The University Of Sheffield.

Agent-based Modelling of Transactive Memory Systems and Knowledge Processes in Agile versus Traditional Software Development Teams

A thesis submitted for the candidature of the degree of Doctor of Philosophy

Department of Computer Science

Andrea Corbett

July 2012

Supervisors: Professor Mike Holcombe, Professor Stephen Wood

Abstract

The objective of this research is to develop an agent-based model of transactive memory systems (TMS - meta-knowledge of expertise and knowledge in a team) simulating software development teams using two different software development methodologies, Waterfall (a structured methodology with a series of large discrete phases) and the Agile, eXtreme programming (XP - more recent, dynamic, and tuned to change and flexibility). There does exist research relating to TMS; comparisons of software development methodologies; cognitive processes of software development teams; and also agent-based modelling of social and cognitive systems. This is interdisciplinary research spanning psychology and computer science aiming to consolidate these discrete streams of research.

The model evaluated the parameters of small/large tasks, and working solo/in pairs to investigate the effect on TMS, knowledge and team output. Over three simulations, increasing in cognitive realism, the model introduced greater complexity and novelty to the agents' work, and various initial conditions of team knowledge and team member familiarity.

The results illustrated a number of differences in TMS, knowledge processes and output between XP and Waterfall teams. The main findings indicate that as the novelty and complexity of the task increases the use of some XP techniques can lower the reduction in output. Also the dependence on TMS accuracy for teams using some XP techniques in complex novel environments is high while the team knowledge distribution becomes much more homogenous. This contradicts the literature that asserts a positive relationship between TMS accuracy and knowledge heterogeneity. Results also suggest that XP techniques can compensate for the advantages relating to team members' prior knowledge of each other allowing newly formed XP teams to perform better.

The results contribute to understanding how knowledge and memory processes in software development teams affect team output, and how the adoption of XP practices can produce results that challenge the established TMS literature.

Dedication

To Taylor, Edward and Dominic.

My three fabulous sons, I am very proud of you.

Never stop learning,

never stop questioning,

never stop solving problems.

Acknowledgements

I am privileged to have worked with two extremely distinguished and accomplished supervisors for the duration of this research. Both have been very generous in providing copious amounts of invaluable advice for which I am very grateful.

I wish to thank my supervisor Professor Mike Holcombe, from the Department of Computer Science for his endless advice, guidance and patience, without which I would have been lost. He has been a constant mentor for my work. I would also like to thank my other supervisor, Professor Stephen Wood, from the Institute of Work Psychology, who has also given much time, advice and guidance. Although he moved to the University of Leicester he has continued to provide me with advice, alternative views and critical feedback.

The ESRC and the EPSRC, who funded this research, deserve thanks, and also the University of Sheffield. I would also like to thank the Verification and Testing group in the Department of Computer Science for taking a mature psychology graduate under their wing and patiently helping her to grasp the more technical aspects of this research. Special thanks go to Mesude Bicak and Mariam Kiran who, as well as providing me with invaluable assistance with FLAME, have been great friends and always ready to talk over a coffee despite busy schedules.

I would also like to thank my slightly incredulous friends, Deborah Cowsill, Steve Banner, Helen Newman, Helen Walshaw, and Mish Wallis for their constant encouragement and belief in my success. My MMU sisters Paula Duxbury, Melanie McKillop and Giulia Cowap deserve a special thanks for their encouragement and support; I am proud of them too for continuing with lifelong learning and going on to achieve great things themselves.

Finally, my greatest thanks go to my long suffering family, my husband Gary, my boys, Taylor, Edward and Dominic, and Lauren, Gary's daughter. They have all become much better cooks and tolerantly shared my many highs and lows. Extra special thanks go to my boys who hopefully will have absorbed the value of education and take that forward in their lives.

Declaration

I declare that this thesis was composed entirely by myself and that the work contained herein is my own except where explicitly stated otherwise. This work has not been submitted for any other degree or professional qualification other than as specified.

Andrea Corbett

Contents

List of figures

List of tables

Publications

Corbett, A., Holcombe, M. & Wood, S., 2008. Computational Modelling of the Development of Human Transactive Memory Systems, *3rd Annual South-East European Doctoral Student Conference*. Thessaloniki, Greece 26-27 June 2008. South East European Research Centre.

Thomson, C., Corbett, A., Holcombe, M., Managing Inquiry Based Learning: Learning from Experience, *In proceedings of Learning Through Enquiry Alliance (LTEA) Conference*, University of Sheffield, UK, June 25-27, (2008). Also distributed at Exploring Inquiry Based Learning, HEA Workshop, CILASS, University of Sheffield, Sheffield, April 24, (2009)

Stannett, M.P., Thomson, C., Cowling, A.J.C., Corbett, A., Holcombe, W.M.L., The Crossover Project, *In proceedings of Learning Through Enquiry Alliance (LTEA) Conference*, University of Sheffield, UK, June 25-27, (2008).

Corbett, A., Holcombe, M., Wood, S., Validation of an Agent-Based Model for Simulating Transactive Memory Systems and Knowledge Processes in Teams. *Cognitive Systems Research*. Under revision for resubmission.

Chapter 1: Introduction

1.1 Overview

This thesis brings together three discrete themes by developing an agent-based model to investigate the cognitive behaviour, specifically of transactive memory systems, of software development teams utilising two different software development methodologies. It is interdisciplinary research spanning both psychology and computer science, taking inspiration from research on team cognition, transactive memory systems, software development methodologies, software development teams, and agent-based modelling.

Psychologists have studied individual cognition for many years but it is increasingly becoming accepted that an individual's cognition is a product of their social environment and interactions with others; therefore team cognition is of interest in order to optimise the benefit of effective teams to modern organisations. Team cognition is considered to be cognition at the team level over and above individuals' cognition (Klimoski and Mohammed, 1994, Theiner et al., 2010). Transactive memory, a particular facet of team cognition, is an individual's beliefs relating to "who knows what" in their team, and a transactive memory system is the transactive memory, communication, and coordination at team level (Wegner, 1995). A transactive memory system allows members to gain access to a knowledge base larger and more complex than any one person alone could possess providing many benefits to a team. The transactive memory system helps team members compensate for one another, reduces cognitive load on individuals, decreases redundancy of effort, facilitates knowledge sharing and assists with the efficient allocation of resources in a team.

