm that without project management tools project managers would not have unhindered access to accurate project information, which should inform decision making.

Thus, software developers have at their disposal a wide variety of commercial tools for selection. In addition, open source tools exist that may also aid project management. These tools support varying aspects of the software development process implemented through the different software development methodologies. The remaining sections of this chapter further examine software project management tools' importance, and present an evaluation of the popular tools that support the two software paradigms (conventional and agile) under consideration in this study.

5.2. Importance of Project Management Tools in Software Development

According to Fox (2002), regardless of the software development methodology employed, project management tools are an asset to the software development process. Globerson & Zwikael (2002) affirm that through the use of tools, project managers are often able to re-schedule and identify suitable project alternatives. Meredith & Mantel (2003) claim that project management tools may act as a repository for project information to enable project managers to learn from their previous mistakes. In respect to the topics addressed in this thesis, Fox & Spence (2005) and Swink (2002) confirm that tools may allow project managers the opportunity to identify deficiencies in team personnel and to utilise resources where they are most effectively needed.

Thus, the use of project management tools in the software process may reduce project risks. In addition to studies by Fox (2002) and Globerson & Zwikael (2002), a study by Fox & Spence (2005) shows that project managers who employ software tools may outperform those who do not employ such tools. Even though there are some controversies over these authors' assertion (see Standish Group (2001) for example), there exists little doubt that software tools offer positive support for project teams if they are appropriately used.

While there are several classes of software tools as described above (lower-CASE, upper-CASE, and process tools), a detailed analysis of the different classes is outside the scope of this research project. Rather, this study examines software tools associated with software project management. Such tools are often used in project planning, allowing the storage of planning information such as cost, time, and resource estimates (Kelter et al., 2003). Schedules are also documented, and the tools' functionality often models the project management techniques selected for development. The project management model acts as an outline for the processes that exist in the particular software project. Hence, during project execution, tasks are engineered according to the models selected for development. In addition, these tasks are assigned to team members, which are monitored and controlled via project management tools. In the following section an analysis of contemporary project management tools is provided.

5.3. Contemporary Project Management Tools

Tools that support conventional waterfall processes seem to be also structured and systematic in nature. Meredith & Mantel (2003) assert that the most integral issue in tool development for conventional waterfall projects is to have these tools act as a reservoir for information enabling project managers to keep project performance in concert with project plans. This coincides with the literature on the conventional waterfall models which stresses systematic planning and phased implementation of software projects. Fox & Spence (2005) assert that project managers who are not themselves structured face difficulties when adopting these tools. They find that a project manager who embraces a structured approach to decision making utilises project management tools most effectively. While this assertion bears some validity, an earlier study by Rowe & Mason (1987) showed that tools that are often overly structured may affect individual's success through their use. This seems to suggest that overly structured tools (can) also affect an individual's effectiveness in a negative way.

Several commercial project management tools exist that are suited to managing the processes prevalent in the conventional waterfall models. Microsoft Project seems to be the most widely used in this environment (Fox & Spence, 2005; Meredith & Mantel, 2003). However, even though Microsoft Project seems robust and offers several process management functionalities (see Table 5.1), considering the details of the evaluation summary presented in Table 5.1, it also possesses several limitations. The reason for Microsoft Project's shortcomings is because this tool supports the tracking of software design information in a stand alone environment. Meredith & Mantel (2003) affirm that Microsoft Project's rigidity may be as a consequence of its purpose. They say that this tool was developed with the underlying principles of most Microsoft products; to facilitate add-on capabilities.

While add-on functionality may be an asset to project management software, the cost for these systems is likely to rise as each module is added, and this affects project management budget. In addition Meredith & Mantel (2003) noted that systems that are configured in such a manner, may hinder effective project management due to information overload, project isolation, and excessive overhead using these tools. Further, Microsoft Project does not offer risk management support, and a significant degree of customisation may be required prior to project data population for process tracking (see Table 5.1). In consequence, risk management software such as Risk Guide may offer a possibility as an add-on for Microsoft Project (Morski & Miler, 2002).

Similar to Microsoft Project, AceProject, and Trackstudio Enterprise are also commercial project management tools. These tools offer process support similar to that offered by Microsoft Project. However, in addition to the support for management functions such as cost, time, and resource estimation, AceProject and Trackstudio allow for a more collaborative management environment, and provide support to track technical risks in the software life cycle (a summary of AceProject's features is included in Table 5.1). In addition to these tools, several similar open source tools exist that support the administration of conventional processes in software production. DotProject is one such tool (see www.dotProject.net for this and similar open source tools).

After examining and comparing conventional project management tools such as Microsoft Project and AceProject, the findings (see Table 5.1) suggest that these tools might not adapt well when used to monitor the processes supported by agile software development methodologies (see Chapter 2 for agile processes, and Table 5.1 for recommended feature support for agile tools). Among the features suggested for agile tools (Kelter et al., 2003; VersionOne, 2006a)**,** it is recommended that agile tools should allow both local and distributed data management, should be remotely accessible, should offer a range of reporting capabilities, should be secured, should offer a risk prioritisation scheme, and be easy to use (see Table 5.1 for further details).

Research has therefore been considering a more appropriate set of tools (Kaariainen et al., 2004; Maurer & Martel, 2002). Recent agile project management tools offer some improvements over conventional tools such as Microsoft Project. However, an evaluation of the popular project management tools (Behrens, 2006) shows that it might not be possible for a single project management tool to be a universal remedy. Agile project management tools are further examined in the following section.

Table 5.1. Evaluation summary of the features existing in popular project management tools, adapted from (Kelter et al., 2003; VersionOne, 2006a)

Feature/Tool Support	**ExtremePlanner**	**VersionOne**	**DevPlanner**	**Rally**	**AceProject**	**MS Project**
Local Installation or Distributed Management for teams of any size	Yes	Yes	LI	HO	Yes	NC
Remote Accessibility	NC	Yes	No	Yes	Yes	No
Reporting and Analytics						
Executive level reporting	No	Yes	No	Yes	Yes	Yes
Project/ Releases reporting	Yes	Yes	Yes	Yes	Yes	NC
Shared Projects reporting	Yes	Yes	No	Yes	Yes	NC
Individual level reporting	No	Yes	No	Yes	Yes	NC
Feature management and planning	Yes	Yes	Yes	Yes	Yes	Yes
Customer request management	Yes	Yes	No	Yes	Yes	NC
Feature estimation and prioritisation	Yes	Yes	Yes	Yes	Yes	NC
Defects estimation and prioritisation	No	Yes	Yes	Yes	Yes	NC
Feature tracking	Yes	Yes	Yes	Yes	Yes	NC
Individual feature tracking and estimation	No	Yes	No	Yes	Yes	No
Password authentication, new team member addition, role management	Yes	Yes	No	Yes	Yes	No
Audit history	Yes	Some	No	No	Yes	No
Project information archiving	Yes	Yes	Yes	Yes	Yes	Yes
Feature Risk rating scheme	Yes	Yes	No	Yes	No	No
Feature Estimation and implementation summary	Basic	Yes	EO	Yes	Yes	Yes
Easy to use, short learning curve	RE	RE	Yes	RE	RE	RE
Risk Mitigating Feature						
Personnel Capability Management	No	No	No	No	No	No
Customer interface to extend the on-site customer	No	No	No	No	No	No

Key: LI - Local Installation, HO - Hosted Onsite, NC – Need Configuration, EO – Estimation Only, RE – Relatively Easy.

5.4. Agile Project Management Tools

It is often emphasised that tools designed in support of agile processes should be tailored for the specific needs of the development team (Kelter et al., 2003; Chin, 2004). However, agile supporters constantly stress that agile methods can be successfully applied exclusive of supporting tools (see Highsmith (2000, 2004) for example). While this assertion sounds plausible, there is little doubt that if properly applied, relevant project management tools may add value to the agile development environment.

Several reasons exist that necessitate tool support for agile software development; larger projects are often said to induce complexity, and managing complexities is seldom achieved without support tools (De Souza et al., 2005). In larger projects, process data and team activities are often managed through project management tools (Kelter et al., 2003). In addition, the use of tools offers a means of quantifying development practices (although its value depends on the usefulness of the measures), while allowing traceability (Behrens, 2006). Furthermore, tool support may become essential for distributed teams (Angioni et al., 2006). Tools supporting the processes recommended by agile methodologies fall into three categories (Highsmith, 2004): collaboration, technical information sharing, and project management. Since the main objective of this research is to verify whether a prototype can be developed for agile teams to help them with their handling of social risks, it is necessary to consider existing agile tools, and how these tools are used to support agile teams. Therefore, in the following section, discussions on agile collaboration, technical information sharing, and project management tools are presented.

5.4.1. Collaboration, Technical Information Sharing, and Project Management Tools

Collaboration tools facilitate teams working in a distributed environment, bringing teams together as if they were occupying a common area (Leuf & Cunningham, 2001). These tools are commonly grouped under the computer supported cooperative work (CSCW) label; they provide facilities for email, discussion groups, conferencing, and instant messaging. Email, discussion groups, and instant messaging primarily enable text-based interactions among stakeholders. These tools can also be considered as interactive content access tools. Such tools allow users to share files and other material.

Similarly, conferencing software also supports distributed groups by allowing teams to operate in a virtual business environment (Damian, Lanubile, & Mallardo, 2006). Conferencing tools include video conferencing, teleconferencing, and text-based conferencing. Video conferencing allows groups to maintain a two way visual and audio communication link, which permits users to simulate a co-located setting, while teleconferencing allows similar functionality but is often limited to only audio exchanges (Olsen, 2006). Like asynchronous communication such as email, text-based conferencing tools facilitate group member cooperation to convey planning information via files and shared whiteboard. There also exists synchronous text-based conferencing which is integrated with chat and whiteboard systems (see Linebarger, Scholand, Ehlen, & Procopio (2005) for example).

Technical information sharing tools are often used to manage architectural and other design artifacts in a distributed environment (Capilla, Nava, Perez, & Duenas, 2006). The Wiki is also becoming popular in CSCW. However recent Wikis are often incorporated into integrated development environments (IDEs) to facilitate process tracking functionalities. As a consequence, there is some disagreement over whether the Wiki is a collaboration or project management tool (Behrens, 2006). Accordingly, Behrens (2006) finds that the Wiki is often perceived as a project management tool, and it is used to manage project data, while also allowing collaboration.

The Agile Manifesto's first characteristic "Individual and interaction over processes and tools" (http://www.AgileAlliance.org) makes clear their stance regarding prescription for tools. Therefore, in the agile software project environment the expectation is for team members to favor communicating in an open work space (employing face-to-face practices). Mixed evidence exists with regard to the effectiveness of face-to-face communication (McKinney & Denton, 2005; Wellington et al., 2005). However, cultural and personality differences aside, this mode of communication offers a medium which helps in the reduction of information omission and ambiguity. Nonetheless, empirical studies verifying agile project management tools' adoption provide evidence to show that agile developers often rely on tools to manage their agile processes (Behrens, 2006).

Behrens (2006) reported several reasons for the use of tools to support agile teams; among them efficiency, productivity, and traceability are emphasised. On the other hand, the experiences of Williams (2003) show that teams employing agile methods use *ad hoc* processes to track project artifacts. Williams (2003) noted that in these teams, team members often employed pen-paper practices, and recorded system requirements on white boards and in Microsoft Excel files to track software artifacts and minimise rework. There are criticisms against these practices. Apart from poor traceability, Williams (2003) contended that these techniques sometimes resulted in agile teams losing important information.

Therefore, to ensure efficiency, productivity, and traceability, several project management tools exist to track agile processes (see Table 5.1). Due to the dynamic nature of agile-based development work, highly structured tools such as Microsoft Project and DevPlanner (see Microsoft Project and DevPlanner features in Table 5.1) might pose some problems if they are adopted in agile projects. Agile principles emphasise unlimited accessibility, suitable process match, and shared knowledge. For these and other reasons, tools such as Microsoft Project and DevPlanner may challenge (and constrain) the practices of agile developers due to their limitations (see Table 5.1 for further details). In addition since agility hinges on time in terms of rapid and frequent releases, the time spent by agile team members on learning to use complex tools reduces agility and productivity in agile projects. Therefore, it is recommended that tools developed to support agile project management should also be easy to use, and should not slow down the software development process. Koch (2005) says that usability issues should be non existent in agile tools.

Correspondingly, academic research is continually examining ways of supporting agile methods' best practices in toolsets. Maurer & Martel (2002) created MILOS to support agile software teams' communication; the main aim behind the development of this tool is to enhance coordination among agile team members. MILOS allows agile teams to manage project requirements. Similar tools called XPSWiki, DotStories, and XPWiki were created by Angioni et al. (2006), Ceravolo et al. (2003) and Rees (2002) respectively. A further alternative, StoryManager, was created by Kaariainen et al. (2004); this tool employs the Eclipse integration framework. Eclipse is a development environment and a tool integration framework, allowing for sharing and managing of agile project requirements. When StoryManager was evaluated in a small live XP project environment, this tool was abandoned after two releases and the testing team reverted to their manual process. According to Kaariainen et al. (2004), usability issues were the main reason for StoryManager's discontinued usage.

Atsuta & Matsuura (2004) also developed a tool that supports the XP pair programming practice. This tool allows pairs of programmers to work in a distributed environment. Their tool offers several elements of XP pair programming support: role shift - where pair programmers shift repeatedly between roles of reviewer and programmer; notification of state - programmers can inform each other of their state, whether they are working or not; chat - programmers can have electronic conversations; and white board - this function allow for the sharing of ideas. This tool is limited to only one XP practice, pair programming.

The literature above shows that several tools exist to assist agile teams with collaboration and information sharing among developers. These tools may also allow the customer to be involved with the development team. However, tools developed for project management do not consider the customer; even though the customer is expected to be regularly involved with project management in agile development teams. This seems to suggest that studies should investigate how project management tools can be built to address customer involvement with the developers.

A survey done by Behrens (2006) found that ExtremePlanner, VersionOne, and Rally are popular commercial tools which support agile processes. Even though there seems to have been no specific attempt to have risk mitigation features built into ExtremePlanner, VersionOne, or Rally, these tools offer several features which are rare in most other process tools (see Table 5.1), and their use may also possibly reduce agile project risks. However, since these tools were developed mainly for traceability and tracking purposes, they neglect the most critical dimension of risks - social risks (Curtis, 1989; Curtis et al., 1988). Empirical evidence exists to show that team cohesion is most critical to project success, and personality differences pose significant threats to software projects' achievements (Jones, 1999; Verner & Evanco, 2005). With agile methods emphasising "individual and interactions over process and tools" (see Chapter 2 for the other agile tenets), social risks are likely to be prevalent in projects where agile (social) practices are used.

Therefore, a tool considering social risks, built to extend the remote customer, and offering team personality assessment and management prior to team formation (see Chapter 4 for further details) would potentially lessen agile project risks through its use in software projects. There is an expectation that such a tool would offer significant support to the agile community. Thus, Agile Social-Risk Mitigation Tool (ASRMT) was built to address the shortcomings of current agile project management tools by offering an extension for remote customer feature management, and personality assessment functionalities. An introduction to ASRMT is presented in Chapter 6.

5.5. Summary

Since the inception of software engineering, software tools have provided useful support for software developers. Though a wide variety of tools exist, all tools might not be appropriate for every software environment. Tools such as Microsoft Project and AceProject may be appropriately applied in a plan driven development environment, whereas tools like VersionOne and Rally may offer better support for agile project environments. Collaboration tools such as email, discussion groups, conferencing, and instant messaging also add value to agile distributed teams by allowing data sharing and bringing teams together as if they are occupying a common space.

Tools such as ExtremePlanner, VersionOne, and Rally offer a wide range of features, and if used by agile teams, these tools may lessen risks in agile projects. However, these tools offer no mechanism for assisting teams to plan for social risks; even though agile practices may attract social risks, and such risks are deemed the most critical encumbrance to project success. ASRMT is a tool built addressing the shortcomings of popular agile tools by offering agile teams' social risk mitigation features. An introduction of ASRMT is provided in the following chapter.

6. Design and Implementation of the ASRMT Tool

This chapter describes the development and implementation of a tool called Agile Social-Risk Mitigation Tool (ASRMT), designed to assist agile project teams with their handling of social risks. In this chapter, ASRMT is first introduced, providing details of its rationale, the approach taken to development, and implementation details. The features of ASRMT are then described, and then the chapter concludes with a summary of the discussions throughout this chapter.

6.1. Introduction to the Agile Social-Risk Mitigation Tool (ASRMT)

In this section the ASRMT research project goals are outlined, and then the software development methodology selected for implementing ASRMT is described. Third, the ASRMT development platform is outlined and justified. Finally, an overview of ASRMT is provided.

6.1.1. Research Project Goals

While the processes recommended by agile software development methods may improve some aspects of the software production environment, Chapter 4 shows that these processes also inherit some challenges of their own. In addition, some agile software development methods are recent, and arguments suggesting that they are untested (see Coram & Bohner (2005) for example) bear some soundness. Thus, research examining ways to strengthen the processes employed by agile methods carries potential significance. The literature examining agile processes and the challenges inherent in their implementation seems to be mostly descriptive, and empirical evidence is sparse (though there are a few experience reports in existence).

Project management tools may improve the agile software development environment and reduce risks in these projects. In addition to support for information management and communication among team members, agile project management tools may also implement other risk mitigation strategies. However, since agile methods place significant emphasis on face-to-face communication, arguments *against* using a project management tool that has been implemented for reasons such as to assist with communication are subtly valid, as it may work against the very principles embedded in agile methods. Nonetheless, the literature shows that significant risks are introduced through human collaboration and that personality differences are magnified in co-located teams (Grisham & Perry, 2005; Nerur et al., 2005).

Since behavioral risks are deemed most significant in software projects (Verner & Evanco, 2005), a tool developed on the basis of social risk mitigation theories possibly will lessen risks through its use in agile projects and present some concrete guidance for the practices recommended by agile methods. To be effective, such a tool should present low overhead in its usage, and should consider aspects particular to agile software development environments. Though there exist some agile project management tools, these have not considered the employment of 'social risk' mitigation strategies (see the tool evaluation in Table 5.1). This research project aims at implementing risk mitigation strategies to directly address, and hopefully reduce, exposure to the social risks that may become inherent in agile projects.

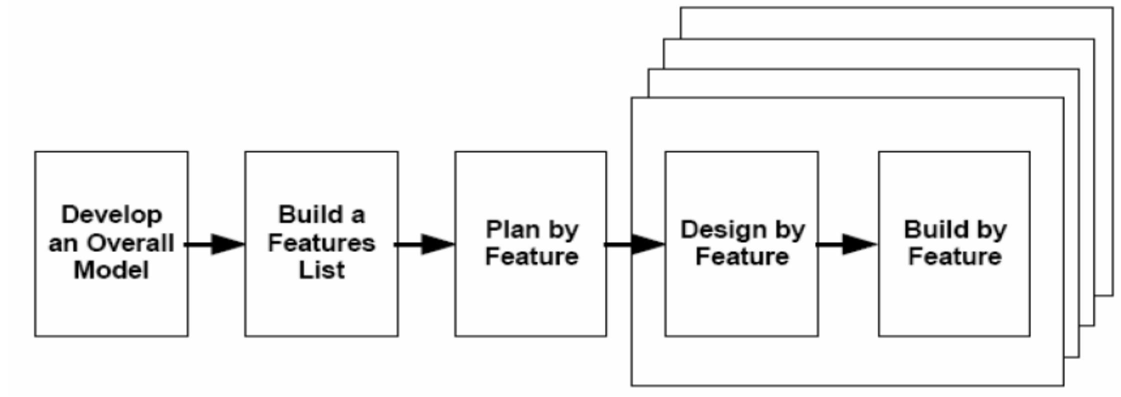
In this context, this tool addresses the extension of customer involvement, the provision of support for remote customers, and the management of team members' personality capability. This reflects the suggestions made by this study in terms of reducing customers' direct involvement with the development team (see Chapter 4 section 4.1.4). Additionally, personality assessment prior to team formation should allow agile project managers to better manage team members, and team assemblages, potentially reducing risks associated with personality conflicts among agile teams. Therefore, the tool under consideration also implements the Belbin's Self-perception Inventory (SPI) to support developers’ personality assessment prior to team formation (see Chapter 4 sections 4.1.2 and 4.1.3 for risks associated with personality conflicts).

6.1.2. Software Development Methodology Used for ASRMT

As described in previous chapters (mainly Chapters 2 and 3), agile software development methodologies are often associated with fast paced software development of innovative products. Given the time and resource constraints of this research project, the principles and practices underlining such an approach (see Chapter 2 section 2.3.2 for further details) seem to be most suited to this project development framework. In addition, by adopting such a methodology, the researcher stands to achieve knowledge which potentially expands their understanding of some of the implications regarding agile software development. Further, in recognition of prior discussions, it is posited that the processes employed by sensible project managers should fit with the characteristics of the software product under consideration, thereby increasing the likelihood of development success (Highsmith, 2004). Therefore, the ASRMT tool is created under the principles and practices that adhere to agile software development methods and agile project management.

Correspondingly, agile’s four favored principles - responding to change, working product, customer collaboration, and individual and interaction - influence how the practices selected throughout the software development life cycle are implemented. While the process of envisioning and exploring is implemented in the design and build phases of this software project, domain object modeling (DOM) is implemented in the project specification phase. According to Koch (2005), domain object modeling is the Feature-Driven Development (FDD) method of capturing the overall design of the system to enhance developers’ understanding of the domain (see Figure 6.1 for the FDD process diagram). Thus, capturing this software project in a DOM should help to simplify the development process. Additionally, FDD’s practices such as developing by feature, regular build schedule, configuration management, and reporting and visibility of result are implemented throughout this software project.

Figure 6.1. FDD process diagram (De Luca, 2002)



Software development projects leading to large mission critical systems can dedicate a significant amount of time towards requirements gathering, which is normally followed by a linear implementation of the software development life cycle (Boehm & Turner, 2003). This is often appropriate for such a context. However, for rapid application development (RAD), working software is preferred over a waterfall mode of development (Agarwal et al., 2000). Thus, while time is invested in requirements specification for ASRMT using the preferred DOM approach, this software is developed feature by feature. FDD defines features to be small "client-valued" functions which are implemented in a maximum of two weeks (Koch, 2005). This practice is said to enhance dependable feedback among stakeholders in the development process, which increases the likelihood of tangible delivery and progress throughout the software development life cycle. Abrahamsson et al. (2003) recommend the use of the FDD method in 'time and resource' constrained development environments where quality is necessary.

Regular build schedules and configuration management (CM) are employed throughout this software development project. After classes are developed to support proposed feature functionalities, unit testing is performed separately on each class. Subsequently, the code existing in the classes is checked thoroughly before being promoted for building. Regular builds are done continuously as features are implemented. This technique helps with ensuring an up to date version of the system exists at any given time of development review (Highsmith, 2004; Koch, 2005), whereas unit testing should enhance software quality (De Luca, 2002). Configuration management is required in scenarios where regular build practices are engaged. This technique is said to assist with efficient management of the artifacts. In this case, SourceSafe is utilised to support the implementation of configuration management methods in this project.

It should be noted, however, that this project environment does not precisely mirror a real life software project environment where stakeholders occupy distinct roles such as customers, developers, project managers, quality assurance engineers and so on (see Highsmith (2004) for agile project roles). Since this project only possesses one primary participant, the researcher must occupy the roles previously mentioned throughout the software project, at times simultaneously. The supervisors of this project offer some customer-style feedback to ensure a reliable prototype is developed, meeting time and resource estimates, whereas the additional customer, the developer, project manager, and quality assurance roles are occupied interchangeably by the researcher. That said, principles such as customer collaboration and individual and interaction are implemented using practices such as specific reporting techniques, including project status reporting, and visibility of results recommended in FDD projects (see Koch (2005) for examples).

Project status reporting to provide visibility of results is implemented throughout this software project, by examining the ratio of completion of the feature list and feature development milestones. This technique is used to enhance continuous project milestone estimation throughout the project in order to keep track of project progress. See Table 6.1 for a full summary of the chosen FDD practices from the practices recommended for software development using this method.

Table 6.1. Chosen FDD practices for ASRMT implementation (De Luca, 2002; Koch, 2005)

Process	**Description**	**Reason for Selection**
Domain Object Modelling (DOM)	An overall model created to capture the domain after project scoping.	To provide the project scope, to allow a detail assessment of project goals prior feature list building.
Development by Feature	After decomposing the DOM into subject areas, and business activities, a feature list is a representation of each category identified.	To identify small units for which to start designing, to allow planning and development in increments, to allow for rapid feedback between developer and supervisors.
Planning by Feature	A plan of the order in which features should be implemented.	To identify and evaluate class feature dependencies, to assist with load balancing throughout the software project, to help to simplify complex features, to identify and plan for risks which may result through feature complexity.
Designing by Feature	A package describing the design details of the feature, design alternatives, task list, and sequence diagram of the feature.	To verify the dependencies among classes identified in the plan, to verify sequence of events, and schedule, to identify dependencies among features.
Building by Feature	An activity to produce a client-valued function.	To produce fast-pace increments, to produce quality software, by implementing practices such as unit testing and code inspection in each feature built, to verify quality output, and produce software on schedule, to ensure quality through rapid feedback.
Configuration Management	Management of artefacts, and source code versions.	To enhance management of source code versions.
Reporting and visibility of results	Mechanism for tracking and reporting the project status.	To ensure rapid application development with proper standards, these include usability, relevance of feature, and adequate documentation.

6.1.3. Development Platform

Given the research goals described above, this prototype tool is by necessity developed to support a distributed environment. While a regular desktop tool may assist co-located teams using agile processes, thereby extending the scope of agile methods and reducing their risks (see Chapter 5 section 5.2 for a list of the importance of software tools), web based tools are most suitable for agile distributed environments (Angioni et al., 2006). This is because most agile methods stress continuous accessibility to the customer, emphasising communication and cooperation, and sometimes distributed data sharing (Kelter et al., 2003); such goals are easily facilitated in web settings. Additionally, support for distributed teams requiring remote customers' interfaces are not facilitated in a non-web situation. Technologies such as teleconferencing software and e-mail systems are examples of such milieu which provide support for distributed communication.