A transactive memory system is particularly useful in environments where tasks can be complex and highly innovative. The work carried out by software engineering teams requires very specialist skills, is often creative, complex and unstructured, and requires teams with high levels of coordination and shared cognition. The study of cognitive processes and specifically transactive memory systems can contribute to understanding many aspects of software development team working.

Since the development of computer systems became larger and more complex than one person could achieve in a short time, different methodologies have been utilised to formalise the process, ensure quality, and manage the production of a commercial product. The first such methodology was defined by Winston Royce in 1970 (Royce, 1970) and became known as the Waterfall method. This is a formal method using a number of discrete phases, formal processes and copious documentation. In contrast a more recent, less formal, Agile (Beck et al., 2001) methodology known as eXtreme programming (Holcombe, 2008, Sommerville, 2004) has been developed which is growing in popularity. There are a number of studies comparing the two approaches however very little work has been done looking at the team cognition aspects of comparison.

As social and cognitive processes cannot be observed directly researchers have had to develop new methods to allow them to study and understand team cognition. One of these is agent-based modelling which has been used many times as an effective alternative tool for investigating social cognition. It has been used for applications such as smoking behaviour in teenagers, cooperative behaviour, group conflict, crowd behaviour and bystander behaviour (Goldstone and Janssen, 2005, Smith and Conrey, 2007)

For systems or phenomena that are too complex or inaccessible often a simple theoretical model allows a scientist to understand how it works. Traditional models predict and describe a system but increasingly computational models have been used that can simulate behaviour in a system allowing it to be observed and measured over time. The use of agent-based modelling allows experiments to be designed and the results interpreted through examining the way the model behaves.

All agent-based models work by synthetically constructing virtual versions of social phenomena from low-level descriptions of individual agents (Goldstone and Janssen, 2005). They are typically defined as having distributed resources, expertise and intelligence, no explicit global control, and operate by emphasising social agency. They are comprised of agents, which are small autonomous software programs that have internal memory and decision-making functions. The agents communicate with other agents and as they do so they exhibit behaviour that follows the rules encoded within their functions. As the individual agents operate, the system as a whole may exhibit emergent behaviour.

So an agent-based model is a computational method that provides a simplified representation of social reality and allows researchers to create, analyse and experiment with models to answer questions posed about the social reality being modelled.

As stated there is a body of research relating to transactive memory systems, there is also research comparing different software development methodologies; there is research into the cognitive processes of software development teams and also agent-based modelling of social systems. This research aims to consolidate these discrete streams of research and take an agent-based modelling approach to investigate the behaviour of knowledge and transactive memory systems for both traditional Waterfall and eXtreme programming conditions in software development teams. The model compares knowledge, TMS and performance for both methodologies in situations relating to routine versus novel tasks and combinations of knowledge distribution.

1.2 Approach and research objectives

The primary approach and objectives of this research are:

To develop agent-based models to simulate Transactive Memory Systems in software development teams using two different software development methodologies.

To contribute to the understanding of how knowledge distribution and transactive memory systems affect the behaviour of software development teams under different conditions.

To identify conditions that may improve the potential output of software development teams.

A secondary research objective is to utilise the FLAME framework in an unconventional manner in order to test alternative uses of the framework.

1.3 Structure of the thesis

The thesis is structured as follows:

Chapter 2. Software development methodologies: describes the two different software development methodologies, the traditional Waterfall method and the Agile methodology, XP.

Chapter 3. Transactive memory systems: outlines the key concepts relating to the cognitive processes in teams, specifically, transactive memory systems; knowledge; and TMS in software development teams.

Chapter 4. Agent-based modelling: describes agent-based modelling and discusses how it has been used for modelling social and cognitive systems.

Chapter 5. Modelling environment: describes X-machines and the FLAME modelling environment.

Chapter 6. The model: describes the model design and construction in detail.

Chapter 7. Model validation: describes the initial validation of the model.

Chapter 8. Aligning the model: describes an exercise to align the model with another model.

Chapter 9. Model Tabula Rasa: describes the initial experiment to investigate transactive memory systems and knowledge when comparing software approaches in a simple, routine environment.

Chapter 10. Model Novelty: describes the next experiment introducing an environment of novelty and non-routine tasks to the previous model, more closely modelling innovative software development projects.

Chapter 11. Model Differentiation: describes the experiment introducing further cognitive realism by varying conditions of initial knowledge and memory to the previous model.

Chapter 12. Discussion: provides a summary of the results and findings of this research, and discusses limitations and further work.

1.4 Contribution

As this is interdisciplinary research spanning both psychology and computer science this thesis aims to contribute in a variety of ways. FLAME, the agent based modelling environment of choice for this research, has been used for many applications but not for modelling cognitive processes. Most commonly FLAME is used to model large numbers of agents spatially in a three dimensional environment. This research aims to use FLAME in a novel way by modelling small numbers of agents where the information of interest is in their memory and not in their position in space. Secondly, there is a body of research into transactive memory systems, however, there are fewer studies focussing on software development teams, therefore, this research also contributes to understanding the mechanisms of transactive memory systems and knowledge processes in software development teams. Finally, this research contributes to the XP development literature and to the understanding of factors that optimise the use of Agile techniques and to identify optimum conditions for best team output, for Agile software development methodologies.

The model developed has illustrated a number of differences in TMS, knowledge processes and output between teams using different methodologies. The main findings indicate that the use of the XP technique of pair programming can help to insulate a software development team from the negative effects on output of additional complexity and novelty of tasks.

Also the dependence on TMS accuracy for XP teams in more complex novel environments is high while the team knowledge distribution becomes much more homogenous. This contradicts the literature that asserts a positive relationship between TMS accuracy and knowledge heterogeneity. Results also suggest that XP techniques can compensate for the advantages relating to team members' prior knowledge of each other potentially allowing newly formed XP teams to perform better.