Several choices exist for web application development. Among these, Java technologies and the Visual Studio.NET seem most popular for such implementations (Atsuta & Matsuura, 2004; Kaariainen et al., 2004). To verify whether the concepts proposed in Chapters 4 and 5 are achievable, this tool is developed using the ASP.NET framework. ASP.NET is a suite of programs which support fast paced platform independent web application development (Moroney & MacDonald, 2006). Hence, the concepts of standard user application techniques such as storing, retrieving, displaying, and modifying data are implemented in this software development project, using a client server architecture.

Of the visual studio collection of programming languages, Visual Basic is used for implementing server side logic. The ASP.NET framework allows for basic user validation, therefore, supplemental validation can be performed using JavaScript. Rapid application development (RAD) is the fundamental trait of this project, thus the assertion made by (Flanagan, 2001), who asserts that JavaScript presents a short learning curve to web developers and enhances efficiency of web application by allowing client side validation, is embraced. Microsoft Windows XP Professional operating system is the platform for development, and Internet Information Services (IIS) is used as the web hosting software for this application, given their dominance in desktop environments.

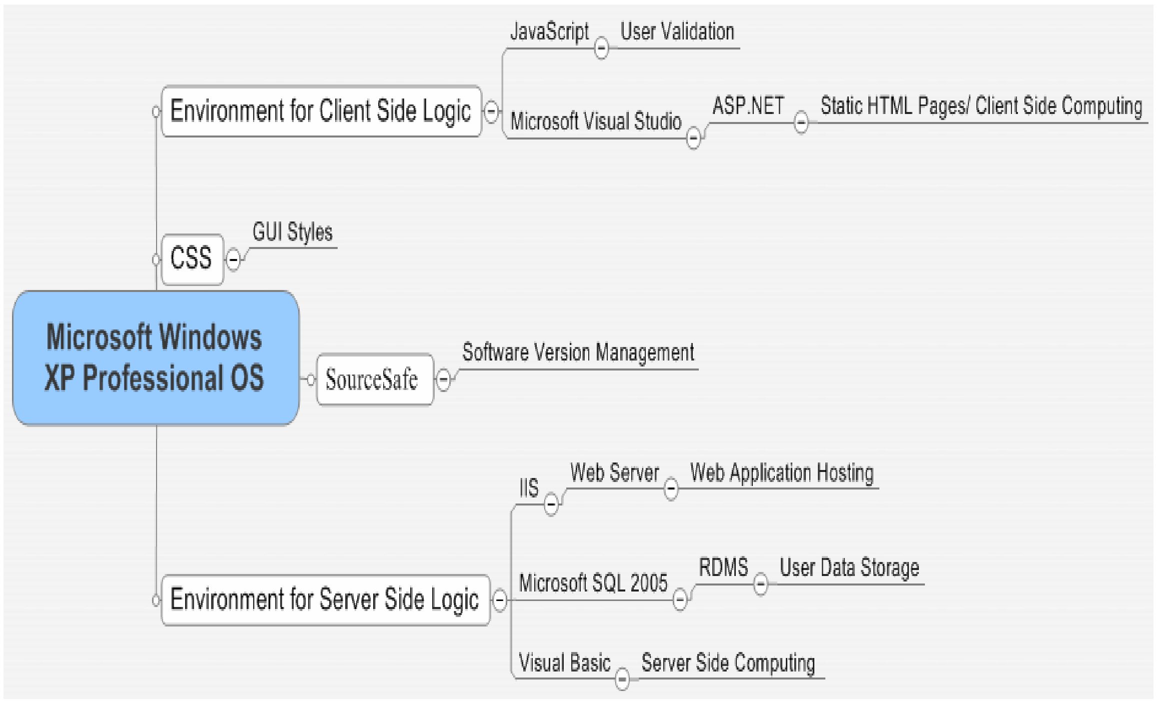
Additionally, data storage is enabled using the Microsoft SQL 2005 relational database management system (RDMS). Client server computing functionalities such as views and stored procedures are also supported by this RDMS. As mentioned previously, SourceSafe is used for managing source code versions. Finally, cascading style sheets (CSS) are used for formatting the web pages. CSS is a technology which allows web developers the opportunity to build web pages with a consistent look and feel, while allowing a separation of style from content (Clark, 1998). See Figure 6.2 for a graphical representation of the proposed development architecture.

As mentioned previously, an alternative to this implementation platform would be to use Java technologies. Java Server Pages (JSPs) could be used to implement the server side logic of the application (Bergsten, 2004). This technology is also platform independent and allows rapid application development (Bergsten, 2004). Additionally, an Apache web server could be used to implement server side programming. In this scenario, JavaScript could also be used for client side validation and CSS for consistent look and feel, while any RDMS can be used to store user data.

Even though this latter option is open source and may appear attractive, there are several drawbacks to this implementation. Configuring IIS to work with Apache has proven tedious because they are from differing vendors (Warrene, 2004). JSPs are only served in a JSP engine; thus, the functioning of this web application will necessitate configuring a JSP engine such as Resin or Tomcat in order that the JSP pages are served. Connection to the RDMS would require a specific database driver such as aveconnect or ODBC. Finally, optimum performance is not guaranteed with differing vendors applications, for example, connecting to a Microsoft database with JDBC drivers (Clark, 2006).

Since the platform of implementation is (almost) exclusively Microsoft, ease of configuration and development should be enhanced through utilising the technologies selected for this tool development (see Appendix A for the ASRMT configuration and deployment instructions). Finding such an approach to software development is constantly stressed by agile proponents such as Jim Highsmith, Kent Beck, and Alistair Cockburn (see Highsmith & Cockburn (2001) for example). A summary of the experiences gained through adopting this approach is presented in Chapter 7.

Figure 6.2. Development Architecture for ASRMT



6.1.4. ASRMT Overview

Since software development is plagued by many challenges which make it risky, the identification and validation of new approaches that improve software development practice is of paramount importance to the future success of software development. While agile models are said to provide practices that are likely to reduce software risks, agile social practices are also risky (see Chapter 4). In addition, social risks are most critical to software development success. In light of this, this research leverages risk theories to develop a prototype to support the social side of agile software development.

The intent of ASRMT is to assist agile project teams with their handling of social risks. This tool specifically takes into consideration the social practices (co-located teams depending on tacit knowledge, and onsite customer) of agile methodologies. Thus, ASRMT is aimed at reducing social risks in agile teams by allowing project managers the opportunity to verify individuals' personality preferences and traits (to unearth their team strengths and weaknesses, so as to place them in most suitable roles) *before* they are assigned to software project roles, and integrated into the software team. Additionally, ASRMT is also meant to extend remote project customers (to reduce conflicts between onsite customers and the development team). Support for remote customers is achieved through ASRMT client feature request management functionality. This functionality allows project customers the chance to participate in the management of features, irrespective of their presence, on – or off-site.

While the primary aim of ASRMT is to assist development teams with the handling of social risks, this tool is also likely to offer several other benefits to software project teams. ASRMT is a (small-scale) project management tool (see section 6.2 for ASRMT project management features) and it provides development teams with the ability to communicate project information across teams (in support of both distributed and co-located teams). This tool also enables project teams to track software projects, and it acts as a reservoir for information in order to enhance project managers' future decision making. The following section provides additional details about the features existing in ASRMT.

6.2. ASRMT Features

In this section an introduction to the features provided by ASRMT is presented. This section is arranged in the following order: firstly, the list of features in ASRMT is provided, and then an introduction to ASRMT follows in the 'Using ASRMT' section. Finally, ASRMT user options are introduced. See Appendix B for the ASRMT requirements specification information.

6.2.1. List of Features in ASRMT

ASRMT was developed to support the management of agile project teams' remote customer and team members' personality preferences. Accordingly, this tool offers the features discussed below (note that 'client' and 'customer' are used interchangeably here to denote the user who is actively participating and involved in requirement specification, software testing, and other development practices often recommended by agile methods).

Client Feature Management

Requirements specification, a standard and ongoing activity in agile software development (see Chapter 2 for an example), can be tracked using ASRMT. Feature requests are initiated by the customer; features are then jointly managed by the development team and the customer. Customers are involved in the management of features for all projects in which they are involved.

Developer Feature Management

While customers are often the initiator of features, developers are responsible for coordinating activities which should lead to feature development and overall software project realisation. ASRMT offers developers the opportunity to monitor development goals in the form of features (entered by Clients), to allow customers the opportunity to keep track of software development progress. Developers are assigned to designated projects, and are responsible for updating ASRMT with project information such as feature technical risks, and their development progress.

Remote Accessibility for Remote Developer and Client

Remote accessibility is integral to the work of distributed teams. ASRMT is web based and offers remote customers the opportunity to interface with local project teams. Additionally, developers can also access ASRMT from anywhere and at anytime to ensure up to date project information and feature tracking. Remote developers and customers are supported by ASRMT through a web browser.

Feature Tracking

Feature information initiated by the customer is jointly managed by both the developers and the customer in ASRMT. ASRMT offers a range of project summaries for feature information (for example: New Feature, Defect, and Enhancement tracking). Through adequate user updates, ASRMT is able to offer real time feature tracking information support for Clients and Developers, in both detailed and summary forms. This information allows customers the opportunity to see developers’ progress while being remote. In addition, project managers are allowed a chance to make expert decisions (related to project management) through the information provided by ASRMT.

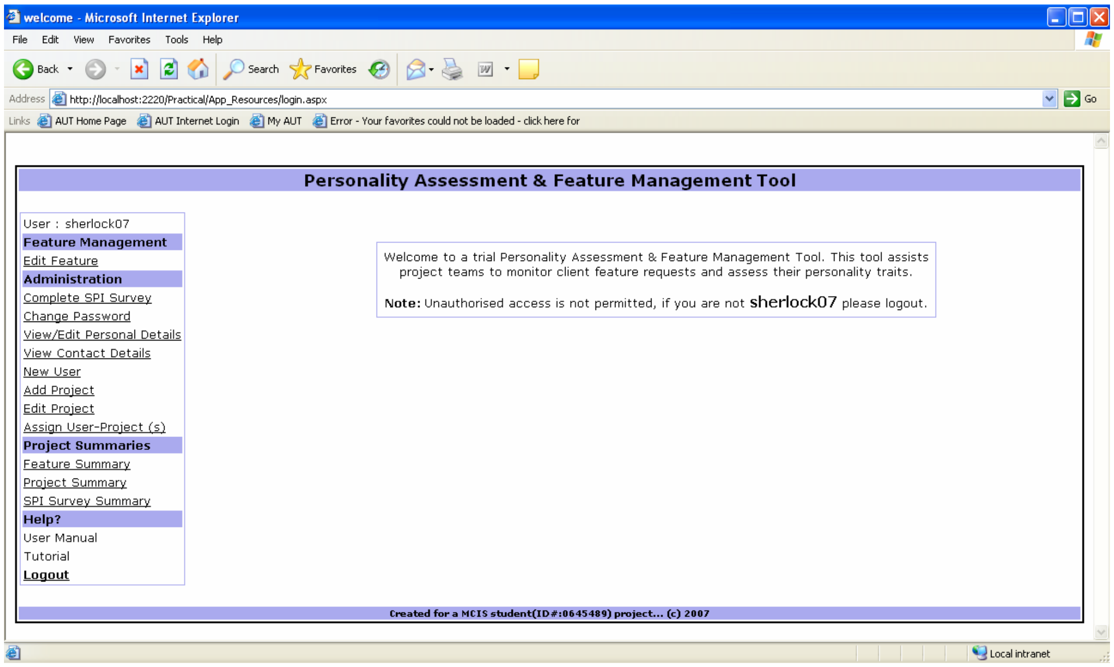
Personality Capability Management

ASRMT allows team members the opportunity to determine their two main personality preferences, in this case using the Belbin Self-perception Inventory. Information regarding individuals’ personality preference is stored to assist with project management decision making. Through assessment summaries, project managers are afforded the opportunity to assemble a balanced team (with the right mix of personalities), taking into account team members' major strengths and weaknesses. In addition, through continued observation, project managers can also compare actual performance against the summary provided by ASRMT to make expert judgment regarding individuals' effectiveness in their given roles.

6.2.2. Using ASRMT

ASRMT takes into account the practices of agile software development teams. Since agile teams are meant to be flexible, ASRMT can be used flexibly in order that agility is maintained. All menu items are placed on ASRMT's main screen to reduce mouse clicks and repetition. The following menu items, shown in Figure 6.3, are provided by ASRMT.

Figure 6.3. ASRMT Main Menu


welcome - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2220/Practical/App_Resources/login.aspx
Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for
Local intranet

Personality Assessment & Feature Management Tool

User : sherlock07

Feature Management
Edit Feature

Administration
Complete SPI Survey
Change Password
View/Edit Personal Details
View Contact Details
New User
Add Project
Edit Project
Assign User-Project (s)

Project Summaries
Feature Summary
Project Summary
SPI Survey Summary

Help?
User Manual
Tutorial
Logout

Welcome to a trial Personality Assessment & Feature Management Tool. This tool assists project teams to monitor client feature requests and assess their personality traits.

Note: Unauthorised access is not permitted, if you are not sherlock07 please logout.

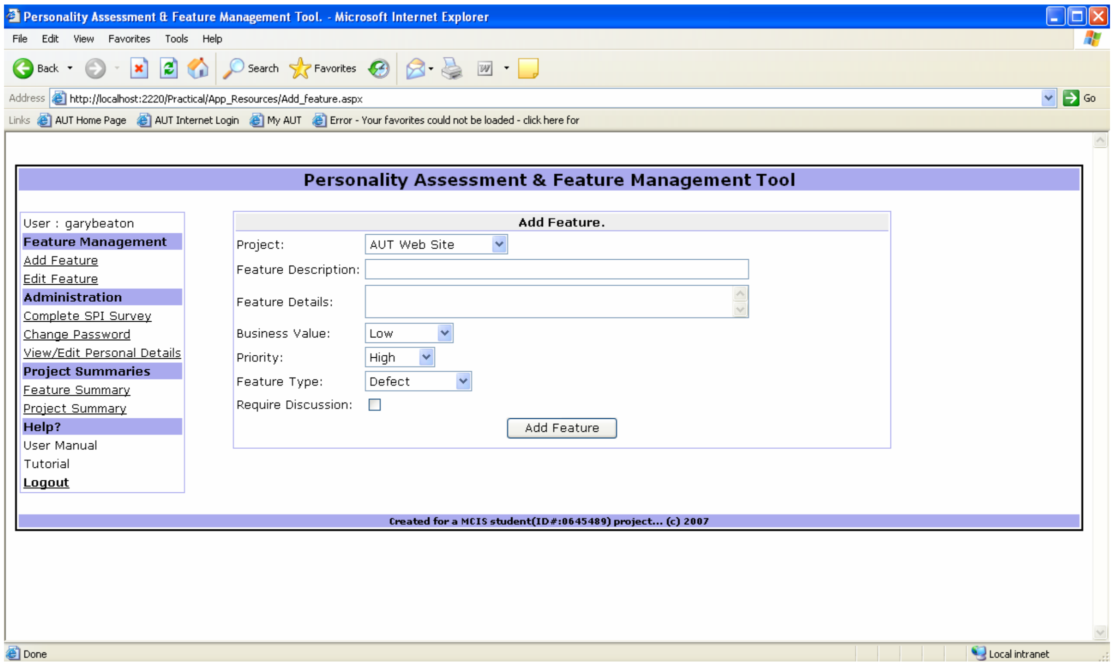
Created for a MCIS student(ID#:0645489) project... (c) 2007

Feature Management

Features are small client valued system functions. Thus, features can be classified as system functionalities. ASRMT allows customers to add features, and all users are allowed to edit features.

Add Feature: ASRMT allows clients to add features (see Figure 6.4). In order for clients to add features, they must provide feature information such as the Project, Feature Description (short statement of not more than 200 words), Feature Details (any additional details regarding each feature), Business Value (Low, Moderate, or Significant – features that are Significant being most important for the business at the time of entry), Priority (High, Medium, or Low – features that are high in priority should attract most of the developers' attention), Feature Type (New Feature, Defect, or Enhancement), and whether the feature should be discussed in a face-to-face meeting. Features added by clients are tagged with the default status 'Requested'.

Figure 6.4. Add Feature Interface


Personality Assessment & Feature Management Tool. - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2220/Practical/App_Resources/Add_feature.aspx
Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for
Done
Local intranet

Personality Assessment & Feature Management Tool

User : garybeaton
Feature Management
Add Feature
Edit Feature
Administration
Complete SPI Survey
Change Password
View/Edit Personal Details
Project Summaries
Feature Summary
Project Summary
Help?
User Manual
Tutorial
Logout

Add Feature.

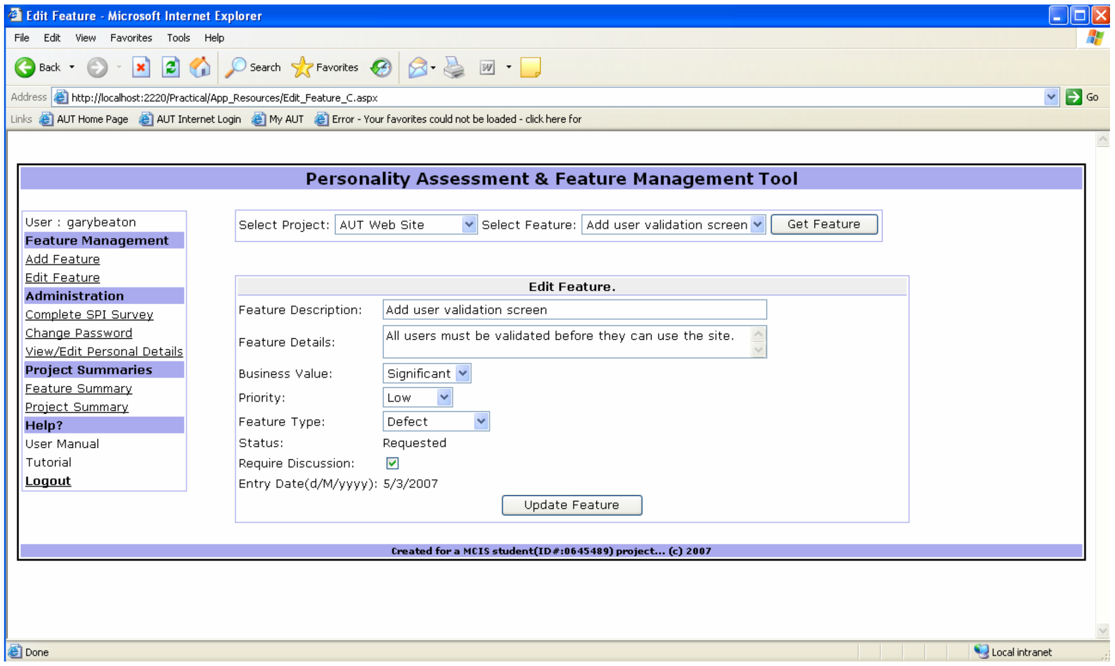
Project:	AUT Web Site
Feature Description:	
Feature Details:	
Business Value:	Low
Priority:	High
Feature Type:	Defect
Require Discussion:	☐

Add Feature

Created for a MCIS student(ID#:0645489) project... (c) 2007

Edit Feature: Clients are allowed to edit information for features previously added when the feature carries the status Requested, Estimated, or Scheduled (see Figure 6.5). If a feature Status is Estimated or Scheduled and it is edited by a Client, then its Status is automatically changed to Requested again. A log is maintained for all changes made to features by clients (see Figure 6.6). For Developers, features are edited and their Status updated automatically. Features traverse states via the Status field (see Figure 6.7). Initially when a feature is entered by a Client its Status is 'Requested' by default. When the Developer is ready to estimate a feature and the developer enters estimation information (Date Estimated and Estimated Hours), the feature Status is automatically changed to Estimated. If the feature is then scheduled (once Date Scheduled is entered), its Status is automatically changed to Scheduled. Once the feature Date Started field is updated, the feature Status is then automatically updated to In Progress. Finally, the feature status is automatically changed to Completed once the Actual Hours and Date Completed are entered. Developers are also able to categorise features based on an estimation of their technical risk (Low, Medium, or High). As mentioned previously, if the developer updates a feature Status to Estimated or Scheduled, and the client makes changes to the feature thereafter, the feature Status is automatically changed to Requested again (see Figure 6.8 for an example of the feature traversing states).

Figure 6.5. Edit Feature Interface (Client)


Edit Feature - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2220/Practical/App_Resources/Edit_Feature_C.aspx
Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for
Feature Management
Add Feature
Edit Feature
Administration
Complete SPI Survey
Change Password
View/Edit Personal Details
Project Summaries
Feature Summary
Project Summary
Help?
User Manual
Tutorial
Logout
Get Feature
Update Feature
Done
Local intranet

Personality Assessment & Feature Management Tool

User : garybeaton

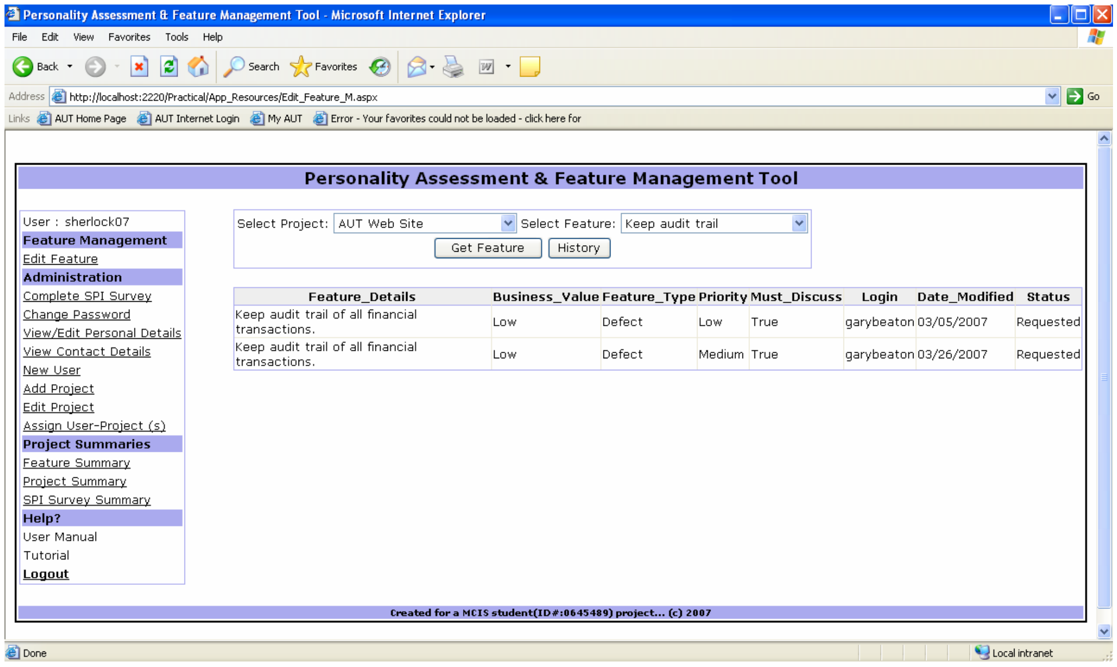
Select Project: AUT Web Site Select Feature: Add user validation screen

Edit Feature.

Feature Description:	Add user validation screen
Feature Details:	All users must be validated before they can use the site.
Business Value:	Significant
Priority:	Low
Feature Type:	Defect
Status:	Requested
Require Discussion:	✓
Entry Date(d/M/yyyy):	5/3/2007

Created for a MCIS student(ID#:0645489) project... (c) 2007

Figure 6.6. Feature Change Log


Personality Assessment & Feature Management Tool - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2220/Practical/App_Resources/Edit_Feature_M.aspx
Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for
Feature Management
Edit Feature
Administration
Complete SPI Survey
Change Password
View/Edit Personal Details
View Contact Details
New User
Add Project
Edit Project
Assign User-Project (s)
Project Summaries
Feature Summary
Project Summary
SPI Survey Summary
Help?
User Manual
Tutorial
Logout
Get Feature
History
Done
Local intranet

Personality Assessment & Feature Management Tool

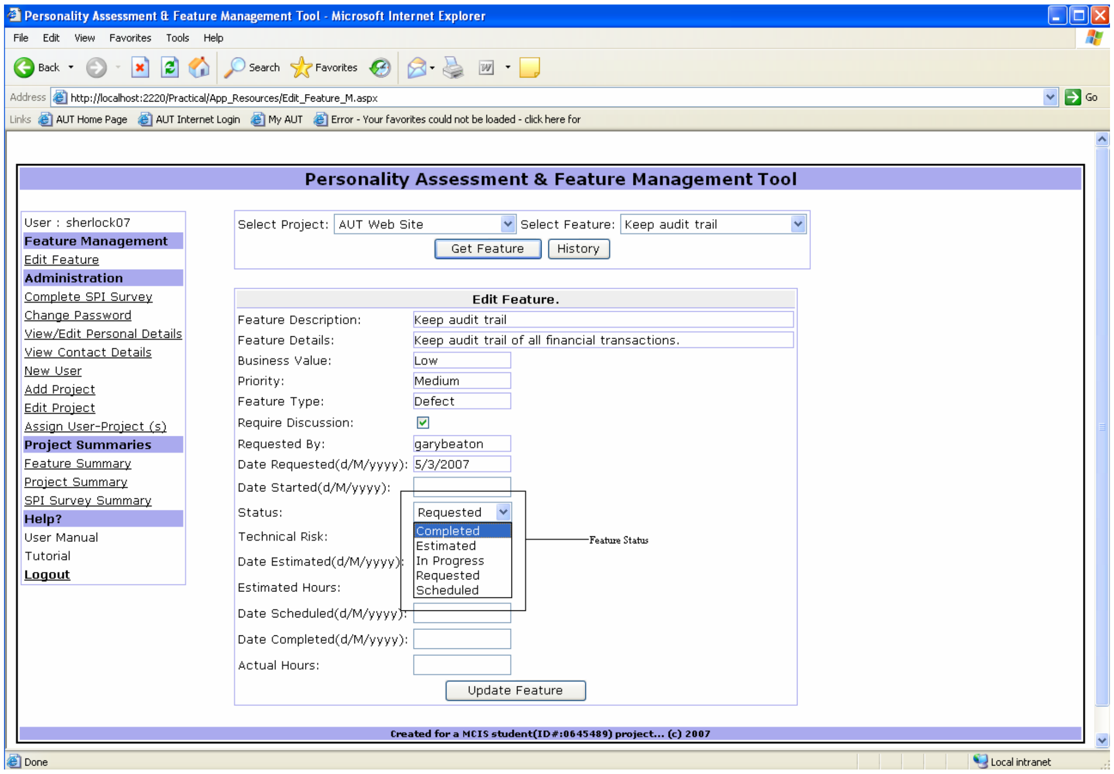
User : sherlock07

Select Project: AUT Web Site Select Feature: Keep audit trail

Feature_Details	Business_Value	Feature_Type	Priority	Must_Discuss	Login	Date_Modified	Status
Keep audit trail of all financial transactions.	Low	Defect	Low	True	garybeaton	03/05/2007	Requested
Keep audit trail of all financial transactions.	Low	Defect	Medium	True	garybeaton	03/26/2007	Requested

Created for a MCIS student(ID#:0645489) project... (c) 2007

Figure 6.7. Edit Feature Interface Showing Feature Statuses (Developer)



Personality Assessment & Feature Management Tool - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost:2220/Practical/App_Resources/Edit_Feature_M.aspx

Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for

Personality Assessment & Feature Management Tool

User : sherlock07

Feature Management

Edit Feature

Administration

Complete SPI Survey

Change Password

View/Edit Personal Details

View Contact Details

New User

Add Project

Edit Project

Assign User-Project (s)

Project Summaries

Feature Summary

Project Summary

SPI Survey Summary

Help?

User Manual

Tutorial

Logout

Select Project: AUT Web Site Select Feature: Keep audit trail

Get Feature History

Edit Feature.

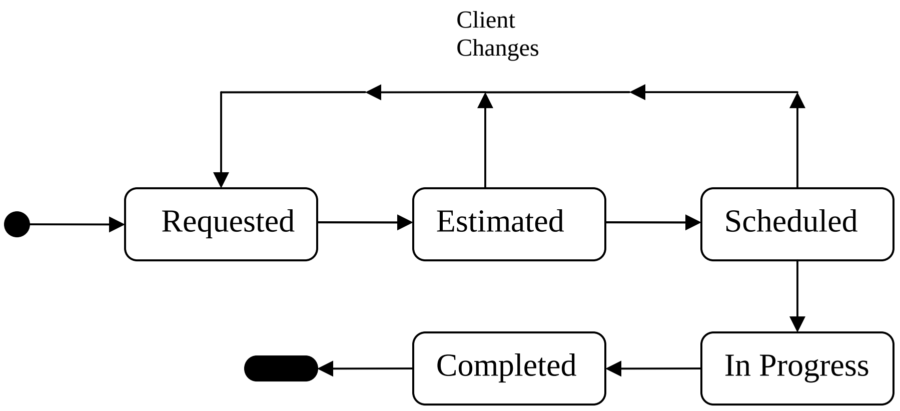
Feature Description:	Keep audit trail
Feature Details:	Keep audit trail of all financial transactions.
Business Value:	Low
Priority:	Medium
Feature Type:	Defect
Require Discussion:	✓
Requested By:	garybeaton
Date Requested(d/M/yyyy):	5/3/2007
Date Started(d/M/yyyy):	
Status:	Requested
Technical Risk:	
Date Estimated(d/M/yyyy):	
Estimated Hours:	
Date Scheduled(d/M/yyyy):	
Date Completed(d/M/yyyy):	
Actual Hours:	

Update Feature

Created for a MCIS student(ID#:0645489) project... (c) 2007

Done Local intranet

Figure 6.8. Feature Traversing States



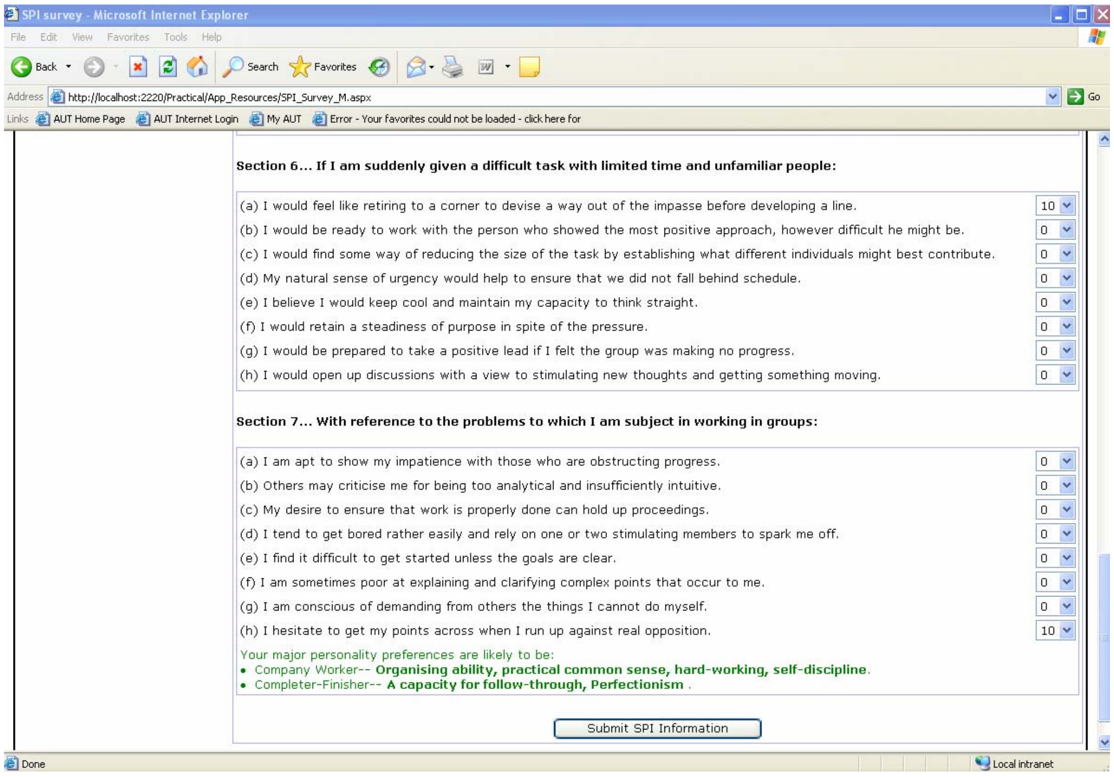
Administration

ASRMT provides an administration menu which allows users to administer projects, user options, and complete the SPI survey (see Figure 6.3). These options are facilitated through the following submenus:

Complete SPI Survey: One of the most important features of ASRMT is its provision of Belbin's SPI survey[1]. Through the SPI survey users are able to determine and store information about their main personality preference. Individuals are expected to possess varying strengths based on their personality preference (see Table 4.1 for a summary). Thus, this feature assists project managers to assemble the most effective teams, having the right mix of personalities. To complete the survey individuals are required to answer fifty six questions, divided evenly among seven sections. For each section, a total of ten points must be distributed based on how the user feels about the questions. When the SPI survey is submitted, the user's two main personality preferences are returned (see Figure 6.9).

[1] Use of the Belbin SPI in ASRMT is not for profit, and is purely on a conceptual basis for this research, to verify whether the concept of personnel capability management can be integrated in an agile project management prototype. Therefore, please ensure that if you are interested in analysing Belbin's team roles, you use one of the three approved methods links which can be found at www.belbin.com .

Figure 6.9. Belbin's SPI Survey Interface



SPI survey - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2220/Practical/App_Resources/SPI_Survey_M.aspx
Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for
Done
Local intranet

Section 6... If I am suddenly given a difficult task with limited time and unfamiliar people:

(a) I would feel like retiring to a corner to devise a way out of the impasse before developing a line.	10
(b) I would be ready to work with the person who showed the most positive approach, however difficult he might be.	0
(c) I would find some way of reducing the size of the task by establishing what different individuals might best contribute.	0
(d) My natural sense of urgency would help to ensure that we did not fall behind schedule.	0
(e) I believe I would keep cool and maintain my capacity to think straight.	0
(f) I would retain a steadiness of purpose in spite of the pressure.	0
(g) I would be prepared to take a positive lead if I felt the group was making no progress.	0
(h) I would open up discussions with a view to stimulating new thoughts and getting something moving.	0

Section 7... With reference to the problems to which I am subject in working in groups:

(a) I am apt to show my impatience with those who are obstructing progress.	0
(b) Others may criticise me for being too analytical and insufficiently intuitive.	0
(c) My desire to ensure that work is properly done can hold up proceedings.	0
(d) I tend to get bored rather easily and rely on one or two stimulating members to spark me off.	0
(e) I find it difficult to get started unless the goals are clear.	0
(f) I am sometimes poor at explaining and clarifying complex points that occur to me.	0
(g) I am conscious of demanding from others the things I cannot do myself.	0
(h) I hesitate to get my points across when I run up against real opposition.	10

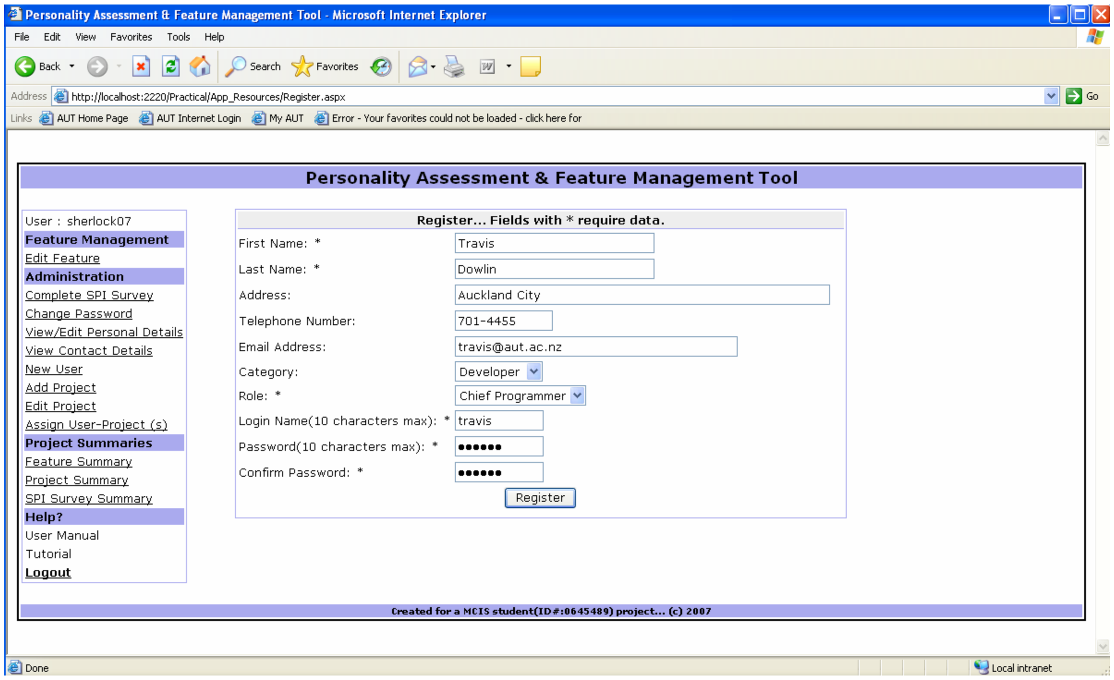
Your major personality preferences are likely to be:

- Company Worker-- **Organising ability, practical common sense, hard-working, self-discipline**.
- Completer-Finisher-- **A capacity for follow-through, Perfectionism** .

Submit SPI Information

New User: ASRMT is a secured software tool. Thus, in order for users to access ASRMT, they must be registered. Only an ASRMT administrator or a registered project manager is allowed to add users. The New User menu on the main interface (see Figure 6.3) opens the Register interface which stores First Name, Last Name, Address, Telephone Number, Email Address, Category (Customer, Developer, or Manager), Login Name, Password, and Confirm Password details to be entered. For users belonging to the 'Developer' category, their Role (Programmer, Analyst, or Chief Programmer) must also be specified (see Figure 6.10).

Figure 6.10. New User Interface


Personality Assessment & Feature Management Tool - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2220/Practical/App_Resources/Register.aspx
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for
Done
Local intranet

Personality Assessment & Feature Management Tool

User : sherlock07
Feature Management
Edit Feature
Administration
Complete SPI Survey
Change Password
View/Edit Personal Details
View Contact Details
New User
Add Project
Edit Project
Assign User-Project (s)
Project Summaries
Feature Summary
Project Summary
SPI Survey Summary
Help?
User Manual
Tutorial
Logout

Register... Fields with * require data.

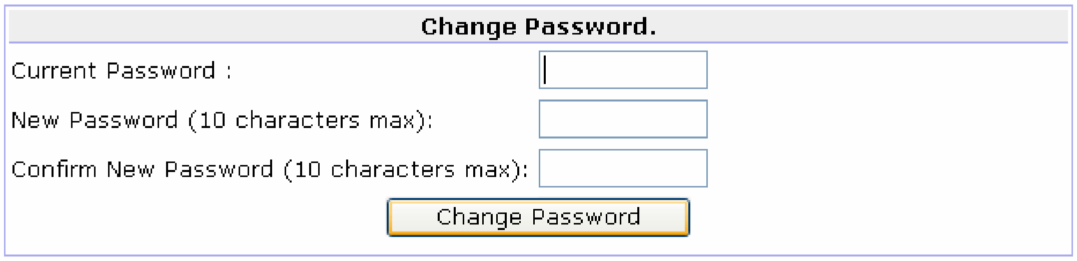
First Name: *	Travis
Last Name: *	Dowlin
Address:	Auckland City
Telephone Number:	701-4455
Email Address:	travis@aut.ac.nz
Category:	Developer
Role: *	Chief Programmer
Login Name(10 characters max): *	travis
Password(10 characters max): *	••••••
Confirm Password: *	••••••

Register

Created for a MCIS student(ID#:0645489) project... (c) 2007

Change Password: ASRMT users can ensure their details are secured by changing their passwords from time to time (see Figure 6.11). To change their password, users are required to enter their Current Password, and then enter their New Password twice.

Figure 6.11. Change Password Interface


Change Password.

Current Password :

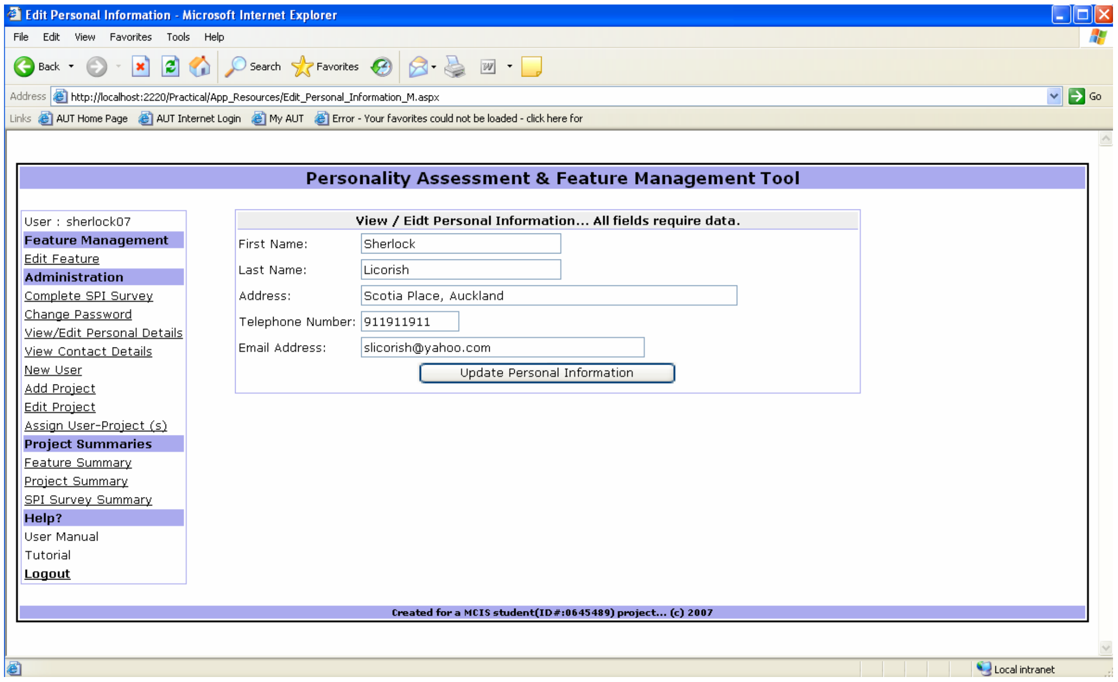
New Password (10 characters max):

Confirm New Password (10 characters max):

Change Password

View/ Edit Personal Details: Since ASRMT users are added by the administrator or project manager, ASRMT allows users the opportunity to view and edit their previously entered information (see Figure 6.12). Users are allowed to edit their First Name, Last Name, Address, Telephone Number and Email Address.

Figure 6.12. View/Edit Personal Information Interface

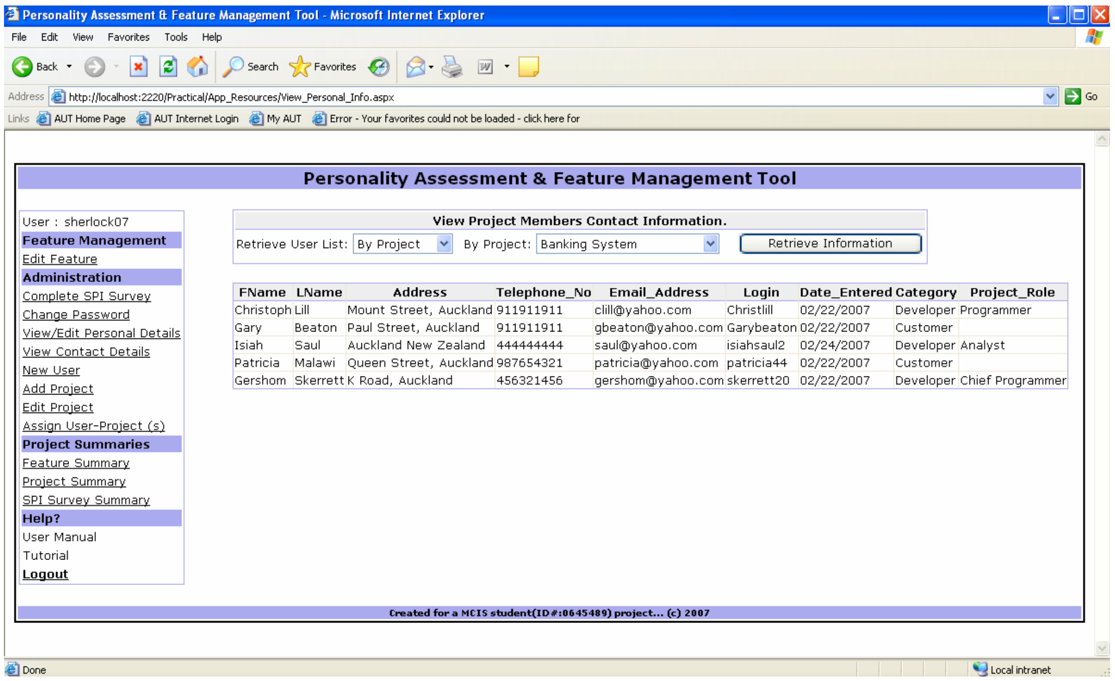

Edit Personal Information - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://localhost:2220/Practical/App_Resources/Edit_Personal_Information_M.aspx
Go
Links
AUT Home Page
AUT Internet Login
My AUT
Error - Your favorites could not be loaded - click here for
Personality Assessment & Feature Management Tool
User : sherlock07
Feature Management
Edit Feature
Administration
Complete SPI Survey
Change Password
View/Edit Personal Details
View Contact Details
New User
Add Project
Edit Project
Assign User-Project (s)
Project Summaries
Feature Summary
Project Summary
SPI Survey Summary
Help?
User Manual
Tutorial
Logout
Update Personal Information
Created for a MCIS student(ID#:0645489) project... (c) 2007
Local intranet

View / Eidt Personal Information... All fields require data.

First Name:	Sherlock
Last Name:	Licorish
Address:	Scotia Place, Auckland
Telephone Number:	911911911
Email Address:	slicorish@yahoo.com

View Contact Details: Developers and project managers are allowed to view clients' contact information (see Figure 6.13). Developers are restricted to viewing of information for clients who are part of their projects, whereas project managers are allowed to view all clients' or developers' contact information. Project managers may also use this interface to see which users are participating in specific projects; see Figure 6.13 for example.

Figure 6.13. View Contact Details Interface


Personality Assessment & Feature Management Tool - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2220/Practical/App_Resources/View_Personal_Info.aspx
Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for
Personality Assessment & Feature Management Tool
User : sherlock07
Feature Management
Edit Feature
Administration
Complete SPI Survey
Change Password
View/Edit Personal Details
View Contact Details
New User
Add Project
Edit Project
Assign User-Project (s)
Project Summaries
Feature Summary
Project Summary
SPI Survey Summary
Help?
User Manual
Tutorial
Logout
Retrieve Information
Created for a MCIS student(ID#:0645489) project... (c) 2007
Done
Local intranet

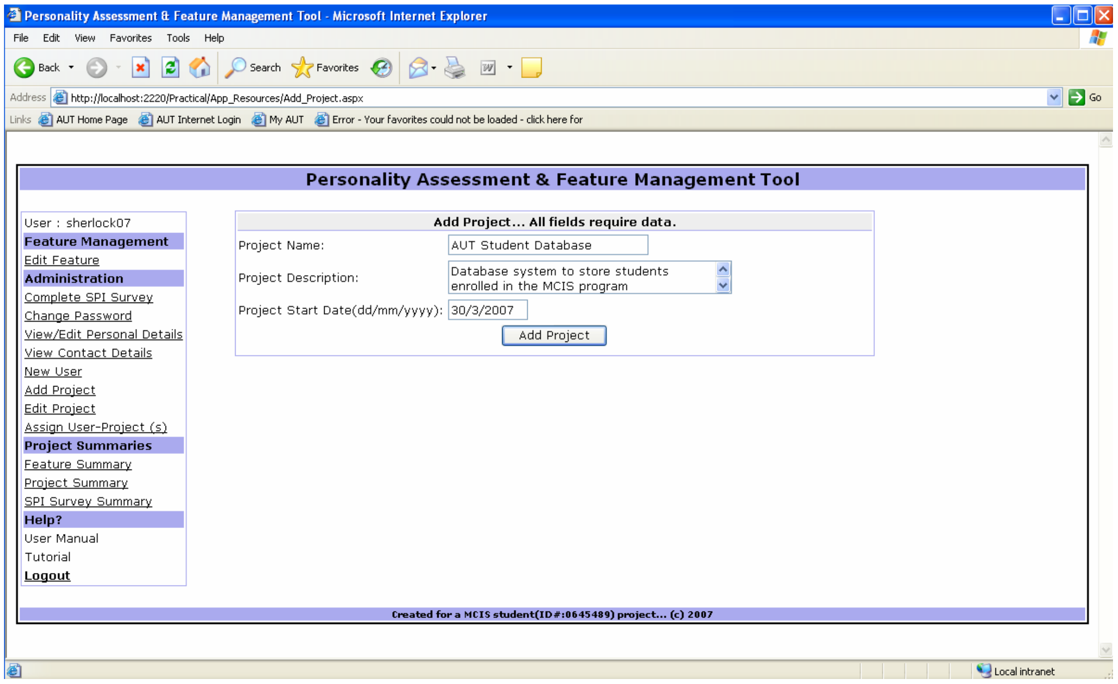
View Project Members Contact Information.

Retrieve User List: By Project By Project: Banking System

FName	LName	Address	Telephone_No	Email_Address	Login	Date_Entered	Category	Project_Role
Christoph	Lill	Mount Street, Auckland	911911911	clill@yahoo.com	Christlill	02/22/2007	Developer	Programmer
Gary	Beaton	Paul Street, Auckland	911911911	gbeaton@yahoo.com	Garybeaton	02/22/2007	Customer	
Isiah	Saul	Auckland New Zealand	444444444	saul@yahoo.com	isiahsaul2	02/24/2007	Developer	Analyst
Patricia	Malawi	Queen Street, Auckland	987654321	patricia@yahoo.com	patricia44	02/22/2007	Customer	
Gershom	Skerrett	K Road, Auckland	456321456	gershom@yahoo.com	skerrett20	02/22/2007	Developer	Chief Programmer

Add Project: Before clients and developers are allowed to interface with projects (manage project features) they must be added to projects by their project manager. ASRMT allows the project manager to add projects using the Add Project menu (see Figure 6.3). To add projects, project managers must specify the Project Name, Project Description, and Project Start Date (this date can be estimated if the date for starting the project is unclear, and updated at a later date) (see Figure 6.14).

Figure 6.14. Add Project Interface


Personality Assessment & Feature Management Tool - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address http://localhost:2220/Practical/App_Resources/Add_Project.aspx
Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for
Local intranet

Personality Assessment & Feature Management Tool

User : sherlock07
Feature Management
Edit Feature
Administration
Complete SPI Survey
Change Password
View/Edit Personal Details
View Contact Details
New User
Add Project
Edit Project
Assign User-Project (s)
Project Summaries
Feature Summary
Project Summary
SPI Survey Summary
Help?
User Manual
Tutorial
Logout

Add Project... All fields require data.