The results contribute to our understanding of how knowledge and memory processes in software development teams affect team output, and how the adoption of some XP practices can produce results that challenge the established TMS literature. Previous agent-based models of TMS have been theoretical and not set within a work context. This research describes an agent-based model used in a contextual manner within the software development industry, and simulating specific software development practices.

Chapter 2: Software development methodologies

2.1 Introduction

This chapter describes the two software development methodologies used in the comparison by the agent-based model developed for this research. It proceeds to look at the advantages and disadvantages of each and outlines some of the literature comparing the two methodologies. This will give some basic understanding of each methodology and outline the issues relating to some of the factors that influence their comparison.

Since the development of computer systems became larger and more complex than one person could achieve in a short time, different methodologies have been utilised to formalise the process, ensure quality, and manage the production of a commercial product. The first such methodology was defined by Winston Royce in 1970 (Royce, 1970) and became known as the Waterfall method. This traditional method was, and is, used extensively with numerous derivatives having been developed since its inception, such as the V method and the Spiral method, amongst others.

Over time some members of the software development industry became frustrated with what they perceived to be the rigid methodologies that were being used and felt that there was a better way. In 2001 a group of 17 software developers met in Utah, US and developed and subsequently published the Manifesto for Agile Software Development (Beck, 1999, Beck et al., 2001). Agility in software development is not a methodology; it is an approach. A number of different Agile methodologies have been developed that have embraced the Agile approach, such as eXtreme Programming (XP), Dynamic Systems Development Method, Feature-Driven Design, Crystal, Agile Modelling and SCRUM.

XP (Holcombe, 2008, Sommerville, 2004), originally used in 1996, was developed and went on to become a widely used Agile methodology. It is based around five values and 12 activities which operate as a lightweight, low-risk, flexible, predictable, scientific and enjoyable way to develop software.

2.2 The Waterfall Method

The Waterfall method (Balci et al., 2010, Sommerville, 2004) originated in the manufacturing and construction industry and the techniques were adopted for the burgeoning software development industry in the 1970s. This method is often used in large and complex software development projects with large project teams requiring high levels of organisation, coordination and cooperation.

2.2.1 Description

It follows a highly structured pattern, where development proceeds sequentially through a series of discrete phases with each phase necessarily complete and signed off before the next phase can commence. To aid this, each process in a Waterfall project is extensively documented in order that information on every aspect of the project is recorded and available.

Ideally there are no feedback loops to revisit previous phases but in practice it does take place if new information is discovered or problems uncovered. Changes to the project after the phase in the Waterfall method life cycle has passed often prove to be very costly.

2.2.1.1 Requirements analysis

This phase is particularly important, as it is the cornerstone of the system that is about to be built. All possible requirements of the proposed system are captured in this phase by working with the customer to establish any requirements, and defining in clear terms the problem that the customer has that the system is expected to solve. It is essential that analysis be done to understand the customer's business and the wider context in which that customer operates. This phase must also define all functions that the system must perform and any performance requirements. Also, links with external systems, with which the system must be compatible, need to be specified. Finally, a Requirement Specification document is created, which is the input document for the next phase of the method.

2.2.1.2 Design

This step takes the requirements analysis from the previous phase and defines the hardware and software architecture, performance and security parameters, data storage, programming language and environment. This phase also establishes strategies to deal with issues such as exception handling, resource management and interface connectivity. This is also the stage at which user interface design is addressed, including issues relating to navigation and accessibility. The output of this stage is a design specification document, which is used as input to the next phase of the method.

2.2.1.3 Implementation

This phase consists of the construction of the system as defined in the design specification developed in the previous phase. This part of the project is carried out by a team of programmers, interface designers and other specialists, using tools such as compilers, debuggers, interpreters and media editors. The output of this phase is the system, built according to a pre-defined coding standard and debugged, tested and integrated to satisfy the design requirements. Version control is often used for large projects to track changes and revert to previous versions in case of problems.

2.2.1.4 Testing

In this stage, both individual components and the integrated whole are methodically verified to ensure that they are error-free and fully meet the requirements outlined in the first phase. Three types of testing typically take place: unit testing of individual code modules; system testing of the integrated product; and acceptance testing, formally conducted by or on behalf of the customer. Defects are recorded and feedback provided to the implementation team to enable correction. This is also the stage at which product documentation, such as a user manual, is prepared, reviewed and published.

2.2.1.5 Installation

This phase occurs once the product has been tested and signed-off. It comprises preparing the system for installation at the customer site. The system is installed and is usually allocated a formal revision number to facilitate updates at a later date.

2.2.1.6 Maintenance

This is the final phase of the method and is typically a very long, if not never-ending one. In the event of change requests initiated by the customer, or defects uncovered during live use, modifications are made to the system. These changes may be minor with little effort required for a new interim release version, or in the case of major changes a new Waterfall project may be initiated culminating in a major system release version. Any new release will have an updated maintenance release number.