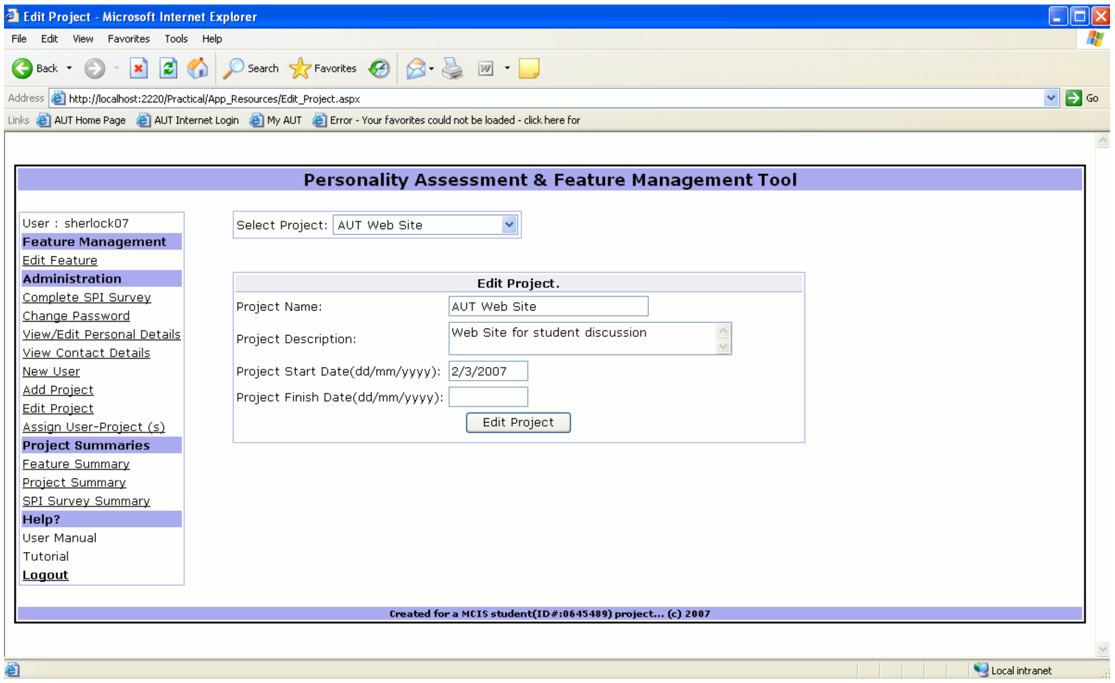
Project Name:	AUT Student Database
Project Description:	Database system to store students enrolled in the MCIS program
Project Start Date(dd/mm/yyyy):	30/3/2007

Add Project

Created for a MCIS student(ID#:0645489) project... (c) 2007

Edit Project: Project managers are allowed to edit projects previously added using the Edit Project menu. Project managers can also update projects when they are completed by adding a Project Finish Date (see Figure 6.15).

Figure 6.15. Edit Project Interface


Edit Project - Microsoft Internet Explorer

Address http://localhost:2220/Practical/App_Resources/Edit_Project.aspx

Personality Assessment & Feature Management Tool

User : sherlock07

Feature Management
- Edit Feature

Administration
- Complete SPI Survey
- Change Password
- View/Edit Personal Details
- View Contact Details
- New User
- Add Project
- Edit Project
- Assign User-Project (s)

Project Summaries
- Feature Summary
- Project Summary
- SPI Survey Summary

Help?
- User Manual
- Tutorial

Logout

Select Project: AUT Web Site

Edit Project.

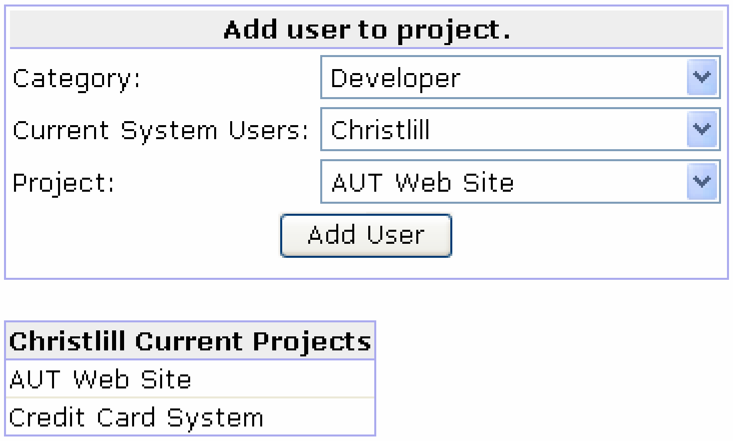
Project Name:	AUT Web Site
Project Description:	Web Site for student discussion
Project Start Date(dd/mm/yyyy):	2/3/2007
Project Finish Date(dd/mm/yyyy):	

Edit Project

Created for a MCIS student(ID#:0645489) project... (c) 2007

Assign Users–Project(s): Project managers are allowed to Add Users (clients and developers) to specific projects. Clients and developers can only interface with projects that they have been previously added to (see Figure 6.16). This interface also allows the project manager to see all the projects each user is participating in.

Figure 6.16. Add user to Project Interface


Add user to project.

Category:	Developer
Current System Users:	Christlill
Project:	AUT Web Site

Add User

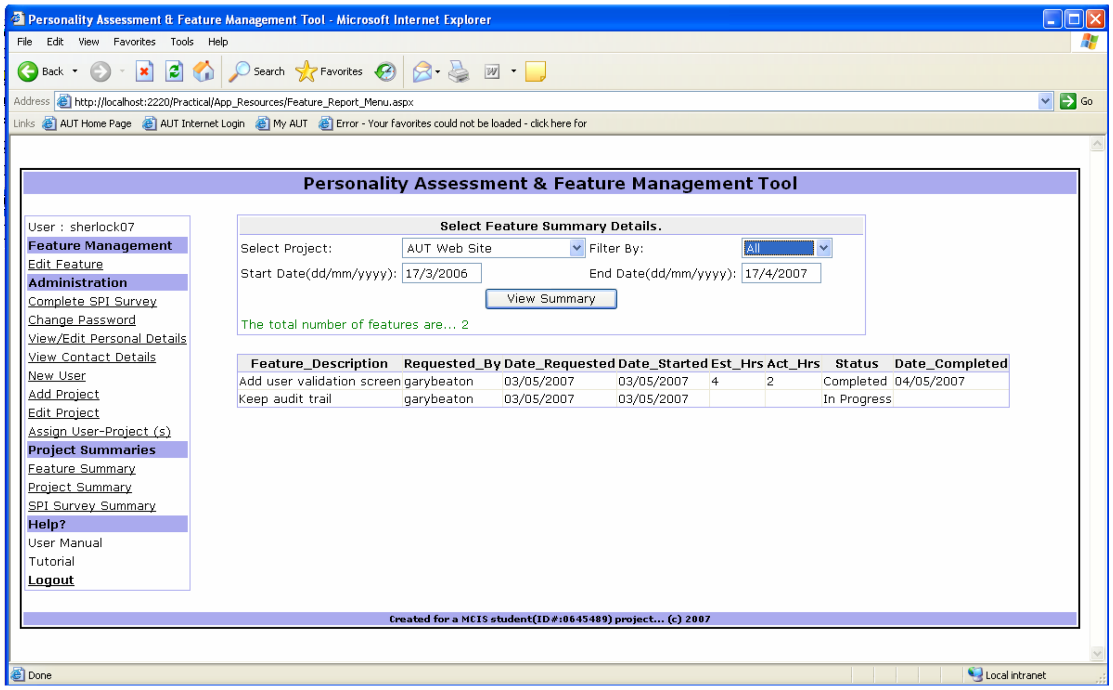
Christlill Current Projects
AUT Web Site
Credit Card System

Project Summaries

ASRMT provides reports in the form of project summaries. Project summaries are available for all features, projects, and users' SPI survey data previously added. These summaries are meant to assist project team members to keep track of project progress, and help project managers with decision making.

Feature Summary: ASRMT provides a wide range of summaries for features (see Figure 6.17). Feature summaries include: All features for specific projects, Features by a specific Type (New Features, Defect, and Enhancement), Features by Priority (Low, Medium, and High), Features for a given Risk Rating (Low, Medium, and High), and Features by Status (Requested, Estimated, Scheduled, In Progress, and Completed). For these summaries, users must specify a given date range (Start Date and End Date).

Figure 6.17. Feature Summary Interface


Personality Assessment & Feature Management Tool - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2220/Practical/App_Resources/Feature_Report_Menu.aspx
Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for
Done
Local intranet

Personality Assessment & Feature Management Tool

User : sherlock07
Feature Management
Edit Feature
Administration
Complete SPI Survey
Change Password
View/Edit Personal Details
View Contact Details
New User
Add Project
Edit Project
Assign User-Project (s)
Project Summaries
Feature Summary
Project Summary
SPI Survey Summary
Help?
User Manual
Tutorial
Logout

Select Feature Summary Details.

Select Project:	AUT Web Site	Filter By:	All
Start Date(dd/mm/yyyy):	17/3/2006	End Date(dd/mm/yyyy):	17/4/2007

View Summary

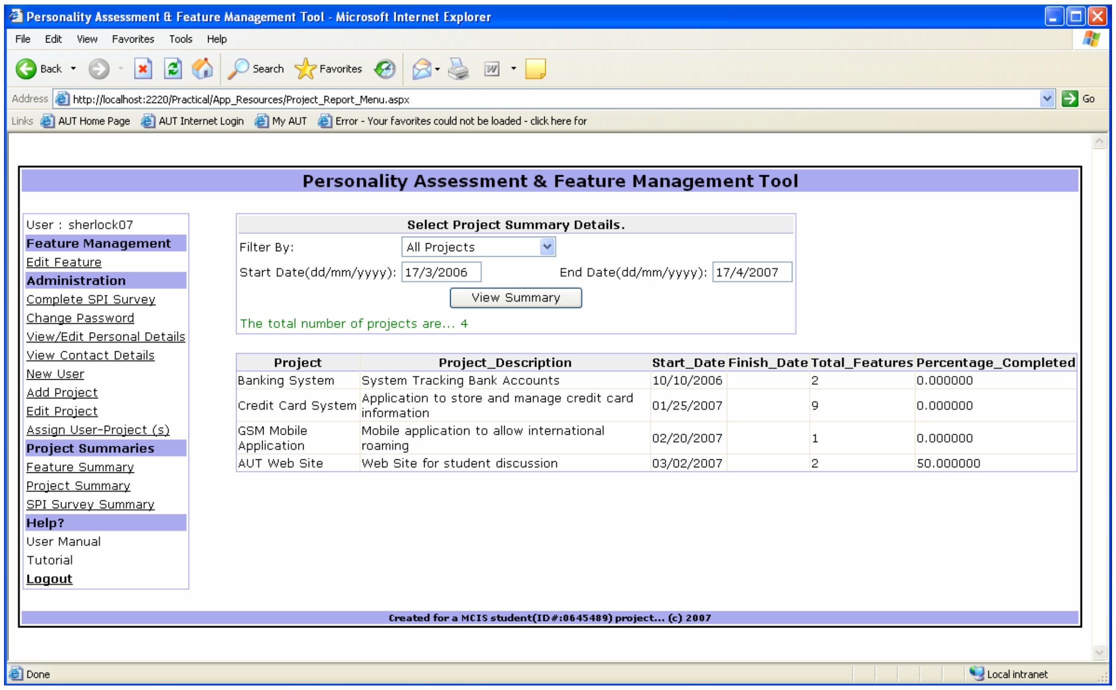
The total number of features are... 2

Feature_Description	Requested_By	Date_Requested	Date_Started	Est_Hrs	Act_Hrs	Status	Date_Completed
Add user validation screen	garybeaton	03/05/2007	03/05/2007	4	2	Completed	04/05/2007
Keep audit trail	garybeaton	03/05/2007	03/05/2007			In Progress	

Created for a MCIS student(ID#:0645489) project... (c) 2007

Project Summary: Project summary options provided by ASRMT include: Summary for All Projects, Summary for Specific Project, Summary for Outstanding Projects, and Summary for Completed Projects (see Figure 6.18). These summaries provide information such as the project name and its description, its start date, its finish date, total number of features, and percentage of the features completed. For each summary (except the Specific Project) the user must specify a Start Date and End Date.

Figure 6.18. Project Summary Interface


Personality Assessment & Feature Management Tool - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2220/Practical/App_Resources/Project_Report_Menu.aspx Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for

Personality Assessment & Feature Management Tool

User : sherlock07
Feature Management
Edit Feature
Administration
Complete SPI Survey
Change Password
View/Edit Personal Details
View Contact Details
New User
Add Project
Edit Project
Assign User-Project (s)
Project Summaries
Feature Summary
Project Summary
SPI Survey Summary
Help?
User Manual
Tutorial
Logout

Select Project Summary Details.

Filter By: All Projects

Start Date(dd/mm/yyyy): 17/3/2006 End Date(dd/mm/yyyy): 17/4/2007

View Summary

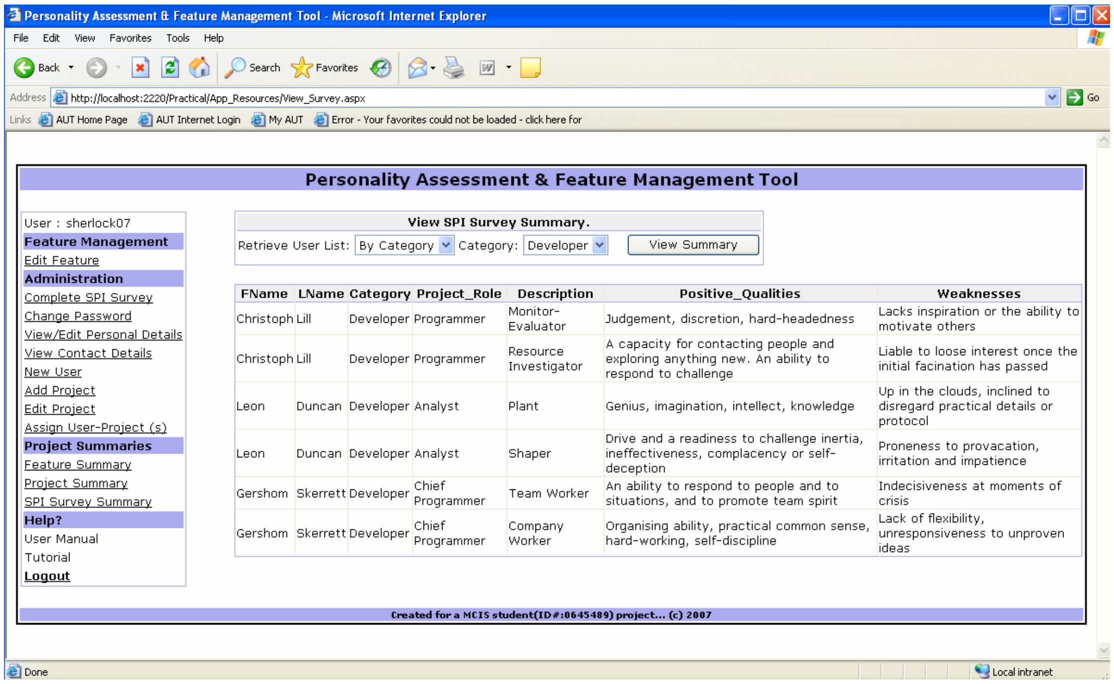
The total number of projects are... 4

Project	Project_Description	Start_Date	Finish_Date	Total_Features	Percentage_Completed
Banking System	System Tracking Bank Accounts	10/10/2006		2	0.000000
Credit Card System	Application to store and manage credit card information	01/25/2007		9	0.000000
GSM Mobile Application	Mobile application to allow international roaming	02/20/2007		1	0.000000
AUT Web Site	Web Site for student discussion	03/02/2007		2	50.000000

Created for a MCIS student(ID#:0645489) project... (c) 2007

SPI Survey Summary: A summary of user personality preference information is also provided by ASRMT. Through this menu project managers can gain information which is likely to enhance activities such as: assigning individuals to roles in the software team, and monitoring their progress by comparing work output and personality traits with what is recorded by the SPI questionnaire. Additionally, ASRMT also highlights the main weaknesses that might be associated with each personality preference. Project managers can work to leverage individuals' strengths while minimising the impact of their weaknesses through team composition. Figure 6.19 shows a sample SPI summary for a few developers currently registered.

Figure 6.19. SPI Survey Summary Interface


Personality Assessment & Feature Management Tool - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2220/Practical/App_Resources/View_Survey.aspx
Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for
Personality Assessment & Feature Management Tool
User : sherlock07
Feature Management
Edit Feature
Administration
Complete SPI Survey
Change Password
View/Edit Personal Details
View Contact Details
New User
Add Project
Edit Project
Assign User-Project (s)
Project Summaries
Feature Summary
Project Summary
SPI Survey Summary
Help?
User Manual
Tutorial
Logout
View SPI Survey Summary.
View Summary
Created for a MCIS student(ID#:0645489) project... (c) 2007
Done
Local intranet

Retrieve User List: By Category Category: Developer

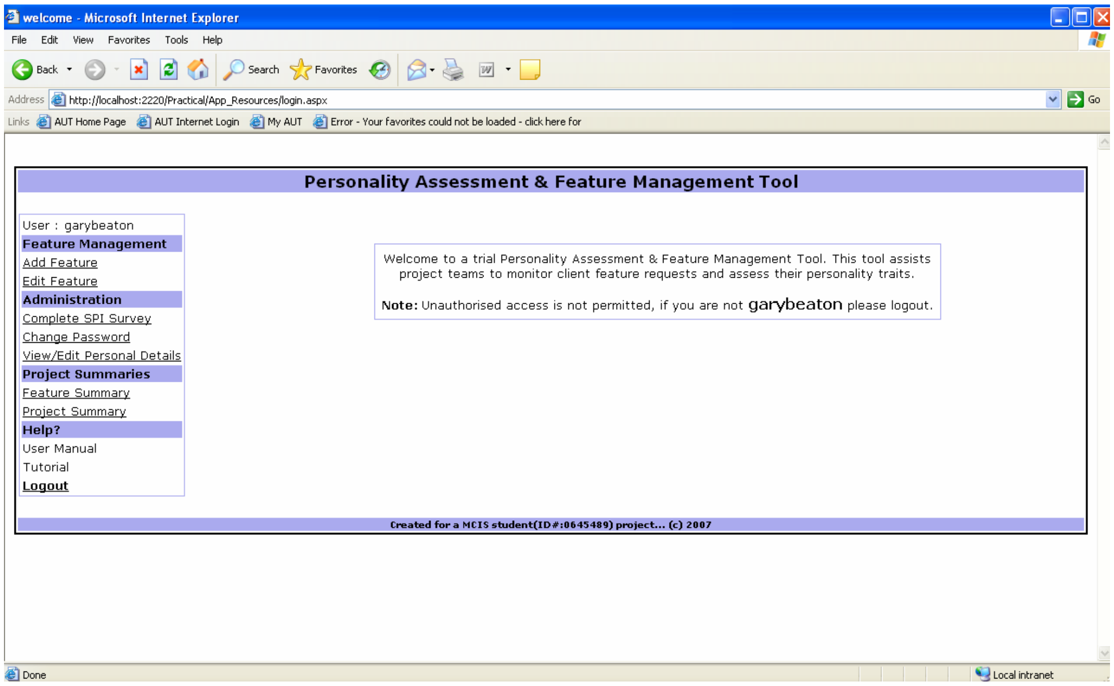
FName	LName	Category	Project_Role	Description	Positive_Qualities	Weaknesses
Christoph	Lill	Developer	Programmer	Monitor-Evaluator	Judgement, discretion, hard-headedness	Lacks inspiration or the ability to motivate others
Christoph	Lill	Developer	Programmer	Resource Investigator	A capacity for contacting people and exploring anything new. An ability to respond to challenge	Liable to loose interest once the initial facination has passed
Leon	Duncan	Developer	Analyst	Plant	Genius, imagination, intellect, knowledge	Up in the clouds, inclined to disregard practical details or protocol
Leon	Duncan	Developer	Analyst	Shaper	Drive and a readiness to challenge inertia, ineffectiveness, complacency or self-deception	Proneness to provacation, irritation and impatience
Gershom	Skerrett	Developer	Chief Programmer	Team Worker	An ability to respond to people and to situations, and to promote team spirit	Indecisiveness at moments of crisis
Gershom	Skerrett	Developer	Chief Programmer	Company Worker	Organising ability, practical common sense, hard-working, self-discipline	Lack of flexibility, unresponsiveness to unproven ideas

6.2.3. ASRMT User Options

ASRMT support three categories of users - Customers, Developers, and Managers. Users possessing manager privileges are provided with access to an extended menu to administer ASRMT, whereas the customer and developer users have restricted access. A description of each category of user and the relevant access limitation are outlined below.

The Client: ASRMT's main Client/Customer purpose is not to eliminate face-to-face contact between clients and developers; rather, ASRMT seeks to allow clients to extend the software development team by offering them a remote interface (reducing their direct interaction). Thus, clients are only allowed access to features of the tool that facilitate 'normal' capabilities but from remote locations (see Figure 6.20). Users having Customer privilege are allowed access to the following: Add Feature, Edit Feature, Complete the SPI Survey, Change Password, View/ Edit Personal Details, Feature Summary, and Project Summary. Clients are only allowed to interface with projects in which they are a member.

Figure 6.20. Client Main Interface


welcome - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://localhost:2220/Practical/App_Resources/login.aspx
Go
Links
AUT Home Page
AUT Internet Login
My AUT
Error - Your favorites could not be loaded - click here for
Done
Local intranet

Personality Assessment & Feature Management Tool

User : garybeaton
Feature Management
Add Feature
Edit Feature
Administration
Complete SPI Survey
Change Password
View/Edit Personal Details
Project Summaries
Feature Summary
Project Summary
Help?
User Manual
Tutorial
Logout

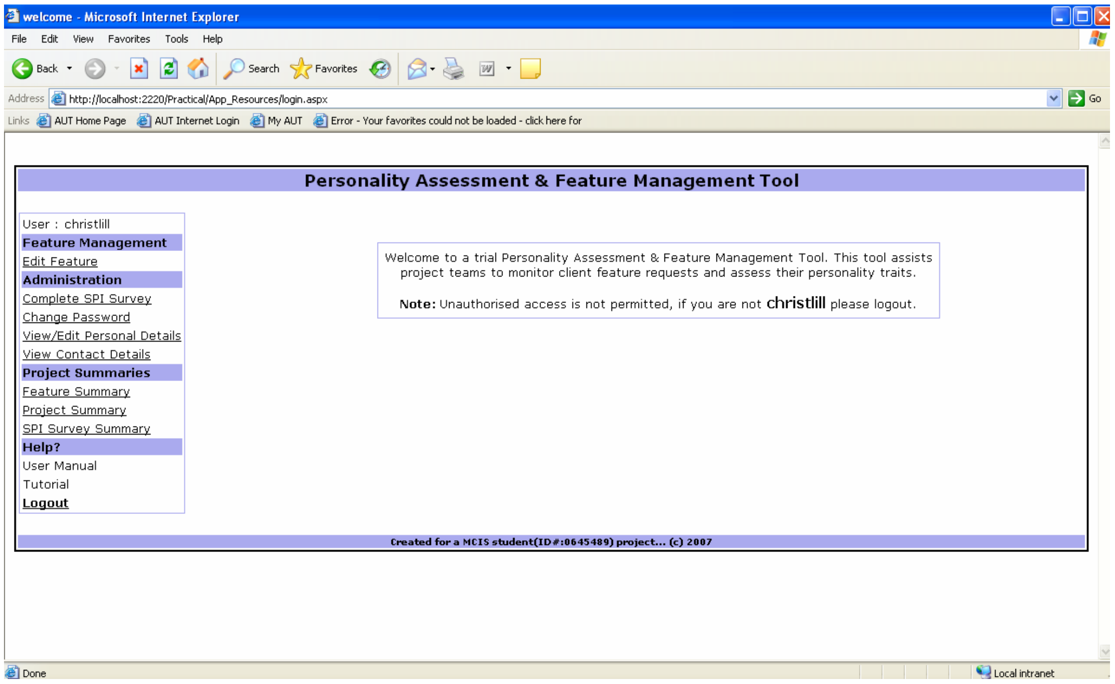
Welcome to a trial Personality Assessment & Feature Management Tool. This tool assists project teams to monitor client feature requests and assess their personality traits.

Note: Unauthorised access is not permitted, if you are not garybeaton please logout.

Created for a MCIS student(ID#:0645489) project... (c) 2007

The Developer: Developers are allowed access to ASRMT features that are meant to support their functions in the software team. Thus, like customers, developers also view a scaled down list of options (see Figure 6.21). Their options include: Edit Feature, Complete SPI Survey, Change Password, View/Edit Personal Details, View Contact Details, Feature Summary, Project Summary, and SPI Survey Summary. Developers are only allowed to interface with projects in which they are a member. They are also only allowed to view the positive qualities associated with their personality preferences. The intent of this restriction is to discourage developers from manipulating their answers to the SPI questionnaire in an effort to avoid appearing to have certain weaknesses.

Figure 6.21. Developer Main Interface


welcome - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2220/Practical/App_Resources/login.aspx
Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for
User : christlill
Feature Management
Edit Feature
Administration
Complete SPI Survey
Change Password
View/Edit Personal Details
View Contact Details
Project Summaries
Feature Summary
Project Summary
SPI Survey Summary
Help?
User Manual
Tutorial
Logout
Done
Local intranet

Personality Assessment & Feature Management Tool

Welcome to a trial Personality Assessment & Feature Management Tool. This tool assists project teams to monitor client feature requests and assess their personality traits.

Note: Unauthorised access is not permitted, if you are not christlill please logout.

Created for a MCIS student(ID#:0645489) project... (c) 2007

The Project Manager: Project managers have administrator privileges. Thus, project managers are allowed full access to the entire functionality of ASRMT (see ASRMT full menu in Figure 6.3).