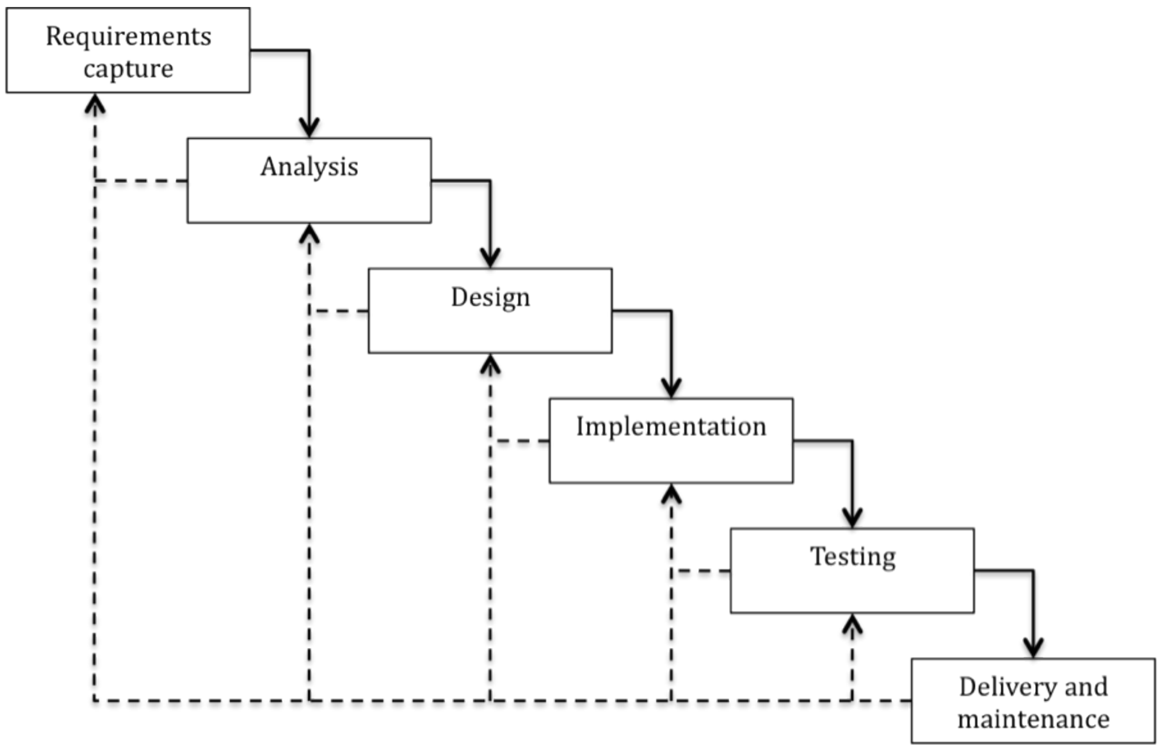


Figure 1: The Waterfall Method

2.2.2 Advantages

The structured approach of the Waterfall method is appealing to many in the software development industry. Each phase is explicitly defined and the project cannot move to the next one until it is completed, with progress identified through the use of milestones by both vendor and client. It is considered a simple and easy to understand approach that is disciplined and structured. For this reason it is often the first example of software development models used in Computer Science and Software Engineering courses and reference material.

The emphasis on getting the requirements and design correct before writing a single line of code ensures minimal wastage of time and effort in future phases. It has been estimated that a requirements defect that is not discovered until construction or maintenance will cost between 50 and 200 times more to fix than if discovered during the requirements phase (McConnell, 1996).

The emphasis on detailed documentation is seen as a strength to exponents of the Waterfall method. This takes the form of both detailed documentation in early phases of the model and the code itself in later phases. This insures the project against loss of knowledge should team members leave or be moved to other projects. New team members can simply read the documentation to learn about the system so far (Balci et al., 2010, Sommerville, 2004).

2.2.3 Disadvantages

The greatest disadvantage of the Waterfall method is that until the final stage of the development cycle is complete, a working model of the software does not lie in the hands of the client. Typically, Waterfall projects take a long time to deliver a working system. There are a number of anecdotal stories of software development projects using the model, which after initial requirements capture, over very long software development cycles, potentially years, have delivered a system that is no longer required by the client.

Moreover, customers often don't really know what they want at the start of the project and what they actually want emerges out of repeated two-way interactions over the course of the project. In this situation, the Waterfall method, with its emphasis on up-front requirements capture and design can be seen by customers as unable to cope with uncertainty and change within the real world.

Further, given the changing nature of customer needs, estimating time and costs with any degree of accuracy is often extremely difficult. Therefore, the method is recommended for use in projects that are relatively stable, and where customer needs can be clearly identified at an early stage.

Another criticism revolves around the method's implicit assumption that designs can be feasibly translated into real products, which sometimes causes problems when developers actually begin implementation. Often, designs that look feasible on paper turn out to be expensive or difficult in practice, requiring a re-design and hence destroying the clear distinctions between phases of the traditional Waterfall method (Balci et al., 2010, Sommerville, 2004).

2.3 Agile philosophy

2.3.1 The Agile Manifesto

The Agile Manifesto (Beck et al., 2001) is an approach to software development, not a methodology. It consists of a high level set of principles that provide guidance for the development of Agile methodologies such as eXtreme programming.

> *"In order to succeed in the new economy, to move aggressively into the era of e-business, e-commerce, and the web, companies have to rid themselves of their Dilbert manifestations of make-work and arcane policies. This freedom from the inanities of corporate life attracts proponents of Agile Methodologies, and scares the begeebers out of traditionalists. Quite frankly, the Agile approaches scare corporate bureaucrats, at least those that are happy pushing process for process, sake versus trying to do the best*

for the "customer" and deliver something timely and tangible and "as promised", because they run out of places to hide." (Beck et al., 2001)

2.3.1.1 The Agile principles

The Agile principles are the fundamental building blocks of the Agile philosophy. They are (Beck et al., 2001):

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity, the art of maximising the amount of work not done, is essential.

11. The best architectures, requirements, and designs emerge from self-organising teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

"At the core, I believe Agile Methodologists are really about mushy stuff, about delivering good products to customers by operating in an environment that does more than talk about "people as our most important asset" but actually "acts" as if people were the most important, and lose the word "asset". So in the final analysis, the meteoric rise of interest in, and sometimes tremendous criticism of, Agile Methodologies is about the mushy stuff of values and culture." (Beck et al., 2001)

2.4 EXtreme Programming (XP)

2.4.1 Overview

The previous Agile principles, as an approach, are used as the fundamental framework for eXtreme programming (Jeffries, 2011, Wells, 2009). One of the central pillars on which XP is built is recognising that people are central to the work, that they are all individuals, and have many different needs and behaviours. Another central ethos to XP is the idea of small steps that create a dynamic continuous flow of added value throughout the project. These fundamental principles underlie the disciplines and mechanisms of XP, namely the five values and the 12 basic practices (Beck, 1999, Holcombe, 2008).