6.3. Summary

Feature-driven development (FDD) practices and Microsoft development tools were used to develop ASRMT to assist agile project teams with their handling of social risks. This tool is intended to reduce social risks in agile teams by allowing project managers the opportunity to assess and manage individuals' personality preferences and traits (to unearth their team strengths and weaknesses so as to place them in most suitable roles) before they are assigned to software project roles and integrated into a software team, and to extend remote project customers (to reduce potential conflicts between onsite customers and the development team). Even though ASRMT was primarily created to assist agile teams with the handling of social risks, this tool also offers features that are likely to support and add benefits to general project management in agile teams. The following chapter provides further discussions on ASRMT intended benefits and addresses the research questions.

7. ASRMT Evaluation and Discussion

This chapter begins with some reflections on the development process employed in ASRMT. The intended benefits of using ASRMT are then explained in relation to the concepts established earlier in this thesis. This chapter also delineates the ASRMT evaluation process and its findings. The research questions listed in Chapter 1 are then revisited. Finally, this chapter concludes with a summary of the discussions presented throughout this chapter.

7.1. Reflections on the Development Process

In general, most of the FDD practices selected for developing ASRMT were followed. Domain object modeling (DOM) was useful; this process enhanced the developer understanding of the domain, and helped the developer to keep the overall project goal in focus. Developing ASRMT feature by feature also allowed the developer to focus on one unit at a time, which reduced the development expectation burden of excessively looking at the requirements of the entire system. Planning by feature, designing by feature, and building by feature also allowed the implementation of complete units of ASRMT, which were subsequently integrated into the overall project after testing. Configuration management enhanced the management of ASRMT versions. In one case, an earlier version of ASRMT was retrieved to revert to a specific user interface (UI) design recommended by the supervisors of this project.

Overall, the benefit of using FDD practices was not clear in this research project, as the relatively small scale of ASRMT did not enable the researcher to manage development outputs and quality against any benchmark. In addition, because of the size of ASRMT, there were moderate differences between the times estimated and the actual times taken for development (see Table 10.2). The increase in development time that did occur was due to the exclusion of time estimates (in the initial feature list summary) for ASRMT research activities (for development), planning and implementing the ASRMT database, and implementing ASRMT reports (see Table 10.2).

Because ASRMT was developed almost exclusively using Microsoft products, the effort for configuring these products to work collectively was trivial. However, open source help on the internet for implementing ASRMT functionalities was scarce. While the technologies proposed for implementing most of ASRMT functionalities were adequate, in addition to cascading style sheets (CSS), themes were also used for implementing ASRMT's consistent visual style. Themes are asp.net technology for maintaining a consistent look and feel of asp.net applications. This technology extends the CSS by offering additional support outside the scope of CSS for asp.net controls.

7.2. Benefits of using ASRMT: Risk Mitigation Capability

ASRMT was developed to support agile processes. Thus, through use of ASRMT, users should be able to reap the benefits existent in similar agile project management tools (see Chapter 5 section 5.2 for the benefits of using project management tools). An outline of the additional benefits offered by ASRMT is presented below.

As mentioned above, ASRMT was developed to track agile processes. Therefore, this tool focuses on providing process support for agile projects and avoids features that may become heavyweight, thereby reducing agility in such project environments. The idea behind ASRMT is to be as lightweight as possible, and easy to learn. Since ASRMT is meant to track agile projects, if information is promptly entered, it is likely to support communication in agile projects. ASRMT is web-based; as such it provides unrestricted accessibility, a key feature of agile methods. ASRMT was developed using an object oriented (OO) approach, thus ASRMT was built with an awareness of the need for scalability, requiring minimal change for additional functionalities (extendibility).

Beyond the architectural benefits stated above, ASRMT also offers functional benefits that are intended to mitigate social risks in agile projects. ASRMT was developed to extend the customer interface and provide personality assessment functionality, in so doing, reducing social risks in agile projects. The following sections further describe the benefit of providing support for remote clients, personnel capability management, highlight some other indirect benefits of using ASRMT, and explain its project management capability.

7.2.1. Support for Remote Clients

User (customer) participation in software development is only beneficial when software teams perform well (see Chapter 4 section 4.1.4). Since there is no guarantee of elevated team performance in software development, having a customer always on-site (as recommended by XP) is a potentially risky practice. In such circumstances, lessening direct interaction with the customer is likely to reduce risks that may be derived through unmanaged customer expectations and interpersonal conflicts. ASRMT provides a way to extend the on-site customer by providing support for remote customer participation. ASRMT was developed to trap customer feature requests, allowing the customer to initiate feature requirements, and to participate in the management of such requirements. For more complex features, ASRMT also offers customers a way of indicating to the development team that they are interested in having a face-to-face meeting to discuss these features (see Figure 6.4).

7.2.2. Support for Personnel Capability Management

Personality differences are the most critical risk affecting software team cohesion. Thus, assessing and managing team members' personalities should provide an opportunity to reduce such risk. The aim here is to enable managers to assemble software teams with a balance of personalities (see Chapter 4 section 4.1.2 for further details). ASRMT allows team members the opportunity to assess their main personality preferences before team formation. Thus, the project manager is afforded the opportunity to assemble a team with the best mix of personality types for a given software development project (see Figure 6.9 for a SPI assessment sample). For example: examining Belbin's team roles shown in Table 4.1 in Chapter 4, a typical project management decision might be - Individuals possessing the *Plant* personality are assigned to the Programmer role, whereas individuals possessing the *Completer-Finisher* personality could function most effectively as Quality Assurance Specialists (QASs).

The reason for the project management decision in the above example might be as follows:

- Since the *Plant* role is linked to 'genius, imagination, intellect, and knowledge' personality traits, individuals occupying this role might be prone to rapid and innovative solutions to software problems, making them most suitable as programmers.
- On the other hand, the *Completer-Finisher* role is associated with 'a capacity to follow-through, perfectionism'. Thus, individuals occupying this role may pay keen attention to details unearthing errors, and ensuring software quality, functioning effectively as QASs.

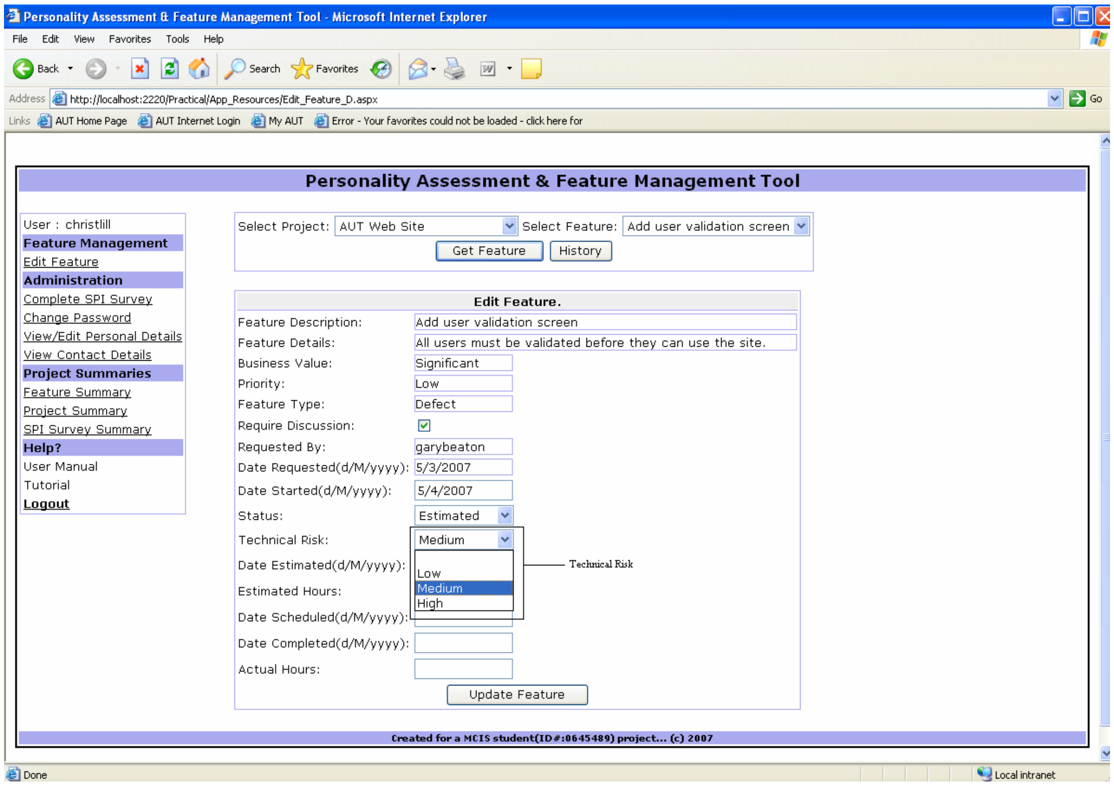
7.2.3. Other Indirect Benefits of ASRMT: Project Management Capability

ASRMT was developed to assist project teams by offering them a way to lessen social risks. In addition to its intended purpose, ASRMT may also offer several other benefits to project teams. Since communication is critical for software development project success, ASRMT's capacity to act as a communication vehicle potentially lessens risks, offering team members a way to store and retrieve information for later use. In this regard, team members do not have to entirely rely on tacit knowledge. Additionally, ASRMT is web-based; therefore this tool can support communication across distributed agile teams. ASRMT may also act as a reservoir for project information, thus team members are afforded the opportunity to perform post mortem reviews which should enhance their future decision making process.

ASRMT is not intended to be a full-scale project management tool (see Table 7.1 for a comparative summary of ASRMT and popular project management tools features). Rather, this tool was developed to primarily assist agile project teams with their handling of some of the social aspects of project management. Thus, through ASRMT use, social project risks are likely to be lessened (see sections 7.2.1 and 7.2.2). However, as mentioned earlier, ASRMT may also provide benefits above and beyond its planned purpose. ASRMT may provide assistance to project teams through its project management capability. In agile software development methods, project coordination and the management of project interdependencies and uncertainties are executed informally. Such a method of project management may itself prove risky, as coordination and communication are seen as key success areas in project management.

ASRMT offers a project team the opportunity to coordinate software development. In ASRMT, customers enter feature requests, which may be beyond those initially requested or discussed at formal meetings. ASRMT offers customers a way to manage features, prioritise features, rate their level of significance, and categorise features in terms of business value. Thus, developers have an opportunity to jointly manage feature development based on customer priorities (see Chapter 6 section 6.2.2). Developers are also provided with the opportunity to manage feature states, and the technical risk (see Figure 7.1) associated with each feature. ASRMT also enables developers to track feature estimation information and provides a mechanism for developers to review feature information to assist with future decision making. It is also possible to track feature rework information using ASRMT. For features entered with the feature type 'Defect' or 'Enhancement', development information can be measured, to allow developers a way to consider rework effort (see Figure 6.17). Additionally, ASRMT could assist project managers with project coordination by offering them a way to manage project information, feature development information, and to track team members' development progress, and their task assignments.

Figure 7.1. Edit Feature Interface Highlighting Technical Risk Field



Personality Assessment & Feature Management Tool - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2220/Practical/App_Resources/Edit_Feature_D.aspx
Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for
Done
Local intranet

Personality Assessment & Feature Management Tool

User : christlill

Feature Management

Edit Feature

Administration

Complete SPI Survey

Change Password

View/Edit Personal Details

View Contact Details

Project Summaries

Feature Summary

Project Summary

SPI Survey Summary

Help?

User Manual

Tutorial

Logout

Select Project: AUT Web Site Select Feature: Add user validation screen

Get Feature History

Edit Feature.

Feature Description:	Add user validation screen
Feature Details:	All users must be validated before they can use the site.
Business Value:	Significant
Priority:	Low
Feature Type:	Defect
Require Discussion:	✓
Requested By:	garybeaton
Date Requested(d/M/yyyy):	5/3/2007
Date Started(d/M/yyyy):	5/4/2007
Status:	Estimated
Technical Risk:	Medium
Date Estimated(d/M/yyyy):	
Estimated Hours:	
Date Scheduled(d/M/yyyy):	
Date Completed(d/M/yyyy):	
Actual Hours:	

Low
Medium
High

Technical Risk

Update Feature

Created for a MCIS student(ID#:0645489) project... (c) 2007

Table 7.1. Comparison summary of ASRMT and popular project management tools features (VersionOne, 2006a)

Feature/Tool Support	ExtremePlanner	VersionOne	DevPlanner	Rally	AceProject	MS Project	ASRMT
Local Installation or Distributed Management for teams of any size	Yes	Yes	LI	HO	Yes	NC	Yes
Remote Accessibility	NC	Yes	No	Yes	Yes	No	Yes
Reporting and Analytics							
Executive level reporting	No	Yes	No	Yes	Yes	Yes	Yes
Project/ Releases reporting	Yes	Yes	Yes	Yes	Yes	NC	No
Shared Projects reporting	Yes	Yes	No	Yes	Yes	NC	Yes
Individual level reporting	No	Yes	No	Yes	Yes	NC	No
Feature management and planning	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Customer request management	Yes	Yes	No	Yes	Yes	NC	Yes
Feature estimation and prioritisation	Yes	Yes	Yes	Yes	Yes	NC	Yes
Defects estimation and prioritisation	No	Yes	Yes	Yes	Yes	NC	Yes
Feature tracking	Yes	Yes	Yes	Yes	Yes	NC	Yes
Individual feature tracking and estimation	No	Yes	No	Yes	Yes	No	No
Password authentication, new team member addition, role management	Yes	Yes	No	Yes	Yes	No	Yes
Audit history	Yes	Some	No	No	Yes	No	Yes
Project information archiving	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Feature Risk rating scheme	Yes	Yes	No	Yes	No	No	Yes
Feature Estimation and implementation summary	Basic	Yes	EO	Yes	Yes	Yes	Yes
Easy to use, short learning curve	RE	RE	Yes	RE	RE	RE	Yes
Risk Mitigating Feature							
Personnel Capability Management	No	No	No	No	No	No	Yes
Customer interface to extend the on-site customer	No	No	No	No	No	No	Yes

Key: LI - Local Installation, HO - Hosted Onsite, NC – Need Configuration, EO – Estimation Only, RE – Relatively Easy.

7.3. The ASRMT User Evaluation

The scale and available resource for this research project do not allow for ASRMT to be validated in live project settings. Nonetheless, in order to measure the extent to which ASRMT is usable, and whether or not the tool presents proof of the concepts presented in this research project, it is both necessary and useful to conduct informal evaluations. ASRMT was therefore verified in informal settings using a small number of software engineering experts. In this section an explanation of the method used for ASRMT user evaluation, and discussion of the evaluation findings are provided.

7.3.1. Method for ASRMT User Evaluation

This section outlines the method used for evaluating ASRMT. In keeping with the research aims, ASRMT was informally tested by a small number of software engineering experts. Seven participants were involved in the evaluation process. These participants were agile software developers with varying levels of experience.

The tool was installed on a local server, where the respondents completed a scenario-based evaluation comprised of two parts (see Appendix C for the ASRMT user evaluation instrument). The first part of the evaluation asked respondents to test ASRMT's functionality using 23 tasks in the roles of project manager, developer, and customer, while the second part of the evaluation was designed to solicit feedback regarding respondents' impressions of the tool and their use of it while working through the scenarios. While questions may arise in relation to validity for randomly constructed evaluation instruments (Kirakowski, 2000), it is important to note that the questions for the ASRMT user evaluation were not randomly selected. Rather, this evaluation was adopted from (Lewis, 1995). Lewis's (1995) instrument has been previously assessed for reliability and validity, and recommended for usability evaluations.

The second part of the evaluation comprised 11 questions (using two sub-scales). Seven close-ended questions, each conforming to a Likert scale, were used to evaluate ASRMT's stability and the users' learning experience (first sub-scale). Two close-ended questions conforming to the Likert scale and two further open-ended questions were used to assess ASRMT's usefulness, to consider whether ASRMT addressed the research objectives, and to solicit respondents' overall impressions and recommendations for improving ASRMT (second sub-scale). The possible answers to the questions conforming to the Likert scale were on discrete continuums. These included strongly agree, agree, disagree, and strongly disagree (bi-polar) options. The answers were linearly scaled from one to four (where a strongly agree choice was represented by one, and four represented a strongly disagree choice), offering respondents no neutral choice such as 'neither agree nor disagree'. This approach was deliberately selected to force respondents to express an opinion. While there may be threats to reliability for usability evaluations employing such an approach, given the target respondents (experts), this option presents a low threat to the reliability of the findings (Kirakowski, 2000). As mentioned previously, the two open-ended questions were aimed at capturing respondents' positive and negative impressions of ASRMT, and their suggestions for improving ASRMT.

Given the scale of the ASRMT user evaluation (only seven respondents), responses to close-ended questions were aggregated to determine the number of the respondents that favored a particular choice (strongly agree and agree were taken to be positive responses, while strongly disagree and disagree were negative responses). Open-ended responses were analysed using content analysis. These responses were summarised into six categories, each response being either positive or negative. The frequency of each occurrence (positive or negative response) was aggregated. Details of the evaluation findings and user feedback are provided in the following section.

7.3.2. ASRMT Evaluation Findings and User Feedback

All respondents completed the evaluation in full. The average time taken to complete the evaluation was 58 minutes. Respondents felt that the concepts and ideas behind the development of ASRMT were excellent. Correspondingly, six of the respondents thought ASRMT was easy to use. Of the seven respondents, three reported one or two bugs whilst using ASRMT. All respondents reported that they were able to successfully complete the scenarios, that ASRMT was easy to learn to use, and that they recovered easily and quickly from errors.

Of the seven respondents, five reported that ASRMT was simple and satisfying to use, while four believed that ASRMT would be useful if used in live projects. The three respondents who did not agree that ASRMT would be useful if used in live projects felt that the tool needed usability improvement before it would be suitable for implementation in live settings. All of the respondents believed that ASRMT offered functionality to address the features in keeping with its purpose. In terms of the respondents' overall impression of ASRMT, all respondents believed that the idea of personnel capability management in a process tool was good and should provide benefits to project management. In addition, all respondents believed that the idea of extending the customer would also be useful for projects where customers are regularly involved with the development tasks and the project teams. Regarding the tool's ease of use, five of the respondents believed that ASRMT's simplicity and ease of use should enhance project management.

Among the recommendations for improvement, two respondents suggested that the idea of personnel capability management should be further investigated to uncover the best instrument for assessing personality preference (and to possibly automatically match assessment outcomes to software roles) before ASRMT is implemented in live settings. Additionally, five respondents believed that a few of ASRMT's user interfaces could be improved; and one respondent suggested that ASRMT might need enhancement if it was to be implemented in large projects. Respondents suggested higher color contrast for user feedback, additional guidance for user tasks, and accommodating multiple date formats. In addition, one respondent also suggested that ASRMT could be extended to include additional functionality such as a discussion feature and automatic e-mail reminders, which are likely to assist project participants.

7.4. Discussion and Contribution

What risks are induced through human collaboration? The first research question sought to identify the risks that arise given the extensive reliance on teams of people in the software development and management processes. In Chapter 4 the issue of human collaboration was extensively examined with respect to software development. A comprehensive review of the relevant literature illustrated that team cohesion and the management of interpersonal skills are most critical to software projects' success. A lack of team cohesion and poor management of interpersonal skills are likely to induce risks associated with personality conflict (see Chapter 4 for further details). In addition, evidence suggests that customer involvement is most beneficial when software teams perform well, but that customer involvement may pose a threat when teams perform poorly. Of the risks that are likely to be induced through human collaboration, poor productivity, project overruns, and software failure are highlighted in the literature.

Even though the risks induced through human collaboration are deemed most severe and are likely to result in software failure, evidence in Chapter 3 shows that current risk management approaches do not adequately address these risks. Furthermore, the assessment of existing tools (reported in Chapter 5) highlights their lack of attention to such risks. Consequently, studies such as this, aimed at examining software processes and offering mechanisms to assist developers to employ most effective practices to assist with social risk mitigation are likely to offer potential significant benefit.

Can a process tool be implemented that effectively addresses social risk management theories? The second research question sought to substantiate whether a process tool could be implemented to effectively address the social risk management theories. It was proposed that such a tool should offer agile teams a way to manage interpersonal skills, and offer remote customers a way to interact with developers by allowing them an interface to the development team in order for them to be directly involved with project management (see Chapter 5 section 5.4.2). This tool should present low overhead in its usage, and also present feature support particular to agile software development contexts.

Based on the findings of the ASRMT user evaluations, this study hypothesizes that a process tool can be implemented that effectively addresses social risk management theories (ASRMT is one such tool). In addition to the management of software requirements, this tool provides support for the management of interpersonal skills and remote customer interactions (see Table 7.1 for a summary of ASRMT features).

In section 7.3.2 above, it is shown that the average time taken to complete the ASRMT evaluation was 58 minutes. This evaluation has 23 tasks; these tasks are composed of personnel capability management functionalities, customer feature management functionalities, and developer feature management functionalities. Since all respondents were able to complete their tasks, in such a relatively small amount of time, it might be said that ASRMT provides low overhead in its usage, and its simple design also makes this tool lightweight and usable.

Agile proponents emphasise process tools that are lightweight, and easy to learn and use. The findings presented in section 7.3.2 also support the solution provided by this study with regard to ease of use and simplicity. Most of the respondents believed that ASRMT was easy to use and learn. This suggests that should ASRMT or any similar tool be adopted by agile developers, such tools are likely to be accepted by these developers. Beyond this study, these findings may also offer insights into the area of research verifying what tool support might be relevant for agile practitioners.

The findings in section 7.3.2 also show that the expert respondents were strongly supportive of the concepts of personnel capability management and remote customer involvement in project management. Even though the instrument selected in this study for determining personality preference might be deemed heavyweight (possessing 56 questions), the *concept* of personality capability management was embraced by the developers. Thus, this offers confirmation that a process tool can effectively address social risk management theories. While the purpose of ASRMT is to present a way to lessen customer interaction with the development team (in response to a particular risk), ASRMT may also offer a way to increase customer involvement for projects where customer presence should be strengthened. Since the literature shows that too little customer involvement may also be catastrophic, balancing customer involvement is beneficial. Therefore, it might be said that the ASRMT solution may support risk mitigation in two ways: it may allow developers to decrease customer involvement when this phenomenon poses a risk, and it may allow developers to increase customer involvement where the lack of customer involvement presents risks.

Will such a tool be useful to agile software teams in terms of improving project risk management? The third research question aimed to confirm whether a tool implemented to address social risks would be useful to agile teams, and could improve their project risk management. The findings of the ASRMT user evaluations (see section 7.3.2) suggest that this tool would indeed be useful to agile teams, and would be likely to improve risk management if implemented in agile projects.

In keeping with earlier discussions, respondents believed that a tool that is simple and easy to use will enhance project management (see section 7.3.2). In addition, respondents believed that considering techniques to assist with social risks is likely to be beneficial to agile teams. While the feedback presented shows that ASRMT can be improved in a number of ways, respondents believed that this or a similar tool is likely to be useful for agile project management, and this solution should assist with project risk management. This evidence supports the solution proposed here, provides confirmation that ASRMT would be useful to agile software teams, and indicates that use of the tool would improve software project risk management.

7.5. Summary

While ASRMT was mainly developed to assist development teams with the handling of social risks, this tool is also likely to offer several other benefits to software project teams. ASRMT is a project management tool; thus, it provides development teams with a facility to communicate project information across teams (in support of both distributed and co-located teams). This tool also presents project teams with an opportunity to track software projects and development effort, and acts as a reservoir for information in order to enhance project managers' future decision making.

ASRMT was informally evaluated by seven software experts of varying levels of experience. The findings of the evaluations revealed that ASRMT is easy to learn, simple to use, relatively bug free, would be useful if used in live projects, and presents a proof of the concepts presented in this research. However, the ASRMT user evaluations also revealed that ASRMT can be improved in a number of ways. It was also suggested that ASRMT could be extended to include additional functionality such as a discussion feature and automatic e-mail reminders, which are likely to assist project participants. From the findings of the ASRMT user evaluation, it can be concluded that such a tool is likely to assist agile developers and project managers with their handling of social risks.

Through the extensive consideration of social risks, the assessment of existing tools and the development and evaluation of ASRMT, the three research questions posed in Chapter 1 have now been addressed. In the following final chapter the work is summarised and overall conclusions are drawn.

8. Conclusions and Recommendations

In this chapter a summary of the experiences and insights gained throughout this research project is presented. Lessons drawn from the work are then outlined before the study concludes with a statement of the limitations of this research and recommendations for future research.

8.1. Summary

In Chapter 2 an introduction to software engineering was presented. Evidence explained that early waterfall techniques used to guide software production were criticised as inappropriate for software project management. In addition, studies examining the reasons for software failure regularly linked software failure to waterfall processes. Consequently, the literature revealed that newer agile software development methodologies might provide improvements to conventional waterfall models, and that their use should also reduce risks. However, it is evident that while agile models may provide improvements (in principle) to the software development process, there exists little evidence to soundly support this position. In addition, this study uncovered that some of the practices recommended by agile methodologies may also directly increase exposure to software project risks.

Analysing the risk management literature in Chapter 3, it was noted that of the risks that are likely to increase through adopting agile methodologies, social risks may be deemed the most critical. In addition, this research found that even though several guides exist for assisting with risk management (provided by the SEI and other research bodies), agile project leaders are most likely to depend on expert judgment. Using this form of project management might not always prove successful, and the agile practice of bringing stakeholders together is likely to introduce serious project risks. To this end, this study examined the effects of stakeholders' interaction in Chapter 4.

Of the risks that are induced through human collaboration, evidence highlighted that personality conflict is potentially the most severe. Personality conflict is likely to result from poor team formation. Personnel capability management is one way of enhancing the process of team formation, and reducing conflicts that are likely to result through personality differences. This research proposed that software development and/or project management tools should implement personnel capability management to assist with risk mitigation in agile teams. In examining the stakeholders' interaction literature it was noted that customer involvement also poses a threat to software development success, especially when software teams perform poorly. In light of this, the study also proposed that providing remote customers with a tool interface to the development team, in order for them to be involved with project management directly but not necessarily on-site, may assist with reducing risks that might otherwise result through customer involvement.

As a consequence, existing software development and/or project management tools were examined in Chapter 5. After evaluating several representative tools findings revealed that VersionOne and Rally are likely to offer agile software developers support for their development activities. In addition, it was suggested that the use of these tools might also reduce risks in agile projects. However, even though social risks are most critical, there has been no attempt to address social risks in VersionOne and Rally. Thus, this study designed and built a tool called ASRMT to address this inadequacy.

In Chapter 6 an outline of the software development methodology and development platform selected for implementing ASRMT was provided. To verify whether agile software development practices enhanced development efficiency, and to understand the implications of using agile approaches, agile feature-driven development (FDD) practices were largely employed throughout the ASRMT software development life cycle (see Table 6.1 for selected practices). The remainder of Chapter 6 described in detail the features and functionality of the ASRMT tool.

A discussion of the development process and an explanation of the benefits expected to accrue from the use of ASRMT are provided in Chapter 7. FDD was found to have been generally useful for the development of ASRMT, and the tool's functionality was verified in term of addressing in principle the research goals identified in earlier chapters. Informal expert evaluations of ASRMT revealed that the tool is likely to provide support for agile project teams. Among the feedback received from ASRMT user evaluations, it was reported that ASRMT is easy to learn and use, relatively bug free, would be useful if used in live projects, and presents a proof of the concepts presented in this research. While these findings upheld ASRMT's value, there are several ways in which this tool could be improved to extend its usefulness. In particular, it was recommended that ASRMT should be extended to include additional functionality such as a discussion feature and automatic e-mail reminders, which are likely to further assist project participants with communication and information sharing.

8.2. Conclusions

Regardless of the methodology employed in software development, evidence shows that this continues to be a very challenging activity. This study found that, while agile methodologies are likely to improve some aspects of software project management, agile methods' human collaboration practices also introduce social risks, and such risks may be deemed most critical. In addressing the first research question, this study showed that, of the risks that are likely to be induced through human collaboration, poor productivity, project overruns, and software failure are highlighted in the literature. While 'standard' risk management theories such as those espoused by the PMI may assist with general software project risk management, this study identified that these theories do not adequately address social risk management.

As a consequence, the effects of stakeholders' interactions were comprehensively examined, and this study proposed to implement a tool to assist with social risks that are likely to increase through these occurrences. The social risk management and psychology literature revealed that personality conflicts and customer disagreements are social risks that are induced through human collaboration, and such risks should be managed as they negatively affect team cohesion and software project success. Previous tools did not consider addressing these risks. ASRMT was successfully designed and built to address this shortcoming, addressing the second research question in the process. This tool offers personality capability management to reduce personality conflicts, and support for remote customer feature management to extend the customers so that they can be directly involved with project management while being physically remote.

This study concludes that a project management tool can be implemented in the agile software development context to address social risk management theories. ASRMT was verified using software engineering experts with different levels of experience. The ASRMT user evaluation findings answered the third research question positively; ASRMT is likely to be useful to agile developers, and should improve their handling of social risks. The findings of the ASRMT user evaluations therefore provide sufficient proof of concept for this research. However, it would be interesting to determine whether this concept would be embraced in live project settings, and by the wider agile community.

8.3. Limitations and Recommendations

Even though the findings of the ASRMT user evaluations are encouraging, there are a few limitations to the tool, and consequently, to the findings of this research project. The personality assessment mechanism used here - the self-scoring Self-Perception Inventory (SPI) - was originally published in Meredith Belbin's book 'Management Teams Why They Succeed or Fail' (1981). Belbin Associates own the copyright for this questionnaire and do not allow it to be reproduced in any form. Furthermore, Belbin Associates no longer recommend the use of this questionnaire, as it is obsolete (no Specialist Role), lacks the balance of observer input, is not properly normed, and most importantly, does not offer any advice (an approved version of the questionnaire can be found at www.belbin.com). While the findings of the ASRMT user evaluations verify and support the idea proposed by this study with respect to personality management, the SPI implemented in ASRMT cannot be used to evaluate individuals or assess team roles, as assessment findings may not be truly representative of individuals' personality preferences. Therefore, accessing the approved version of Belbin's SPI and re-implementing it (or an alternative) in ASRMT, offer an avenue for extending and enhancing the tool.

Additionally, even though the ASRMT user evaluations were meaningful (with targeted representative users), due to resource constraints, ASRMT was not used in the management of real life software projects where there are many project members occupying varying roles, and there is a need to coordinate and manage many concurrent development tasks, perhaps across a portfolio of projects. Further ASRMT user evaluations should therefore be carried out in live project environments. Thus, ASRMT should be enhanced, taking into account the suggestions provided by the participants in the ASRMT initial user evaluations, and then re-evaluated in live project settings.

9. References

Abrahamsson, P., & Koskela, J. (2004). *Extreme programming: a survey of empirical data from a controlled case study.* Paper presented at the Proceedings of the Empirical Software Engineering Symposium, Oulu, Finland.

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). *New directions on agile methods: a comparative analysis.* Paper presented at the Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon.

Acuna, S. T., & Juristo, N. (2004). Assigning people to roles in software projects. *Software Practice and Experience.*

Acuna, S. T., Juristo, N., & Moreno, A. M. (2006). Emphasizing human capabilities in software development. *IEEE Software, 23*(2), 94 - 101.

Agarwal, R., Prasad, J., Tanniru, M., & Lynch, J. (2000). Risks of rapid application development. *Communications of the ACM, 43*(11), Article No.1.

Angioni, M., Carboni, D., Pinna, S., Sanna, R., Serra, N., & Soro, A. (2006). Integrating XP project management in development environments. *Journal of Systems Architecture, 52*(11), 619-626.

Atsuta, S., & Matsuura, S. (2004). *eXtreme Programming support tool in distributed environment.* Paper presented at the Computer Software and Applications Conference.

Augustine, S., Payne, B., Sencindiver, F., & Woodcock, S. (2005). Agile project management: steering from the edges. *Communications of the ACM, 48*(12), 85 - 89.

Barki, H., & Hartwick, J. (2001). Interpersonal conflict and its management in information system development. *MIS Quarterly, 25*(2), 195 - 228.

Barki, H., & Rivard, S. (1993). Toward an assessment of software development risk. *Journal of Management Information Systems, 10*(2), 203 - 225.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change.* Reading, MA: Addison-Wesley Longman, Inc.

Behrens, P. (2006). Trail Ridge Consulting: Agile project management (APM) tooling survey results. Retrieved January 5, 2007, from http://www.trailridgeconsulting.com

Belbin, R. M. (1981). *Management teams: why they succeed or fail.* Woburn, UK: Butterworth-Heinemann.

Belbin, R. M. (2002). *Management teams: why they succeed or fail.* Woburn, UK: Butterworth-Heinemann.

Beranek, G., Zuser, W., & Grechenig, T. (2005). *Functional group roles in Software Engineering teams.* Paper presented at the Workshop on Human and Social Factors of Software Engineering, HSSE'05, St. Louis, Missouri, USA.

Bergsten, H. (2004). *JavaServer Pages.* (Second ed.). CA, USA: O'Rilley and Associates Inc.

Beznosov, K., & Kruchten, P. (2004). *Towards agile security assurance.* Paper presented at the Proceedings of the 2004 workshop on new security paradigms, Nova Scotia, Canada.

Boehm, B. (1976). Software Engineering. *Computers, C-25*(12), 1226 - 1241.
Boehm, B. (2006). *A view of 20th and 21st century software engineering.* Paper presented at the Proceeding of the 28th international conference on Software engineering, Shanghai, China.
Boehm, B., & Turner, R. (2003). Using risk to balance agile and plan-driven methods. *IEEE Journal, 36*(6), 57 - 66.
Boehm, B., & Turner, R. (2005). Management challenges to implementing agile processes in traditional development. *IEEE Journal, 22*(5), 30 - 39.
Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer, 21*(5), 61 - 72.
Boehm, B. W., & DeMarco, T. (1997). Software risk management. *IEEE Software, 14*(3), 17 - 19.
Bostrom, G., Wayrynen, J., Boden, M., Beznosov, K., & Kruchten, P. (2006). *Extending XP Practices to Support Security Requirements Engineering.* Paper presented at the Software Engineering for Secure Systems, Shanghai, China.
Bradley, J. H., & Hebert, F. J. (1997). The effect of personality type on team performance. *Journal of Management Development, 16*(5), 337 - 353.
Brewer, J. L. (2005). *Project managers: can we make them or just make them better?* Paper presented at the Proceedings of the 6th conference on Information technology education, Newark, NJ, USA.
Brooks, F. P. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. *Computer, 20*(4), 10 - 19.
Capilla, R., Nava, F., Perez, S., & Duenas, J. C. (2006). A web-based tool for managing architectural design decisions. *Software Engineering Notes, 31*(5).
Carlson, J. G. (1985). Recent Assessments of the Myers-Briggs Type Indicator. *Journal of Personality Assessment, 49*(4), 356 - 365.
Carr, M. J., Konda, S. L., Monarch, I., Ulrich, C. F., & Walker, C. F. (1993). *Taxonomy-Based Risk Identification.* (Technical Report): Software Engineering Institute, Carnegie Mellon University.
Ceravolo, P., Damiani, E., Marchesi, M., Pinna, S., & Zavatarelli, F. (2003). *A ontology-based process modeling for XP.* Paper presented at the Tenth Software Engineering Conference, Asia.
Chang, C. J., & Ho, J. L. Y. (1997). The effects of justification, task complexity and experience/training. *Behavioral Research in Accounting, 9*, 98 - 116.
Chapman, C., & Ward, S. (2004). *Project Risk Management.* (Second ed.). West Sussex, England: John Wiley & Sons Ltd.
Charette, R. N. (2005). Why software fails. *IEEE Spectrum, 42*(9), 42 - 49.
Chin, G. (2004). *Agile Project Management: How to Succeed in the Face of Changing Project Requirements.* New York: American Management Association.
Clark, D. R. (2006). *Beginning object-oriented programming with VB 2005: From novice to professional.* Berkeley, CA: Apress.
Clark, S. (1998). CSS Simplifies Your Life.   Retrieved November 20, 2006, from http://www.webdeveloper.com/html/html_css_1.html
Clavadetscher, C. (1998). User involvement: key to success. *IEEE Software, 15*(2), 30 - 32.

Cockburn, A. (2004). The End of Software Engineering and the Start of Economic-Cooperative Gaming. *Computer Science and Information Systems, 1*(1), 1 - 32.

Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. *Computer*, *34*(11), 131 - 133.

Collins, H. (2006). *Collins Dictionary and Thesaurus.* Glasgow, GB: HarperCollins Publishers.

Collis, J., & Hussey, R. (2003). *Business Research: A practical guide for undergraduate and postgraduate students.* (Second ed.). New York: Palgrave Macmillan.

Connell, J. L., & Shafer, L. (1989). *Structured Rapid Prototyping: An Evolutionary Approach to Software Development.* New Jersey: Yourdon Press.

Coram, M., & Bohner, S. (2005). *The impact of agile methods on software project management.* Paper presented at the Engineering of Computer-Based Systems, 2005. ECBS '05, USA.

Curtis, B. (1989). *Three Problems Overcome With Behavioral Models Of The Software Development Process.* Paper presented at the 11th International Conference on Software Engineering.

Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM, 31*(11), 1268 - 1287.

Damian, D., Lanubile, F., & Mallardo, T. (2006). *The role of asynchronous discussions in increasing the effectiveness of remote synchronous requirements negotiations.* Paper presented at the Proceeding of the 28th international conference on software engineering.

David, B. A. (1991). Vulnerability and Agenda: Context and Process in Project Management. *British Journal of Management, 2*(3), 121 - 132.

De Luca, J. (2002). Feature Driven Development.   Retrieved November 26, 2006, from http://www.featuredrivendevelopment.com/

De Souza, S., Anquetil, N., & De Oliveira, K. (2005). *A study of the documentation essential to software maintenance.* Paper presented at the Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information, Coventry, United Kingdom.

Denning, P. J. (1997). A New Social Contract for Research. *Communications of the ACM, 40*(2), 132 - 134.

Dixon, M. (2000). *The association for project management (APM) body of knowledge(BoK).* (Fourth ed.). High Wycombe, UK: Assoc. Project Management.

Faraj, S., & Sproul, L. (2000). Coordinating expertise in software development teams. *Journal of Management Science, 46*(12), 1554 - 1568.

Flanagan, D. (2001). *JavaScript: The Definitive Guide.* (Fourth ed.). CA, USA: O'Rilley and Associates Inc.

Foster, S. T., & Franz, C. R. (1999). User involvement during information systems development: a comparison of analyst and user perceptions of system acceptance. *Journal of Engineering and Technology Management, 16*, 329 - 348.

Fox, P. (2002). Tapping the right tools. *Computer World, 36*(17), 43.

Fox, T. L., & Wayne Spence, J. (2005). The effect of decision style on the use of a project management tool: an empirical laboratory study. *Advances in Information Systems*, *36*(2), 28 - 42.

Gallagher, B. P. (1999). *Software Acquisition Risk Management Key Process Area (KPA)--A Guidebook Version 1.02.* (SEI Technical Report). Pittsburg PA: Carnegie Mellon University.

Gallagher, B. P., Alberts, C., & Barbour, R. (1997). *Software Acquisition Risk Management Key Process Area (KPA)--A Guidebook Version 1.0.* (SEI Technical Report). Pittsburg PA: Carnegie Mellon University.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software.* Reading, MA: Addison-Wesley.

Gilb, T. (1988). *Principles of software engineering management.* Wokingham, UK: Addison-Wesley.

Glass, R. (1999). On Design. *IEEE Software*, *16*(2), 103 - 104.

Globerson, S., & Zwikael, O. (2002). The Impact of the Project Manager on Project Management Planning Processes. *Project Management Journal, 33*(3), 58 - 65.

Goodman, P. S., & Fichman, M. (1995). Customer-firm relationships, involvement, and customer satisfaction. *Academy of Management Journal, 38*(5), 1310 - 1324.

Gordon, V., & Bieman, J. (1993). Reported effects of rapid prototyping on industrial software. *Software Quality Journal, 2*(2), 93 - 108.

Gorla, N., & Lam, Y. W. (2004). Who should work with whom? *Communications of the ACM, 47*(6), 79 - 82.

Grisham, S. P., & Perry, E. D. (2005). *Customer relationships and Extreme Programming.* Paper presented at the Proceedings of the 2005 workshop on Human and social factors of software engineering.

Heinbokel, T., Sonnentag, S., Frese, M., Stolte, W., & Brodbeck, F. C. (1996). Don't underestimate the problems of user centeredness in software development projects - there are many! *Behavior & Information Technology, 15*(4), 226 - 236.

Hevner, A., March, T. S., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly, 28*(1), 75 - 105.

Highsmith, J. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems.* New York: Dorset House Publishing.

Highsmith, J. (2004). *Agile Project Management: Creating Innovative Products.* Boston, MA: Pearson Education, Inc.

Highsmith, J., & Cockburn, A. (2001). Agile software development: the business of innovation. *Computer, 34*(9), 120 - 122.

Hodgson, D. E. (2004). Project Work: The Legacy of Bureaucratic Control in the Post-Bureaucratic Organization. *Organization, 11*(1), 81 - 100.

Hofstede, G., Neuijen, B., Ohayv, D. D., & Sanders, G. (1990). Measuring Organizational Cultures: A Qualitative and Quantitative Study Across Twenty Cases. *Administrative Science Quarterly, 35*(2), 286.

Hulkko, H., & Abrahamsson, P. (2005). *A multiple case study on the impact of pair programming on product quality.* Paper presented at the International Conference on Software Engineering, St. Louis, MO, USA.

Humphrey, W. S. (1989). *Managing the software process.* Reading, MA: Addison-Wesley.

Hunt, A., & Thomas, D. (1999). *The pragmatic programmer: From journeyman to master.* Reading, MA: Addison-Wesley.

Jiang, G., & Chen, Y. (2004). *Coordinate metrics and process model to manage software project risk.* Paper presented at the Engineering Management Conference, Changsha, China.

Jiang, J. J., Klein, G., & Balloun, J. (1996). Ranking of system implementation success factors. *Project Management Journal, 27*(4), 50 - 55.

Johnson, D. A. (1992). Test-retest reliabilities of the Myers-Briggs Type Indicator and the Type Differentiation Indicator over a 30 month period. *Journal of Psychological Type, 24*, 54 - 61.

Jones, A. (1999). Experience of profile-based group composition. *Computer Science Education, 9*(3), 242 - 255.

Jorgensen, M., & Molokken-Ostvold, K. (2006). How large are software cost overruns? A review of the 1994 CHAOS report. *Information and Software Technology, 48*(4), 297 - 301.

Jung, C. (1971). *Psychological types.* (Vol. 6). New Jersey: Princeton University Press.

Jurgen, H. (2000). Realistic Criteria for Project Manager Selection and Development. *Project Management Journal, 31*(3), 23 - 32.

Kaariainen, J., Koskela, J., Abrahamsson, P., & Takalo, J. (2004). *Improving requirements management in extreme programming with tool support - an improvement attempt that failed.* Paper presented at the Euromicro Conference Proceedings, Euromicro.

Karolak, D. W. (1998). *Software Engineering Risk Management: Finding Your Path through the Jungle*: Wiley-IEEE Computer Society Press.

Keirsey, D. (1998). *Please Understand Me II: Temperament Character Intelligence.* CA: Prometheus Nemesis Book Company.

Kelter, U., Monecke, M., & Schild, M. (2003). Do we need 'Agile' software development tools? *Springer Berlin / Heidelberg, 2591*, 412 - 430.

Kim, H. W. (2006). Towards a process model of information systems implementation: The case of customer relationship management (CRM). *Advances in Information Systems, 37*(1), 59 - 76.

Kirakowski, J. (2000). *Questionnaires in usability engineering: a list of frequently asked questions.* Cork, Ireland: Human Factors Research Group.

Kirk, D., & Tempero, E. (2006). *Identifying Risks in XP Projects through Process Modeling.* Paper presented at the Software Engineering Conference, Australian.

Kitchenham, B., & Pfleeger, S. L. (2002). Principle of Survey Research- Part5: Populations and Samples. *Software Engineering, 27*(5), 17 - 20.

Klein, S. A. (1999). *Putting methodology in perspective from a project risk viewpoint.* Paper presented at the IEEE Power Engineering Society Winter Meeting.

Koch, A. (2005). *Agile Software Development: Evaluating the methods for your organisation.* Boston, MA: Artech House.

Kontio, J. (2001). *Software Engineering Risk Management: A method, Improvement Framework, and Empirical Evaluation.*, Helsinki University of Technology.

Kontio, J., Hoglund, M., Ryden, J., & Abrahamsson, P. (2004). *Managing commitments and risks: challenges in distributed agile development.* Paper presented at the ICSE.

Koskela, L., & Howell, G. (2002). *The underlying theory of project management is obsolete.* Paper presented at the Proceedings of the PMI Research Conference, Professional Management Institute, Seattle, USA.

Kuppuswami, S., Vivekanandan, K., Ramaswamy, P., & Rodrigues, P. (2003). The effects of individual XP practices on software development effort. *ACM SIGSOFT Software Engineering, 28*(6), 1 - 6.

Lan, C., & Peng, X. (2005). *Activity Patterns of Pair Programming.* Paper presented at the Proceedings of the 38th Annual Hawaii International Conference on System Sciences.

Law, A., & Charron, R. (2005). *Effects of agile practices on social factors.* Paper presented at the Proceedings of the 2005 workshop on Human and social factors of software engineering, St. Louis, Missouri.

Lehman, M. M., & Ramil, J. F. (2001). Rules and Tools for Software Evolution Planning and Management. *Annals of Software Engineering, 11*(1), 15 - 44.

Leuf, B., & Cunningham, W. (2001). *The Wiki Way.* New Jersey: Addison-Wesley.

Lewis, J. R. (1995). IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use. *International Journal of Human-Computer Interaction, 7*(1), 57 - 78.

Linebarger, J. M., Scholand, A. J., Ehlen, M. A., & Procopio, M. J. (2005). *Benefits of synchronous collaboration support for an application-centered analysis team working on complex problems: A case study.* Paper presented at the Proceedings of the 2005 international ACM conference on supporting group work.

Markus, M. L., Majchrzak, A., & Gasser, L. (2002). A Design Theory for Systems that Support Emergent Knowledge Processes. *MIS Quarterly, 26*(3), 179 - 212.

Maurer, F., & Martel, S. (2002). *Process Support for Distributed Extreme Programming Teams.* Paper presented at the ICSE Workshop on Global Software Development.

McCarley, N., & Carskadon, T. (1983). Test-retest reliabilities of scales and subscales of the Myers-Briggs Type Indicator and of criteria for clinical interpretive hypotheses involving them. *Research in Psychological Type, 6*, 24 - 36.

McCrae, R. R., & Costa, P. T. (1990). *Personality in adulthood.* New York: The Guildford Press.

McKeen, J. D., & Guimaraes, T. (1997). Successful strategies for user participation in systems development. *Journal of Management Information Systems, 14*(2), 133 - 150.

McKinney, D., & Denton, F. L. (2005). *Affective assessment of team skills in agile CS1 labs: the good, the bad, and the ugly.* Paper presented at the Proceedings of the 36th SIGCSE technical symposium on Computer science education, St. Louis, Missouri, USA.

Melymuka, K. (2000). Born to lead projects. *Computerworld, 34*(13), 62 - 64.

Meredith, J. R., & Mantel, S. J. (2003). *Project Management: A managerial approach.* (Fifth ed.). New York: John Wiley and Sons Inc.

Misic, V. (2006). Perceptions of extreme programming: an exploratory study. *ACM SIGSOFT Software Engineering, 31*(2), 1 - 8.

Montgomery, S. (2002). *People Patterns: A Modern Guide to the Four Temperaments.* CA: Archer Publications.

Moroney, L., & MacDonald, M. (2006). *Pro ASP.NET 2.0 in VB 2005: Create next-generation web applications with the latest version of Microsoft's revolutionary technology*. Berkeley, CA: Apress.

Morski, J., & Miler, J. (2002). *Towards an integrated environment for risk management in distributed software projects.* Paper presented at the 7th European Conference on Software Quality, Helsinki, Finland.

Moynihan, T. (1997). How experienced project managers assess risk. *IEEE Software, 14*(3), 35 - 41.

Murthi, S. (2002). Preventive risk management software for software projects. *IT Professional, 4*(5), 9 - 15.

Myers, I., & McCaulley, M. (1985). *Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator*. Palo Alto, CA: Consulting Psychologists Press.

Naur, P., & Randell, B. (1968). *Software Engineering: Report of a conference sponsored by the NATO Science Committee.* Paper presented at the NATO Science Conference, Garmisch, Germany.

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM, 48*(5), 72 - 78.

Newman, M., & Robey, D. (1992). A social process model of user-analyst relationships. *MIS Quarterly, 16*(2), 249 - 266.

Nord, R. L., & Tomayko, J. E. (2006). Software Architecture-Centric Methods and Agile Development. *IEEE Software, 23*(2), 47 - 54.

Nosek, J. T. (1998). The case for collaborative programming. *Communications of the ACM, 41*(3), 105 - 108.

Olsen, L. (2006). *Being There: A "teach them to fish..." approach to training and support using WebEx, videoconferencing, and the telephone.* Paper presented at the Proceedings of the 34th annual ACM conference on user services.

Oriogun, P. K. (1999). A Survey of Boehm's Work on the Spiral Models and COCOMO II--Towards Software Development Process Quality Improvement. *Software Quality Journal, 8*(1), 53 - 62.

P.M.Institute. (2000). *A Guide to the Project Management Body of Knowledge (PMBOK Guide).* USA: Project Management Institute Inc.

Packendorff, J. (1995). Inquiring into the temporary organization: New directions for project management research. *Scandinavian Journal of Management, 11*(4), 319 - 333

Palmer, S. R., & Felsing, J. M. (2002). *A practical guide to Feature-Driven Development.* Upper Saddle River, New Jersey: Prentice-Hall.

Parker, R. (2002). Poor IT purchase planning 'wastes $500bn worldwide'. *Supply Management, 7*(8), 11.

Perry, D. E., & Wolf, A. L. (1992). Foundations for the study of software architecture. *Software Engineering Notes, 17*(4), 40 - 52.

Pressman, R. S. (2001). *Software engineering: A practitioner's approach.* (Fifth ed.). New York: McGraw-Hill Companies Inc.

Rajendran, M. (2005). Analysis of team effectiveness in software development teams working on hardware and software environments using Belbin Self-Perception inventory. *Journal of Management Development, 24*(8), 738 - 753.

Randell, B., & Buxton, J. (1969). *Software Engineering Techniques: Report of a conference sponsored by the NATO Science Committee.* Paper presented at the NATO Science Conference, Rome, Italy.

Rees, M. J. (2002). *A feasible user story tool for agile software development?* Paper presented at the ninth Asia-Pacific Software Engineering Conference, Asia.

Rowe, A. J., & Mason, R. O. (1987). *Managing with style.* San Francisco: Jossey-Bass.

Roy, G. G. (2004). *A risk management framework for software engineering practice.* Paper presented at the Australian Software Engineering Conference, Australia.

Ryder, B. G., Soffa, M. L., & Burnett, M. (2005). The impact of software engineering research on modern progamming languages. *Transactions on Software Engineering and Methodology, 14*(4), 431 - 477.

Schoenthaler, F. (2002). *Risk management in challenging business software projects.* Paper presented at the IEEE Conference on Requirements Engineering.

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum.* Upper Saddle River, New Jersey: Prentice-Hall.

Serich, S. (2005). Prototype stopping rules in software development projects. *Transactions on Engineering Management, 52*(4), 478 - 485.

Sharp, H., Robinson, H., & Segal, J. (2004). *Customer collaboration: successes and challenges in practice systems.* (Technical Report No. TR2004/10): Computing Department, The Open University.