2.4.2 The five values

The above fundamental principles lead on to the five values that form the basis of XP; they are good communication, simplicity, feedback, courage and respect (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.2.1 Communication

It is accepted that communication is important in any team venture and XP projects are no different. It is explicitly addressed as it is seen as being of paramount importance to implement all other aspects of XP. XP recognises a number of stakeholders in a project and effective communication between all stakeholders is enabled using a wide collection of procedures and activities. Stakeholders include:

- Customers – managers, financial directors, marketing departments etc.

- Users – admin staff, the general public etc.
- Developers – programmers, managers, financial directors etc.
- Any other organisations that may have an interest such as media or government.

XP makes it clear that communication is important, not only between these organisations but within them. Communication is encouraged not only within the team of developers but also with the customer and within the customer. This is to ensure that all views within the customer organisation are taken into account to produce the best system possible. Of course there are a number of XP practices that contribute to effective communication in all directions (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.2.2 Feedback

This is closely allied to communication and again it is enabled by XP mechanisms designed to create effective feedback to all parties. Traditional Waterfall projects often go long periods before reporting back on the progress of working on the same large piece of work. XP has mechanisms for the client to see continuous results and to allow them to relate progress to their business needs. So regular meetings with the customer to show them small pieces of functionality contributes to the feedback for the developers and allows the project to maintain the right direction for the customer (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.2.3 Simplicity

This refers to the need to maintain simplicity and to ensure that there is no unnecessary functionality built into the system being developed. It is easy to become seduced by greater and greater functionality and clever enhancements; however, the system must be only as complex as absolutely necessary for the business need. This will enhance user uptake and the longevity of the system as well as keeping development and maintenance costs low (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.2.4 Courage

The courage to embrace change and to challenge the historical methodologies for software development is the basis of this value. Change is fundamental to the philosophy of XP and the enthusiasm for change is one of a number of disciplined activities that make up the XP ethos. It also takes courage to eschew years of software development wisdom and to do things differently and confidently. This value also applies to the developers having the courage to throw away code and start again, contradicting more traditional practices of using and reusing code if possible. By rewriting code, developers can often abandon entrenched habits and write more innovative, efficient code (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.2.5 Respect

As people lie at the centre of all XP projects this final but very important value relates to the relationships between all people connected with the team and, to some extent, the wider context such as families and other work colleagues. Problems in software development projects are often related to people issues, not technical issues, and by treating everyone with respect, hearing their point of view, and negotiating the way through problems, these issues can be minimised (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.3 The twelve practices

The above five values relate to the fundamental tenets of the XP methodology whereas the 12 basic practices of XP dictate the day-to-day activities during a project. They are described below (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.3.1 Test first programming

Testing is carried out continuously in an XP project, in fact the tests are written before the code and often the customer contributes to the testing regime. Of course the tests will fail but the discipline of testing is a central feature of XP and for practitioners it becomes second nature. The test sets replace the specification and design thus presenting a rapid feedback mechanism that indicates how much code is correct. This practice relies on good tests being written and this is not always easy to do at the outset. Often tests are written informally from whatever information is available at the time but in a culture of test-first, as more information becomes available, the tests are consistently written before the code (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.3.2 Pair programming

The tenet of this practice is that all code is written by two people working together, using one machine. They take it in turns to use the keyboard while the other navigates and they continuously discuss the work. This ensures a process of continuous review, errors are less frequent and the code is open to constant quality control. In fact it is not only programming, but all activities that should be carried out in this way in an XP project, encouraging pooled expertise, knowledge and skills. Partners are regularly changed which enables information relating to the project to be disseminated quickly and efficiently around the team. Pair programming is also a very efficient method of increasing the knowledge and skills in a team as less experienced programmers can learn from the more experienced by them working together.

Teams using pair programming are always talking, the communication value of XP, and with another value, respect, pair programming has been shown to be a very successful practice (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.3.3 On-site customer

Ideally, but unusually, the customer is on site all of the time although more often representatives from the customer have a very close and frequent working relationship with the XP team. Their role will be to define functionality, set priorities and to guide the direction of the project. Numerous methods are utilised to garner the actual customer requirements, as sometimes they do not know themselves when they are asked. Methods include video, discussion, observation and studies of similar systems in other organisations. As with pair programming, this practice makes use of intense face-to-face communication (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.3.4 The planning game

The customer looks at the requirements and creates a bank of stories relating to discrete parts of the required system. Each story corresponds to a small piece of working software and will typically be delivered within a week. The customer will review the stories, where they sit in relation to the big picture and will decide on the relative priority of each. For each story an estimate of the time, cost and resources required will be made.

The test sets will be developed first and will address not only the functional but also the non-functional requirements such as usability, efficiency or security (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.3.5 System metaphor

This is the organisation of a collection of classes and methods that will achieve the functionality described by the stories in development. To start with this may be quite vague as the problem is addressed with the customer but over time will become more detailed and can be documented more precisely. The system metaphor can be likened to a primitive architecture (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.3.6 Small frequent releases

The philosophy states that the developers should release early and release often. As soon as there is a functional implementation of a story that provides some business benefit then it will be installed in the customer business. This allows the users to start using it and provide feedback. This can take place in small time periods, as short as days or a week. As the stories are developed additional functionality is installed (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.3.7 Always use the simplest solution that adds business value

As mentioned in the values, always use the simplest solution that adds value to the customer's business. The development team should not be tempted to add enhancements because they are clever, or pleasing, they should ask themselves, does the customer really need this feature? (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.3.8 Continuous integration

The units of code are integrated into the system at least a few times every day, if not every few minutes. To be integrated, the units of code must have passed all the unit tests. Functional tests must pass after integration. This is an iterative method of slowly building a working system in very small steps and integrating everything continuously. Adding another trusted XP story (see 2.4.3.4 The Planning Game) to the fully tested system demands that all functional and system tests are carried out again. This is a disciplined process often carried out by the same pair every time. Allowing anyone to do it at any time results in chaos and numerous different versions of the system (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.3.9 Coding standards