Siddiqui, M. S., & Hussain, S. J. (2006). *Comprehensive Software Development Model.* Paper presented at the IEEE International Conference on Computer Systems and Applications.

Silver, M. S., Markus, M. L., & Beath, C. M. (1995). The Information Technology Interaction Model: A Foundation for the MBA Core Course. *MIS Quarterly, 19*(3), 361 - 390.

Sommerville, I. (1997). *Software Engineering.* Reading, MA: Addison-Wesley.

Standish Group. (1995). The Chaos Report. *West Yarmounth, MA: The Standish Group* Retrieved May 1, 2006, from http://www.standishgroup.com/sample_research/PDFpages/chaos1994.pdf

Standish Group. (2001). Extreme Chaos. *West Yarmounth, MA: The Standish Group* Retrieved May 1, 2006, from http://www.standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf

Swink, M. (2002). Product development--faster, on-time. *Research Technology Management, 45*(4), 50 - 58.

Tesch, D., Jiang, J. J., & Klein, G. (2003). The impact of information system personnel skill discrepancies on stakeholder satisfaction. *Decision Sciences, 34*(1), 107 - 127.

Theunissen, M. H. W., Kourie, G. D., & Watson, W. B. (2003). *Standards and agile software development.* Paper presented at the ACM International Conference Proceeding Series, South African.

Tolvanen, J. P. (1998). *Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence.* Unpublished Doctoral, University of Jyvaskyla, Finland.

VanDeursen, A. (2001). *Program comprehension risks and opportunities in extreme programming.* Paper presented at the Reverse Engineering, 2001, Proceedings of the Eighth Working Conference Stuttgart.

Verner, J. M., & Evanco, W. M. (2005). In-house software development: what project management practices lead to success? *IEEE Software, 22*(1), 86 - 93.

VersionOne. (2006a). Agile tool evaluator guide. Retrieved October 10, 2006, from http://www.versionone.com

VersionOne. (2006b). The state of agile development. Retrieved November 12, 2006, from http://www.versionone.com

Warrene, B. (2004). Hardening Apache - A Conversation with the Author. Retrieved November 20, 2006, from http://www.sitepoint.com/article/hardening-apache

Wellington, C., Briggs, T., & Girard, D. (2005). *Examining team cohesion as an effect of software engineering methodology.* Paper presented at the Proceedings of the 2005 workshop on Human and social factors of software engineering, St. Louis, Missouri.

Williams, L. (2003). The XP Programmer: The few minutes programmer. *IEEE Software, 20*, 16 - 20.

Williams, L., & Cockburn, A. (2003). Guest Editors' Introduction: Agile Software Development: It's about Feedback and Change. *Computer, 36*(6), 39 - 43.

Williams, L., & Kessler, R. (2003). *Pair Programming Illuminated.* Boston: Addison-Wesley.

Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Software, 17*(4), 19 - 25.

Williams, T. (2005). Assessing and moving on from the dominant project management discourse in the light of project overruns. *Engineering Management: IEEE Journal, 52*(4), 497 - 508.

Winch, G., Millar, C., & Clifton, N. (1997). Culture and Organization: The Case of Transmanche-Link. *British Journal of Management, 8*(3), 237 - 249.

Zmud, R. (1997). Editors Comments. *MIS Quarterly, 21*(2), xxi - xxii.

10. Appendices

Appendix A. Deploying ASRMT

Deploying ASRMT can be as simple as copying the directory structure (ASRMT) and the associated files to a web server. Once this is done, simple configuration to a database and windows user security settings should get ASRMT up and running. Apart from the implementation of styles, ASRMT was developed using Microsoft software tools. Using Windows XP professional operating system, IIS was used as the web server, Visual Studio was used to create ASP.NET web pages, and SQL Server 2005 supported data storage. The remaining sections of this annotation give detail of how the preceding software can be configured to deploy ASRMT.

Pre-requisites

The ASRMT tool requires the following to be installed prior to its installation:

1. Windows 2003 Server, Windows 2000, or Windows XP Professional operating system.
2. Microsoft SQL Server 2005.
3. ASP.NET.
4. Internet Information Services (IIS V5.1).

Extracting ASRMT compressed archive

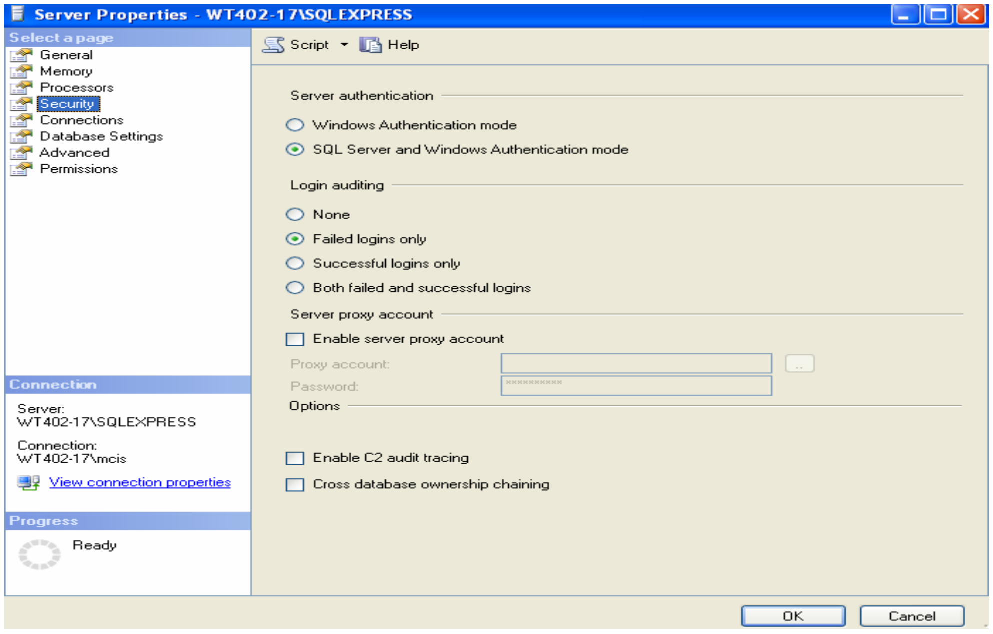
Extract the ASRMT.rar file using WinRar to the desired application directory where you wish to install ASRMT (Example: C:\ASRMT). After extracting ASRMT.rar, the following sub-directories will be placed in the ASRMT directory:

1. /ASRMT – Contains the ASRMT application.
2. /ASRMT_DB – Contains the database files.
3. /ASRMT_Help – Contain the ASRMT installation manual and tutorial files.

Configuring ASRMT with Microsoft SQL Server 2005

1. Create a windows system user ‘asp’ having password ‘asp’. If the system user is created with another username and password, you will need to update the identity tag (<identity impersonate="true" userName="asp" password="asp" />) in the web.config file located in the ASRMT directory with your username and password details.
2. Connect to the SQL Server – Click on SQL Server Management Studio Express in the SQL Server Start Menu, Click Connect.
3. Configure the SQL Server mixed mode option – Select the SQL object explorer, Right-click on the server explorer and select properties, select the SQL Server and Windows Authentication Mode radio button in the security page, Click ok (see Figure 10.1 for details).
4. Ensure the ‘asp’ user is assigned RW permission to the ASP.NET Temp folder.
5. Attach the ASRMT database (see Figure 10.2 – Right-click on the Database explorer and select Attach Database, Select the Add Button in the Attach Database Page, Select the location of the ASRMT database (Example: C:\ASRMT\ASRMT_DB\CDAPMT.mdf), Click ok. **For SQL consistency, you can copy the CDAPMT.mdf and CDAPMT_log.ldf file from the C:\ASRMT\ASRMT_DB directory to your local SQL’s default data location, and then attach the ASRMT database from there.**
6. The ASRMT tool can now communicate with the database.

Figure 10.1. Security object tab of Server Properties


Server Properties - WT402-17\SQLEXPRESS

Select a page: General, Memory, Processors, Security, Connections, Database Settings, Advanced, Permissions

Server authentication
- Windows Authentication mode
- [✓ SQL Server and Windows Authentication mode]

Login auditing
- None
- [✓ Failed logins only]
- Successful logins only
- Both failed and successful logins

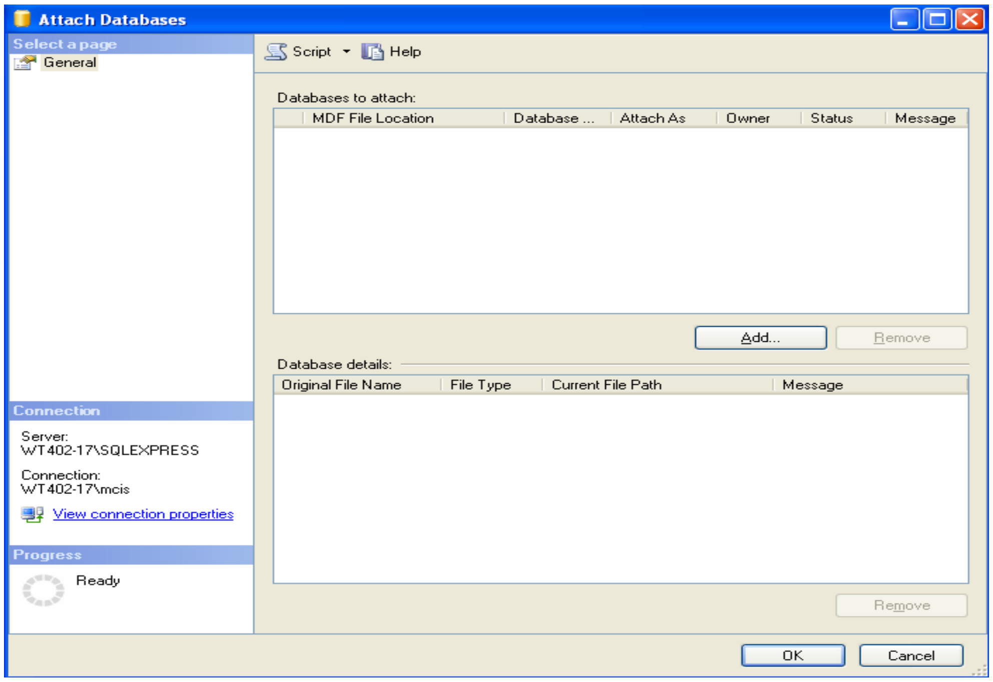
Server proxy account
- Enable server proxy account
- Proxy account:
- Password:

Options
- Enable C2 audit tracing
- Cross database ownership chaining

Connection
Server: WT402-17\SQLEXPRESS
Connection: WT402-17\mcis
View connection properties

Progress
Ready

Figure 10.2. Attaching the ASRMT database


Attach Databases

Select a page: General

Databases to attach:

MDF File Location	Database ...	Attach As	Owner	Status	Message

Database details:

Original File Name	File Type	Current File Path	Message

Connection
Server: WT402-17\SQLEXPRESS
Connection: WT402-17\mcis
View connection properties

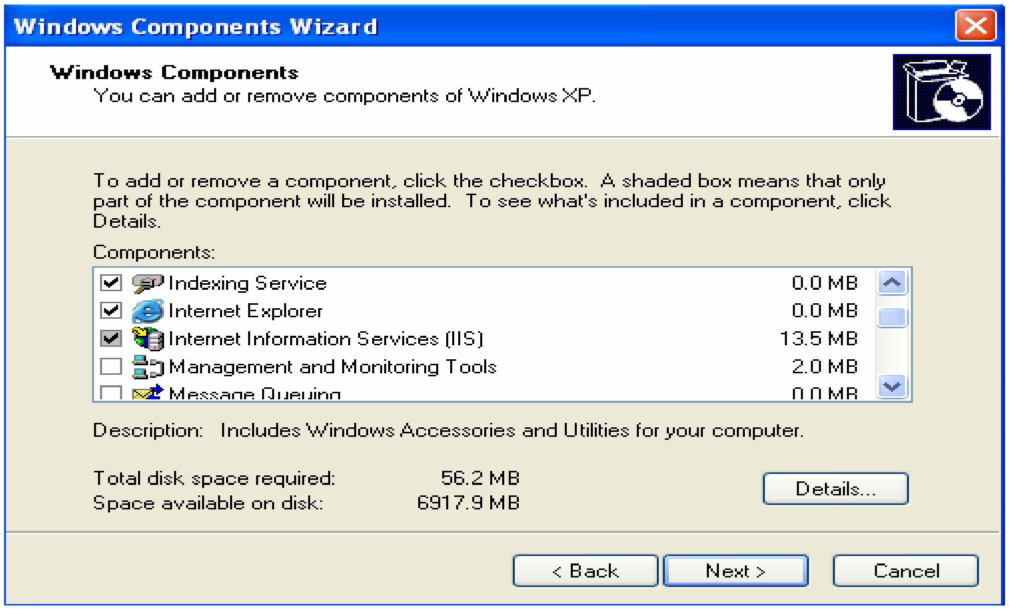
Progress
Ready

Configuring IIS for ASRMT

ISS is included with windows; however it is not installed with a default windows installation. While it is recommended that a server version of windows should be installed to publish web application, IIS can be setup on XP professional to allow up to 10 concurrent user connections. The following explains the steps for configuring ISS on windows 2000, or windows XP professional:

1. Click start, select settings – Control Panel.
2. Choose Add or Remove Programs.
3. Click Add/Remove Windows Components.
4. If Internet Information Services is checked (see Figure 10.3), you already have this component installed. If not, check IIS and click next to install the required ISS files; you may be prompted for your windows setup CD.

Figure 10.3. IIS is currently installed



Windows Components
You can add or remove components of Windows XP.

To add or remove a component, click the checkbox. A shaded box means that only part of the component will be installed. To see what's included in a component, click Details.

Components:

✓	Indexing Service	0.0 MB
✓	Internet Explorer	0.0 MB
▨	Internet Information Services (IIS)	13.5 MB
☐	Management and Monitoring Tools	2.0 MB
☐	Message Queuing	0.0 MB

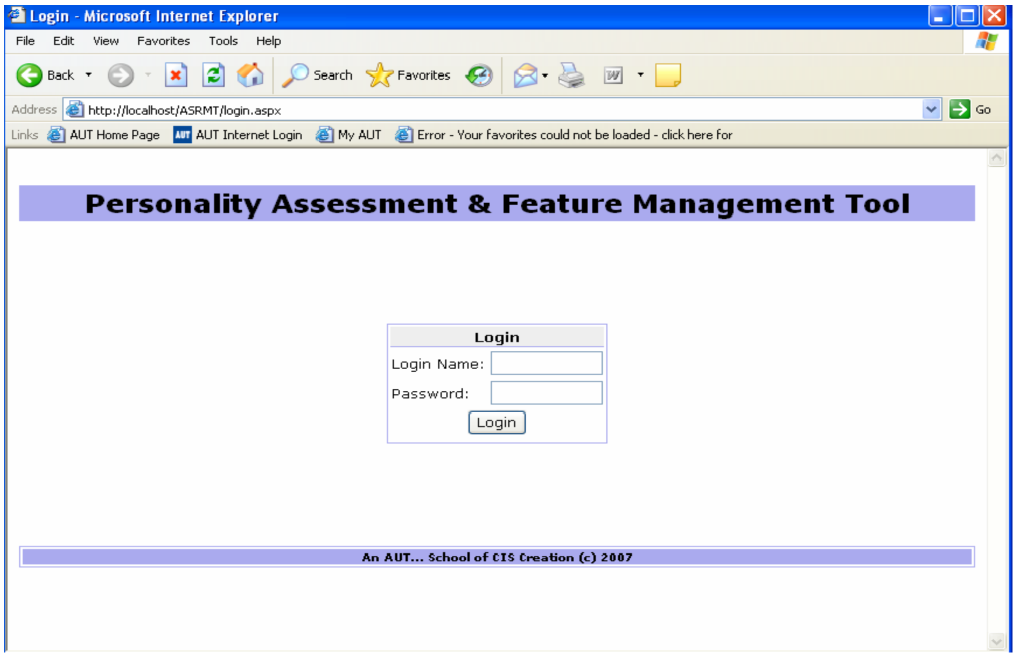
Description: Includes Windows Accessories and Utilities for your computer.

Total disk space required: 56.2 MB
Space available on disk: 6917.9 MB

You have two options for deploying ASRMT using IIS:

1. When IIS is installed, the C:\Inetpub\wwwroot directory is automatically created on your computer. This directory represents your website. To configure ASRMT with IIS you can copy the ASRMT directory directly into the c:\Inerpub\wwwroot directory. You can then test ASRMT by requesting it in a browser using the URL http://localhost/ASRMT. You should see the ASRMT login screen (see Figure 10.4).

Figure 10.4. ASRMT login screen


Login - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/ASRMT/login.aspx
Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for

Personality Assessment & Feature Management Tool

Login

Login Name:

Password:

Login

An AUT... School of CIS Creation (c) 2007

2. The technique used above for configuring ASRMT is uncomplicated. However, if you are using the wwwroot directory to organise other files this mode of configuration may become disorganised. Therefore, ASRMT can be more efficiently configured through IIS virtual directory. The following steps are required to configure ASRMT using an IIS virtual directory:

1. Click start, select settings – Control Panel.
2. Choose Administrative Tools.
3. Click Internet Information Services from the Start Menu.
4. Right-click the Default Website item in the IIS tree.
5. Choose New – Virtual Directory, a wizard will start to manage the process.
6. The wizard requires three (3) sets of information (Alias, Directory, and Permission).
7. For Alias type ASRMT.
8. For directory select the ASRMT physical directory on the hard drive that will be exposed as the virtual directory (Example: C:\ASRMT\ASRMT).

9. For Permission check ‘Read’ and ‘Run Scripts’.
10. Click Next, then Finish.

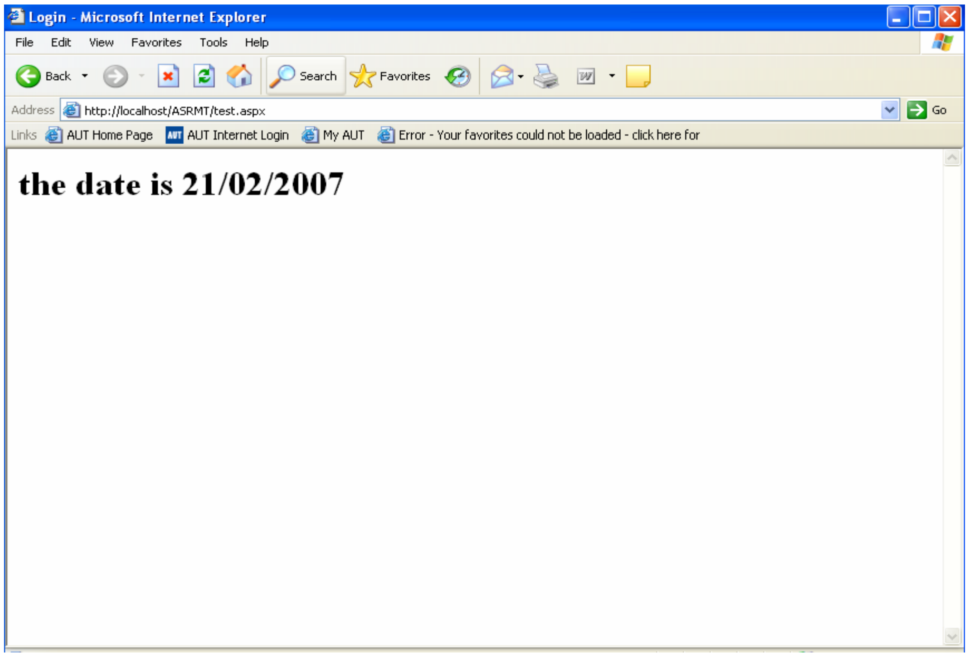
ASRMT can now be accessed using the URL http://localhost/ASRMT. If your internet browser does not open the login page (see Figure 10.4) you should restart your web server using the Default Website icon in the IIS tree. To verify that your ASP.NET installation is working, enter the URL http://localhost/ASRMT/test.aspx in your browser, you should see Figure 10.5. If the text ‘the date is’ appears without the local system date, you may have to repair IIS using the following utility:

C:\[WinDir]\Microsoft.NET\Framework\[Version]\aspnet_regiis.exe –i
(See http://support.microsoft.com/_default.aspx?scid=kb;en-us;325093 for further details regarding IIS troubleshooting).

Note: Before you can connect to the ASRMT database (log on and use ASRMT) you must change the Data Source name in the connection string tag located in the C:\ASRMT\ASRMT\web.config file. The Data Source name should be the name of the instance of SQL server installed on your computer.

Example:
```
<connectionStrings>
        <add name="ASRMT" connectionString = "Data Source= SQL;
         Initial Catalog =ASRMT; Integrated Security=SSPI"/>
</connectionStrings>
```

Figure 10.5. ASRMT test page

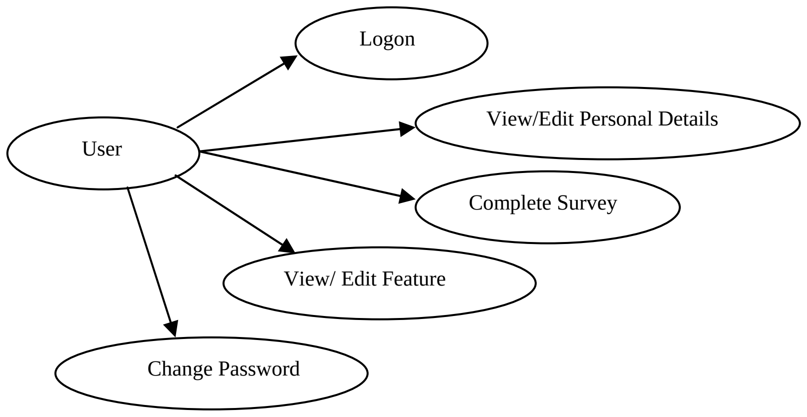

Login - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/ASRMT/test.aspx Go
Links AUT Home Page AUT Internet Login My AUT Error - Your favorites could not be loaded - click here for

the date is 21/02/2007

Appendix B. ASRMT Requirements Specification

<u>User</u>

Figure 10.6. UML use case diagram depicting user functionalities



- Only registered users can logon to the system.
- Users can view and edit their personal details as desired.
- Users must complete the Belbin self-perception inventory (SPI) survey; this survey should automatically assess users 2 major personality traits.
- Users can view and edit features.
- Users can also change their password.

The following notes further describe the preceding use case diagram in Figure 10.6:

Logon - Description: A registered user can/is able to logon to the system. The user must enter a username and password for validation. Users with invalid username and password should not be allowed to logon.
Pre-conditions: User must be registered.
Post-conditions: User can complete the Belbin SPI survey and proceed and use other system resources.

View/Edit Personal Details - Description: A user views or edits their personal details. The logged on user can modify their personal details.
Pre-conditions: User must be logged on to the system.
Post-conditions: User can edit, and save changes to their personal details.

Complete Survey - Description: A user completes Belbin's (SPI) survey. The user must complete Belbin's SPI survey as a part of the registration process in order to assist

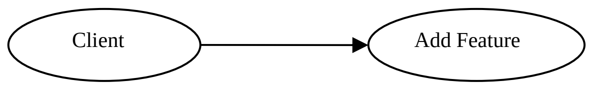
project team with decision making. After the survey is completed the system stores summary information regarding the user's major personality traits.
Pre-conditions: User must be logged on to the system.

View / Edit Feature - Description: A user views or edits a previously entered feature request. Users can view all features that are a part of their project; Users can edit all features that are a part of their project; Users belonging to the client category can only edit features if the status is Requested, Estimated, or Scheduled (a log of all the client feature changes should be maintained).
Pre-conditions: User must be logged on to the system, User must be a member of a project, and Client previously made a feature request.
Post-conditions: Feature is updated with user changes.

Change Password - Description: A user changes their password.
Pre-conditions: User must be logged on to the system.
Post-conditions: User can save new password.

<u>Client</u>

Figure 10.7. UML use case diagram depicting the client functionalities



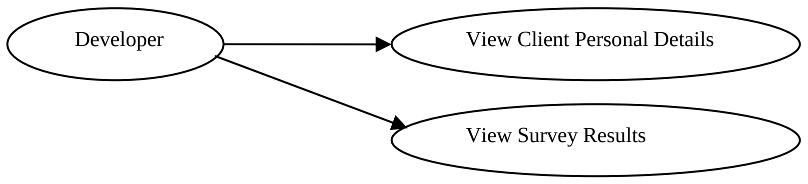
- Clients can add a feature request (example: for XP projects, features are entered as stories) for the project developers. For this they must be part of specific project teams; the client selects the required project, inputs the feature description (this serves as the feature name), inputs the feature details (an area for additional feature information), selects the business value (low business value, moderate business value, significant business value), selects the priority (high, medium, or low), selects the type of feature (New Feature, Defect, or Enhancement) and whether the feature should be discussed.

The following notes further describe the preceding use case diagram in Figure 10.7:

Add Feature Request - Description: A client adds a feature request to the development team (clients are only allowed to add features for projects that they are a member of). The feature request is entered as a simple statement (example: create GUI to allow users to select items for sales).
Pre-conditions: Client must be logged on to the system.
Post-conditions: Client can save feature, Feature is added to feature set for project.

<u>Developer</u>

Figure 10.8. UML use case diagram depicting the developer functionalities



- Developer can view individual client’s or all clients’ contact details once they share the same project with clients.
- Developer can view their survey results at any given time, but should be restricted from viewing others’ SPI survey information.