This practice defines the rules for shared code ownership and for communication between team members consisting of class and method naming conventions. At the start of the project standards for test sets, story cards, the metaphor and coding styles are defined and everyone is expected to abide by the standards. This has obvious benefits in terms of communication, understanding and quality (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.3.10 Collective ownership of code

Everyone in the team owns all the code and can change any part of it. This will rankle with many experienced developers who feel possessive of their own code. In an XP environment with high levels of communication and cooperation the concept of collective ownership of code works well. It has the additional benefit of shared knowledge of the system, which creates resiliency within the team in the event of staff turnover (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.3.11 Refactoring

Refactoring is the restructuring of code without changing its functionality. The idea is to make it simpler and more understandable as the code is acting as part of the documentation for the system. Refactoring could include:

- Moving methods used in several classes to a separate class
- Extracting superclasses
- Renaming classes, methods and functions
- Simplifying conditional expressions
- Reorganising data.

Refactoring should also take into account the tests and these should also be changed to reflect the refactoring of code (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.3.12 Forty hour week

As people are the most important part of any XP project it is important to maintain happy programmers and this includes not expecting too much of them. Stressed and tired programmers produce poor code, make mistakes and are unhappy. XP is designed to minimise stress by producing results continuously without having to worry if the system will work at the eleventh hour. Proponents of XP claim that it allows a steady pace, increased quality, and improved job satisfaction. Thus it is no longer necessary to work around the clock (Beck, 1999, Cockburn and Highsmith, 2001, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

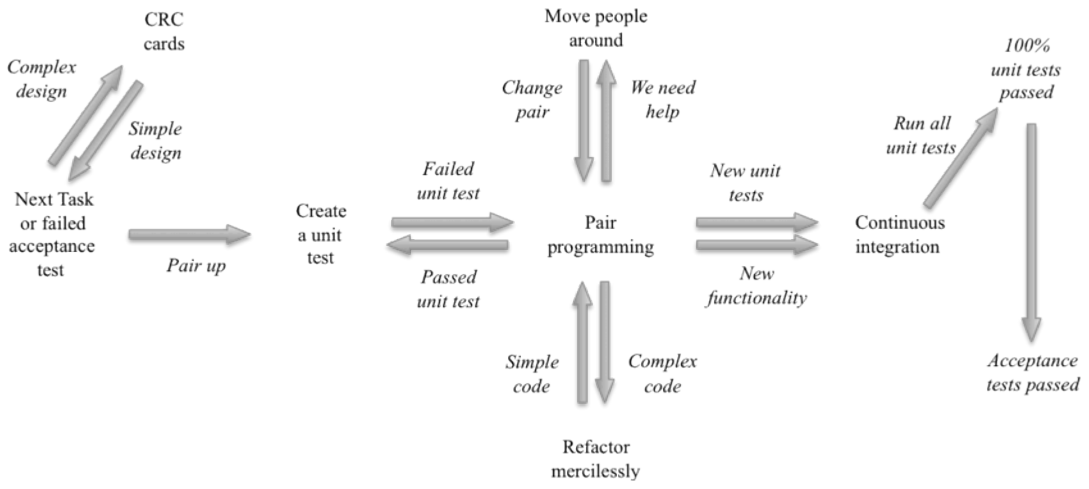


Figure 2: Software development in XP (Wells, 2009)

2.4.4 Advantages

XP is designed as a holistic process and to try to implement individual features such as on site customer or pair programming in isolation often results in failure. One of the strengths of XP is that the values and practices work together to produce an effective development environment.

XP helps achieve a high degree of customer satisfaction because customers notice that XP creates working software faster, and that software tends to have very few defects. XP meets the urgent need for a more responsive approach to software development for real people with real requirements and it allows the customer to change their mind regularly, with minimal cost, with no complaining from the developers (Begel et al., 2008, Macias et al., 2003, Syed-Abdullah et al., 2005, Syed-Abdullah et al., 2006, Williams et al., 2000). Given that XP works on an iterative approach, it is much easier to be able to add features or modify these features based on reviews. XP delivers working software at lower cost, and the software is more likely to do what the end users actually need.

For developers, XP allows the developers to focus on coding and avoid needless paperwork and meetings. It provides a more social atmosphere, more opportunities to learn new skills, and a chance to go home at a sensible time each night. It gives very frequent feelings of achievement, and generally allows the developer to be proud of the code produced. XP projects reduce risk by reducing dependence on individual superstar programmers, and at the same time improving employee satisfaction and retention. XP encourages a high degree of teamwork, getting the entire team including the customer to work together in a room. Since software development is a people-intensive process, the principal measures concern people and for this reason an XP team is less stressed (Cockburn and Highsmith, 2001).

XP projects reduce risk by providing the ability to abandon development at almost any time, and still have valuable functioning code, and probably even a valuable working (albeit incomplete) application. XP emphasises developing the most important features first, rapidly providing the customer with the functionality required and minimising the risk of total project failure. With a focus on testing, each line of code is tested thoroughly, and all code is reviewed thus ensuring a high level of quality.

Finally, the whole development process is visible and accountable. It is very easy to measure performance against the planned schedule (Beck, 1999, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

2.4.5 Disadvantages

With XP being relatively new and fairly contentious there are numerous perceived advantages and disadvantages due to the lengthy debate around the subject. XP proponents often defend criticism of XP by asserting that the very disadvantages that are cited by traditionalists are actually advantages of the method; it is just that the method is misunderstood.

It takes a lot of cultural change to adopt eXtreme programming and often the migration is difficult. It can be difficult to get developers to accept the practices, and it takes discipline to continue doing them all. Also, customers, not used to such a high level of involvement in development can have problems adapting to the practice of being so involved and sometimes find the practice of frequent meetings can incur high expense (Beck, 1999, Begel et al., 2008, Holcombe, 2008, Jeffries, 2011, Wells, 2009).

XP teams can be reluctant to agree to fixed-price, fixed-scope terms, because they know that change will happen. They believe that it is better to deliver what the customer needs at the end of the project than what he thought he wanted before the project began. It is impossible to develop realistic estimates of work effort needed to provide a quote, because at the beginning of the project no one knows the entire scope and requirements. This is used as a criticism that XP projects are a means to take larger amounts of money from customers through the lack of defining a deliverable. This can lead to more difficult contractual negotiations (Watts, 2001).

Agile is feature-driven which means the emphasis is on code rather than design, non-functional quality attributes are hard to be written as user stories. This has led to the criticism that XP incorporates insufficient software design. Although the lack of XP design practices might not be serious for small programs, it can be disastrous when programs are larger than a few thousand lines of code or when the work involves more than a few people.

One criticism is scalability. Some critics claim that XP projects are not suited to large projects. Scalability does not just happen by accident. It must be designed in, whether in a physical machine, executable software, or a work process for coordinating the activities of many people (Bollinger, 2001). XP's focus on doing only what is immediately necessary can cause architectural design problems, especially in complex or larger systems. Failure to take into account long-term goals early on in the lifecycle of a system can lead to project failure.

2.5 Comparison of Waterfall and XP

When comparing the efficacy of XP with that of the Waterfall method there are many factors that may influence the results including: number of lines of code produced; ratio of comments to lines of code; retrospective updating of original specification; number of test cases; hours spent writing code; number of bugs in the code; time taken to produce a working application; and wellbeing of programmers. One view posits that there is little difference in results for both methodologies, and some studies have endorsed this view by suggesting that the amount of code produced and features completed become interchangeable in terms of development cost (Canfora et al., 2005, Ji and Sedano, 2011).

Other research has found benefits to XP by asserting that professional collaborative programmers, who jointly developed algorithms and code, outperformed individual programmers (Macias, 2004, Nosek, 1998, Williams et al., 2000). However this study raises pertinent questions; can two average or less experienced workers collaborate to perform tasks that may be too challenging for each one alone? Can a company bring a product to market substantially earlier by using collaborative programming? Can collaborative programming offer a competitive edge?

Williams et al. (2000) answered one of these questions when they found that pair programming does improve software quality and reduce time to market. They also found that both student and professional programmers consistently find pair programming more enjoyable than working alone (Williams et al., 2000). However, the situation is undoubtedly further complicated by factors such as the expertise of the programmers, the length of time they have been working together, the complexity and the size of the project that is being addressed.

A controlled experiment on pair programming (Arisholm et al., 2007), for complex and simple problems, did not support the hypotheses that pair programming in general reduces the time required to solve the tasks correctly, or increases the proportion of correct solutions. For the more complex system, the pair programmers had an increase in correct solutions but no significant difference in the time taken to solve the tasks correctly. For the simpler system, there was a decrease in time taken but no significant differences in correctness. However, the moderating effect of system complexity depends on the programmer expertise. The benefits of pair programming in terms of correctness on the complex system apply mainly to juniors, whereas the reductions in time to perform the tasks correctly on the simple system apply mainly to intermediates and seniors.

A meta-analysis (Hannay et al., 2009) was carried out on the effects of pair versus solo programming. Results showed that pair programming is not uniformly beneficial or effective. The study found a high level of inter-study variance and recommended that the moderating factors of the effect of pair programming should be investigated further. In addition, the study reported central factors relating to pair programming and in support of Arisholm et al. (2007) they suggested that pair programming is beneficial for achieving correctness on highly complex projects and has time benefits on simpler projects. Of course there is a trade-off, and they found that the higher quality required for complex tasks takes considerably higher effort while reduced time for simpler tasks results in lower quality.

Another review of research with respect to the productivity of agile and traditional teams (Dyba and Dingsoyr, 2008) reported that three of the four comparative studies found that using XP increases productivity in terms of lines of code per hour. The review results also suggest that agile methods can improve job satisfaction, productivity, and customer satisfaction. The strongest, and probably most relevant evidence is from the studies of mature agile teams, which suggest that focusing on human and social factors is necessary to succeed.

In summary, it is clear that the picture for XP is not clear with a number of moderating factors (Hannay et al., 2009) that affect the level of benefit of applying the XP methodology. The evidence suggests that agile methods aren't necessarily the best choice for large projects but more research is needed to better understand XP, its place in teams and the software development industry for greatest benefit. Dyba and Dingsoyr recommend that managers carefully study their projects' characteristics and make use of whichever method or combination of methods they believe is most appropriate for effective completion (Dyba and Dingsoyr, 2008).

Chapter 3: Transactive Memory Systems

3.1 Introduction

One of the foci of this thesis is to understand transactive memory systems in software development teams. This chapter gives an overview of the current research relating to teams, team cognition, transactive memory systems, and finally research focussing on this in software development teams. It provides the necessary background and identifies the key issues that will be investigated by the agent-based model.

3.2 Teamwork

In the modern world organisational teamwork is carried out in every type of business and teams are now expected to perform across functional, disciplinary, technological and geographical boundaries, which introduce new challenges. However, working as a team does not guarantee success and as organisations have grown the importance of effective and efficient teams has become clear. As the functioning of teams is a complex and dynamic subject relating to both social and cognitive processes, understanding how and why they function or not is an ongoing area of study.

Cannon-Bowers and Salas (2001) defined teamwork as achieved when members interact interdependently and work together toward shared and valued goals. Further, teamwork involves the adaptation of coordination strategies through closed-loop communication and a sense of collective orientation so that they can reach those goals (Salas and Fiore, 2004). Teamwork can be described as the process of carrying out a joint task and it relates to the quality of interaction and collaboration in a team, which is often task independent. Teams and teamwork have been extensively studied from many different perspectives (Cooke et al., 2000, Cooke et al., 2003, Hoegl and Gemuenden, 2001, Hoegl and Parboteeah, 2007, Kang et al., 2006, Salas and Fiore, 2004, Salas et al., 2008) with team cognition often being a focal point (Salas and Fiore, 2004).