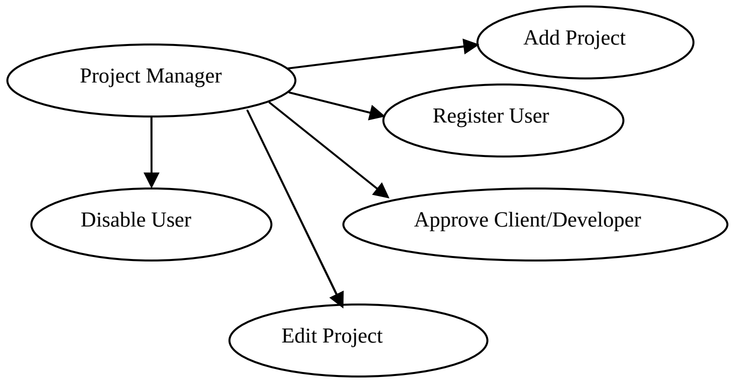
The following notes further describe the preceding use case diagram in Figure 10.8:

View Client Personal Details - Description: A developer views previously entered client personal information. The developers are only allowed to view client personal details that are a part of their projects.
Pre-conditions: Developer must be logged on to the system, for developer to view client details client must also be registered.

View Survey Information - Description: A developer views their personality information assessed by Belbin’s SPI survey. After the developer completes Belbin’s SPI survey, they can view this information anytime in the future. Developers are restricted to viewing their information only.
Pre-conditions: Developer must be logged on to the system, and developer previously completed Belbin’s SPI survey.

<u>Project Manager</u>

Figure 10.9. UML use case diagram depicting the project manager functionalities



- Project Manager can add software project to the system. To add a software project, project manager must indicate project name, project description, and start date.
- Project Manager can register new users (client and developers).
- Project Manager can approve clients and developers, by adding them to specific projects. By adding developers and clients to projects, they are allowed to manage features specific to those projects.
- Project Manager can disable users from the system.
- Project Manager can edit software project information and enter project completion date.
- Project managers also have developer's functionalities.

The following notes further describe the preceding use case diagram in Figure 10.9:

Add Project - Description: A project manager adds software projects to the system. Project managers are allowed to add details for software projects to the system.
Pre-conditions: Project manager must be logged on to the system.
Post-conditions: After projects are completed, the project manager can enter a project completed date.

Register User - Description: A project manager registers users to use the system. The project manager must complete a draft registration for the clients and developers.
Pre-conditions: Project manager must be logged on to the system.
Post-conditions: User is registered and able to logon to the system (to complete the Belbin SPI survey), Clients and developers are approved by the project manager after they complete the Belbin SPI survey.

Approve Client/Developer - Description: A project manager approves a client or a developer. After a client or a developer is registered, and completed the Belbin SPI survey, the client or developer must be approved by the project manager before they are allowed to interface with projects. Approval is done when the project manager adds the client or developer to particular projects. The client or developer can only interface with projects with which they were added.
Pre-conditions: Client or Developer must be registered, and Project manager previously added projects.

Disable User - Description: A project manager disables previously entered users from the system. Project managers are allowed to disable users (clients, and developers) from the system.
Pre-conditions: Client or developer personal details must exist.

Edit Project - Description: A project manager edits software projects. Project managers are allowed to edit the details of software projects and enter a project completion date.
Pre-conditions: Project data must exist.
Post-conditions: Project manager can save changes to the project.

Notes:

- For feature updates, developers can enter the feature start date, the technical risks, date estimated, estimated hours for completion, date scheduled, actual hours taken for completion, and date completed (feature status should automatically change to reflect the feature different states when feature information is updated - Estimated, Scheduled, In Progress, or Completed).
- Project Manager is allowed access to view all clients contact details if the project manager is logged on to the system.
- Project Manager should be allowed to view any developer's or group of developers' contact details, or any client's SPI assessment details at any time (this feature should be offered in the project summaries section).
- New project managers must log in as the administrator then complete the registration form. When a project manager registers to use the system they are also given administrative privilege.

<u>Feature Summary</u>

- All features for Specific project
- All features of a specific Feature Type (New Feature, Defect, or Enhancement) for each project
- All features of a specific Priority (High, Medium, or Low) for each project
- All features of a specific Risk Rating (High, Medium, or Low) for each project
- All features of a specific Status (Requested, Estimated, Scheduled, In Progress, or Completed) for each project

Project Summary

- Summary of All projects
- Summary for Specific project
- Summary of Completed projects
- Summary of Outstanding projects

SPI Summary

- Summary by Category
- Summary by Role
- Summary by User

Project and Feature summaries should be driven by a date range.

Figure 10.10. Domain Model illustrating the relationships among system objects

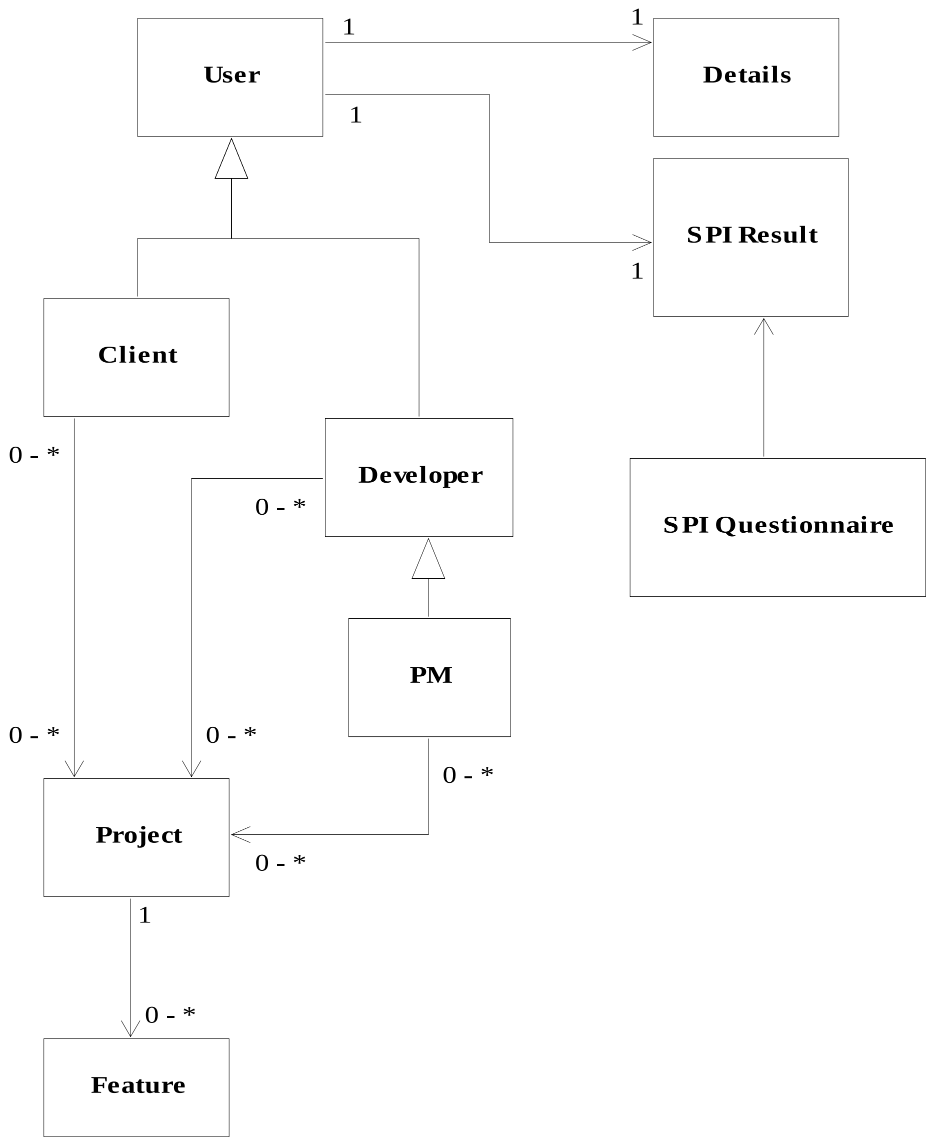


Table 10.1. Summary of System Features and Time Estimate

User Activity	ID	Subject Area (Feature(s))	Task (s) to enable activity.	Estimated Time (day (s))
System Security	P001	Store registration data for users.	Create GUIs to capture new users' personal information. Create tables in a database to store information. Create classes to write data to the database.	4
	P002	Log on authenticated system users.	Create GUIs to capture user login and password information. Create classes to validate user.	3
	P003	Allow Viewing and Editing of Personal Information by users.	Create GUIs to capture user previously stored personal information. Create classes to retrieve user previously stored personal information. Create classes to update changes made by users to their personal information.	4
	P004	Secure Passwords by allowing change password functionality for users.	Create GUI to capture password information. Create classes to update password data.	1
	P005	Disable Users functionality should exist for the project managers.	Create GUI to capture users' information. Create classes to retrieve previously added users. Create classes to update user database for disabled users.	2
Team Collaboration and Feature Management	P006	Add Project functionality should exist for the project managers.	Create GUI to capture project information. Create table in the database to store project information. Create classes to save project information.	2
	P007	Edit Project functionality should exist for the project managers.	Create GUI to capture project information. Create classes to retrieve previously added project information. Create classes to update project information if there are changes.	2
	P008	Add/Approve Developer or Client to be done by the project managers.	Create GUI to capture project information. Create classes to retrieve project information, existing developers, and existing clients. Create classes to update developers or clients project information.	2
	P009	Add Client Feature Request to be done by the clients.	Create GUI to capture feature information. Create a table in a database to store client features. Create classes to write feature information to the database.	4
	P010	View or Edit Feature should be possible for all users.	Create GUI to capture feature information. Create classes to retrieve previously added features. Create classes to update features if	5

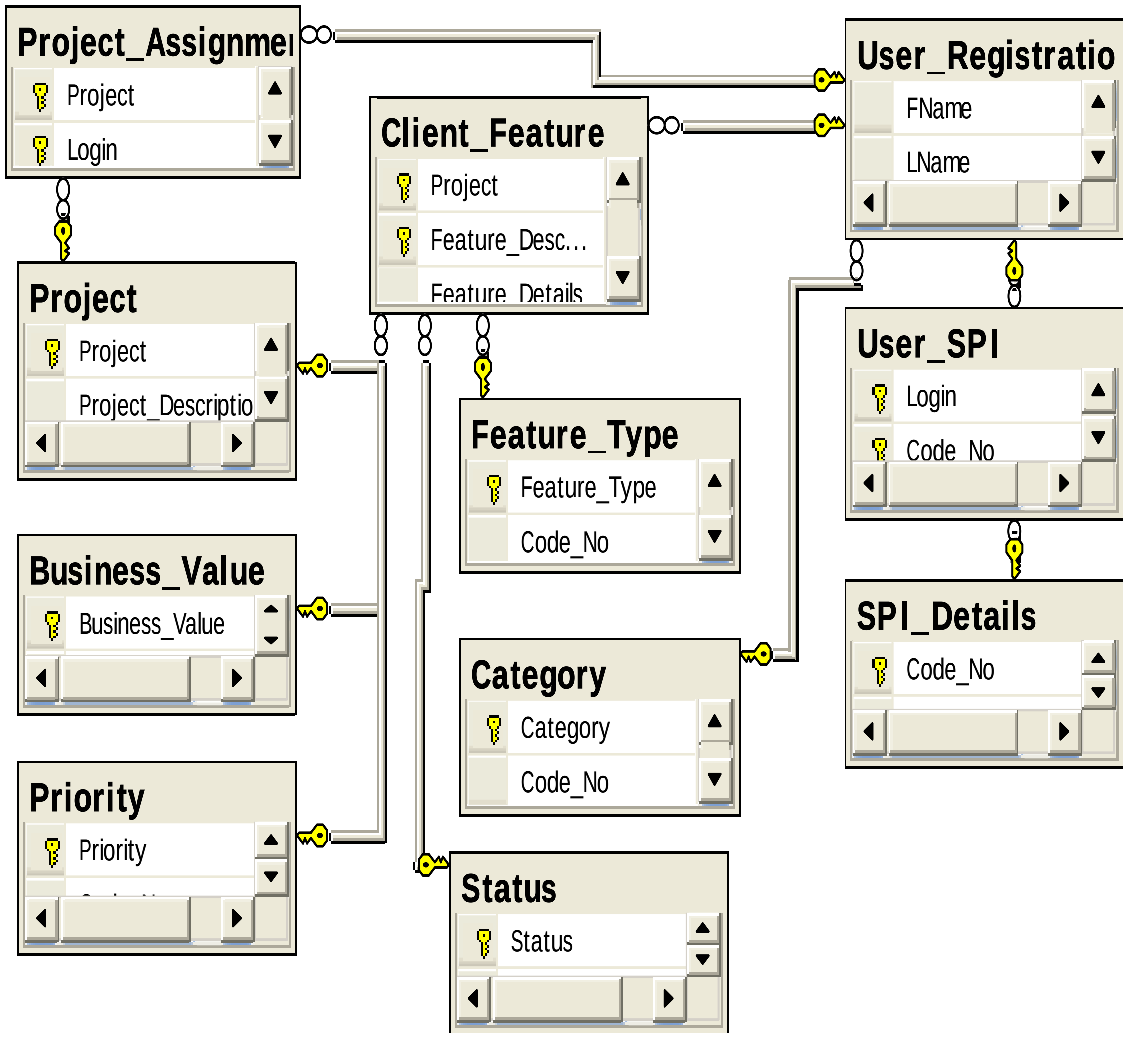
User Activity	ID	Subject Area (Feature(s))	Task (s) to enable activity.	Estimated Time (day (s))
			they are edited. **See requirements for specific levels of access.**	
	P011	Viewing of Client Contact Information is permitted by developers.	Create GUI to capture client personal details. Create classes to retrieve client personal details. Include security feature to restrict developers from viewing clients' information that are not a part of their project(s). **See requirements for specific levels of access.**	2
	P012	Main Interface should be presented to users.	Create the main GUI with links to the various system options.	4
Personnel Capability Management	P013	Allow Completion of SPI Survey by users.	Create GUI to ask survey questions and capture user responses to survey questions. Implement survey logic in classes to return assessment summary. Create tables to store user assessment summary. Create classes to write summary to database.	7
	P014	View Survey Summary functionality is presented for developers.	Create GUI to capture survey summary. Create classes to retrieve survey summary. **See requirements for specific levels of access.**	2
Help and Documentation	P015	Provide Help for users.	Create help GUI for all user features. Create installation manual for the system.	10

* Number of Developer(s) --- 1
* Number of System Features --- 15

* Estimated time for development --- 54 days --- A work day is expected to be 8 hours --- total hours 432

* Restriction --- Security feature should prevent clients and developers from accessing projects which are not assigned to them. Security feature should prevent clients from editing features that are not Requested, Estimated, or Scheduled. Security feature should prevent users from deleting records.

Figure 10.11. Database Diagram



Glossary

- Client - In this context a client is seen as the customer representative who is actively participating and involved in requirement specification, software testing, and other development practices often recommended by agile models.
- Developer – A software builder who occupies one or more specific roles in the software team.
- Project Manager - A software development team leader.
- Non-registered User - A user (client, developer, or project manager) whose draft registration details have not been entered on the system by the project manager.
- Draft Registration – Temporary personal information entered for a specific user by the project manager.
- Registered User – A user whose draft registration is completed.
- Personal/Contact Details – User personal information which includes: name, address, telephone number, email address, login. For registration purposes, the password becomes part of user personal details.
- Belbin’s SPI Survey - A simple inventory used for assessing personality traits (see Belbin (2002) for further details).
- Client Feature – A client-valued system function.
- Project – An agreement between a client and the development team to deliver a software solution.
- Add/Approve User – The project managers’ user approval process; includes completing a draft user registration, and adding a user to projects after the Belbin SPI survey is completed by the user.

Table 10.2. ASRMT Development Summary

ID	Implemented	Modified	New Activities	Estimated Time (days)	Actual Time (days)	Rework Time (days)	Rework Activities
P001	Yes	No	-	4	3	1	GE
P002	Yes	No	-	3	3	-	-
P003	Yes	No	-	4	3	-	-
P004	Yes	No	-	1	1	-	-
P005	No	-	-	2	-	-	-
P006	Yes	No	-	2	2	1	GE
P007	Yes	No	-	2	2	-	-
P008	Yes	No	-	2	2	-	-
P009	Yes	No	-	4	4	1	GE
P010	Yes	No	-	5	6	1	GE
P011	Yes	No	-	2	3	-	-
P012	Yes	No	-	4	5	1	GE
P013	Yes	No	-	7	6	3	GE
P014	Yes	No	-	2	3	1	GE
P015	No	-	-	10	-	-	-
Key: GE – GUI Enhancement							

Additional Development Activities (beyond initial estimates)

* 14 days were utilised for ASRMT (development) research activities

* 6 days were utilised for planning and implementing the ASRMT database

* 7 days were utilised for designing and implementing ASRMT reports (summaries)

***** Total time taken for development --- 79 days**

Appendix C. Agile Social-Risk Mitigation Tool (ASRMT) User Evaluation

This evaluation is for the test version of ASRMT, a personality assessment and feature management tool that has been designed and built to primarily assist software project managers in their handling of some of the social aspects of agile software development. This scenario-based evaluation is informal and is aimed at verifying whether ASRMT demonstrates proof of the concepts proposed in the research project (Title: Tool Support for Social Risk Mitigation in Agile Projects). As someone with expert knowledge regarding software development, your feedback and suggestions will be extremely valuable for the current study.

ASRMT Purpose

Among the concepts explored in this study, managing group dynamics, personality differences, and team cohesion have been identified as major factors contributing to software development projects' success. However, research examining the means of supporting the management of group dynamics, personality differences and team cohesion is scarce. As a consequence, this research project has proposed a way to address this gap by offering a toolset called ASRMT which allows personality assessment and management using Belbin's Self-perception Inventory (SPI), and client feature management, extending customer involvement through an interface.

ASRMT is by no means meant to replace current project management tools, but is intended to complement them, offering developers a way to extend interaction with customers by allowing them to enter and manage requirements in the form of feature requests. In addition, ASRMT offers software team members a way to verify their main personality profile in order for project leaders to properly assemble teams with the right mix of expertise and to manage according to team members' capabilities.

ASRMT User Evaluation

The evaluation has two (2) parts. In the first part (part I) you will undertake a few tasks in order to test the tool's functionality, in the roles of Project Manager, Developer, and Customer. The intention of part II is to solicit feedback regarding your experience whilst conducting these tasks.

Part I: Tool Functionality - Personality assessment and client feature management

Project Manager

Users belonging to the 'Manager' category have administrator privilege. Thus, in this role you are allowed access to all of ASRMT functionalities. The following tasks are aimed at verifying ASRMT project management system functionalities:

1. You are a project manager that has just started work with an agile development company. Log in to ASRMT as the administrator (login: asrmtadmin, password: asrmtadmin) and sign up as a new user with 'Manager' privilege using the New User menu.
2. Every new user (including project managers) needs to provide information on their personality, so that teams can be composed of compatible people. Logout as asrmtadmin, and re-login with your new user details (from 1 above). Complete the Self-perception Inventory (SPI) survey using the Complete SPI Survey menu.
3. Add a new project to the system using the Add Project menu (Project Name – AUT Security System, Project Description - Security of Student's Novell Network, and Start Date – 10/03/2007). Note: other project data exists in ASRMT.
4. Other users exist in the ASRMT database; assign a customer (garybeaton) and a developer (christlill) to the project just created (AUT Security System) by using the Assign User-Project (s) menu. The project is now available to accept feature requests from the customer.

5. Next you need to change an ongoing project – the Credit Card System. Edit a feature previously added using the Edit Feature menu. The status of the feature determines what data is added to the feature; for example, if the values for ‘Estimated Hours’ and ‘Date Estimated’ is entered, the feature status automatically changes to **Estimated**. Let the feature traverse state by updating its associated information (enter the Date Scheduled, notice the status changes to **Scheduled**). (Features entered by clients have ‘Requested’ status by default, features then traverse status in the following order: Estimated, Scheduled, In Progress, and Completed).
6. Take some time to check out the other different menu options in the Administration section of the ASRMT main menu (Change Password, View/Edit Personal Details, View Contact Details, and Edit Project).
7. As a new employee you decide it would be useful to get an overview of the current projects. Go to the Project Summaries Section of the main screen and view summaries for the projects. To get a feature summary, select the Feature Summary menu - information should exist pertaining to the features previously entered (Select the Credit Card System, Banking System or AUT Website Project, you want features requested by the client between 01/10/2006 (Start Date) and 01/05/2007 (End Date), Filter By (in any order) – All, Type, Priority, Risk Rating, or Status). To get a project summary, select the Project Summary menu. This should enable you to retrieve information pertaining to the projects previously entered that are started with features (Filter By (in any order) – All Projects, Selected Project, Completed Projects or Outstanding Projects, Start Date (01/01/2007), End Date (01/05/2007)). You can also get a summary of the SPI survey data by selecting the SPI Survey Summary menu (Retrieve User List (in any order) – All, By Category, By Role, or By User).
8. You can continue to use ASRMT to exhaust the different features as desired.
9. Logout using the **Logout** link at the bottom-left of the screen.

Developer

Users belonging to the ‘Developer’ category are assigned certain restricted privileges. Thus, in this role you should only be allowed access to specific ASRMT functionalities (for instance, developers can only access projects that they have been added to). The following tasks are aimed at verifying ASRMT developers’ system functionalities:

1. Your name is Christoph Lill, an agile software developer. In the last few months you have been working on the Credit Card System and AUT Website projects. Log in to ASRMT (login: christlill, password: christlill).
2. View your SPI survey data using the SPI Survey Summary menu. (You should only be able to see your data, not that for anyone else.)
3. Take some time to check out the other different menu options in the Administration section of ASRMT main menu (Change Password, View/Edit Personal Details, and View Contact Details).
4. You have been on holiday for the last week and need to catch up with the current status of your projects. Go to the Project Summaries Section of the main screen and view the summaries for them. (Note: you are only allowed to see project summaries for projects that you have been added to by the Project Managers.) Get a feature summary for the Credit Card System project – information should exist pertaining to the features previously entered (Insert Start Date (10/09/2006), End Date (05/03/2007), Filter by Priority, then by Status). Then retrieve a project summary filtered by Outstanding Projects, Start Date (05/01/2007), and End Date (10/04/2007).
5. You can continue to use the ASRMT to exhaust the different features as desired.
6. Logout using the **Logout** link at the bottom-left of the screen.

Customer

Users belonging to the ‘Customer’ category are assigned certain restricted privileges. Thus, in this role you should only be allowed access to specific ASRMT functionalities (for example, customers can only access projects that they have been added to). The following tasks are aimed at verifying ASRMT customers’ system functionalities:

1. Your name is Gary Beaton, a customer who has contracted the software development company to produce a Credit Card System for you. Log in to ASRMT (login: garybeaton, password: garybeaton).
2. It has been some time since you used the tool and you cannot recall completing a Self-perception Inventory survey. Try to complete the SPI survey using the Complete SPI Survey menu.
3. Add a few client feature requests to the Credit Card System project using the Add Feature menu (only customers can add features).
4. Edit a previously added feature using the Edit Feature menu (for example, Feature – Create automated email); please note the feature Status, can you change it? Now come out of the Edit Feature Menu and change your password using the Change Password menu under the Administration section of the tool). Return to the Edit Feature menu, try to edit the same feature – note the feature Status again.
5. Take some time to check out the other different menu options in the Administration section of ASRMT (for example, View/Edit Personal Details).
6. You want to know how the team is getting on overall with the project. Go to the Project Summaries Section of the main screen and get a summary. (Note: you are only allowed to see project summaries for projects that you have been added to by the Project Managers.) Get a feature summary for the Credit Card System project – information should exist pertaining to features previously entered (Insert Start Date (22/02/2007), End Date (15/04/2007), Filter by Priority, then by Status, then by Risk Rating). Note the restrictions.
7. You can continue to use the ASRMT to exhaust the different features as desired.
8. Logout using the **Logout** link at the bottom-left of the screen.

You have finished testing out the functionality of the system – thank you. Please move on to the next page so that you can give us your impressions of the tool and your use of it.

Part II: User Experience Feedback

Please read the instructions and supply an appropriate answer for all questions. This will either be a ticked box (normally indicating the extent to which you agree or disagree with a statement) or some free text. If you make a mistake please put a large cross through your original answer and then select your new answer.

ASRMT Stability and Learning

1. You found it easy to use the features of ASRMT.

Strongly agree ☐ Agree ☐ Disagree ☐ Strongly disagree ☐

2. Did you encounter any bugs whilst using ASRMT?

Yes ☐ No ☐

If yes, approximately how many distinct bugs did you encounter?

1-2 ☐ 3-5 ☐ 6-10 ☐ More than 10 ☐

3. You were able to successfully complete the tasks in part I.

Strongly agree ☐ Agree ☐ Disagree ☐ Strongly disagree ☐

4. It was easy for you to learn to use ASRMT.

Strongly agree ☐ Agree ☐ Disagree ☐ Strongly disagree ☐

5. When there was an error using ASRMT, you recovered easily and quickly.

Strongly agree ☐ Agree ☐ Disagree ☐ Strongly disagree ☐

6. You found using ASRMT frustrating to use.

Strongly agree ☐ Agree ☐ Disagree ☐ Strongly disagree ☐

7. ASRMT is intuitive and simple to use.

Strongly agree ☐ Agree ☐ Disagree ☐ Strongly disagree ☐

ASRMT Usefulness and Research Objectives

8. ASRMT would be useful if used in live projects.

Strongly agree ☐ Agree ☐ Disagree ☐ Strongly disagree ☐

9. ASRMT offers functionality to address the features discussed in the 'ASRMT purpose' section at the beginning of this document.

Strongly agree ☐ Agree ☐ Disagree ☐ Strongly disagree ☐

10. In terms of your overall impression of ASRMT…

List any negative aspect (s):

1 …………………………………………………………………………..

2 …………………………………………………………………………..

3 …………………………………………………………………………..

List any positive aspect (s):

1 …………………………………………………………………………..

2 …………………………………………………………………………..

3 …………………………………………………………………………..

11. Please outline any suggestions you have for improving ASRMT.

1 …………………………………………………………………………..

2 …………………………………………………………………………..

3 …………………………………………………………………………..

Thank you for your participation!