3.3 Team cognition

The additional complexity of working in a team requires an additional layer of cognitive demands for team members to manage team processes such as planning, decision-making and problem solving (Cooke et al., 2003). It is increasingly becoming accepted that an individual's cognition is a product of their social environment and interactions with others (Salas et al., 2008). An obvious extension to this is the concept of shared, or team, cognition, which is considered to be cognition at the team level over and above individuals' cognition (Klimoski and Mohammed, 1994, Theiner et al., 2010).

Team cognition has been described as rule based performance in fairly stable routine environments, and the term macro cognition has recently been used (Fiore et al., 2010) to distinguish a more complex collaborative cognition. Macro cognition emphasises expertise out of context, where novel tasks are addressed outside routine environments and new knowledge and performance processes are generated. It is distinct from team cognition in that it defines the generation or adaptation of rules to novel situations and externalises internal knowledge through individual and team knowledge-building processes. This may be particularly relevant for software development teams that often operate in novel and innovative environments.

Understanding team cognition can be beneficial in three different ways. Firstly, it can explain how members of high performing teams interact with one another and therefore what differentiates high and low performing teams. Secondly, it can allow prediction of a team's ability to fulfil a given task, and thirdly it can allow the diagnosis of team problems and insight into how to solve them, leading to interventions to improve performance (Cannon-Bowers and Salas, 2001).

A number of studies have linked team cognition with team performance (Ancona and Caldwell, 1992, Cooke et al., 2003, DeChurch and Mesmer-Magnus, 2010, Hoegl and Gemuenden, 2001, Rentsch and Klimoski, 2001). Firstly team cognition is expected to directly influence task performance by affecting factors such as output, time and quality. Secondly, team cognition is expected to improve team processes resulting in enhanced communication and coordination leading to better team performance. Finally, more generic aspects of team functioning can be improved by team cognition such as morale, motivation and job satisfaction, which it is argued, are loosely related to team performance (Cannon-Bowers and Salas, 2001).

Team cognition has been described in the literature in many different ways including shared mental models, team member schema agreement, team situation awareness, transactive memory systems, mutual knowledge and collective mind (Cooke et al., 2003, Klimoski and Mohammed, 1994, Levesque et al., 2001, Lewis, 2003, Lewis, 2004, Rentsch and Klimoski, 2001, Salas and Fiore, 2004) resulting in a broad definition of team cognition. Each construct has unique aspects but they all incorporate the effect of common knowledge between members with transactive memory systems focusing on the location of knowledge and expertise in a team.

3.4 Transactive memory systems

"Knowledge is of two kinds. We know a subject ourselves, or we know where we can find information on it."

Samuel Johnson (English author, critic, & lexicographer, 1709 - 1784) (Boswell, 1986)

Transactive memory is an individual's beliefs relating to "who knows what" in their team and a transactive memory system (TMS) is the transactive memory, communication, and coordination at team level (Wegner, 1995). Over the course of a project, team members increase their beliefs about each other, allowing more efficient coordination and communication. A TMS allows members to gain access to a knowledge base larger and more complex than any one person alone could possess.

Definition: Let there be N members of a team, and suppose that there are M areas of expertise that are relevant.

$e_{ij}^{k}(t)$ represents the level of expertise of team member *j* at time time *t* in area *k* according to team member *i*, $e_{ij}^{k}(t) \geq 0$.

The TMS of the team at time *t*, is:

$$T(t) = (T_1(t), \ldots..T_N(t)) \text{ where } T_i(t) = ((e_{i1}^{1}(t),\ldots e_{i1}^{M}(t)), \ldots., (e_{iN}^{1}(t)),,\ldots, e_{iN}^{M}(t)))$$

Transactive memory systems benefit teams in a number of ways (Akgun et al., 2005):

- Help team members compensate for one another
- Reduce each team member's cognitive load
- Provide access to an expanded pool of knowledge and expertise
- Decrease redundancy of effort
- Facilitate sharing and dissemination of tacit knowledge relating to different knowledge domains
- Allocate resources according to team member expertise.

The development of a TMS will be influenced by the initial conditions of the team and the task processes that the team undertakes. To respond to a task requirement a team must assess what knowledge is required, where that knowledge resides in the team and then bring that knowledge to bear. The mere presence of knowledge in a team is not sufficient for the efficient functioning of the team, the team members must know where knowledge is located and have the ability to utilise that knowledge when it is required (Faraj and Sproull, 2000, Yuan et al., 2007).

Task interdependence and cooperative goal interdependence both have a positive relationship with TMS, which has been found to mediate the relationship between these factors and team performance (Zhang et al., 2007). Cognitive interdependence within a team is described as when an individual's contribution to the team is dependent on their own and at least one other's input (Hollingshead, 2001) and cooperative goal interdependence refers to team members' perceptions of how their goals are related to the goals of other team members. Cooperative goal interdependence occurs when team members believe that personal attainment of goals helps others achieve their goals and that this influences expectations, communication and performance within the team (Zhang et al., 2007).

For cognitive interdependence to be essential the task requirements of the team must be sufficiently complex, and each team member needs to know about teammates' knowledge, have the ability to anticipate how the teammates will behave, and have trust in the influence that the teammates will exert on the outcome of the shared task (Akgun et al., 2005). The knowledge that each member of a team holds is likely to be useful to that team only if the other team members are aware of that knowledge. As group members start working together their knowledge of each other develops and this can increase the potential for cognitive interdependence resulting in improved team performance (Brandon and Hollingshead, 2004).

A further essential element for cognitive interdependence is knowledge relating to team processes allowing the effective coordination of task fulfilment. Early research on TMS argues that team coordination is an important element of TMS and essential for team performance. The transactive memory system not only enhances a team's ability to respond to the task, but to respond to the processes required to coordinate the team in order to fulfil the task. It has been found that TMS and subsequently team performance are improved not only by training in task skills (Liang et al., 1995, Moreland, 1999) but also training in team skills (Prichard and Ashleigh, 2007).

As the knowledge and expertise in a team develops there can be a tendency for experts to emerge as tasks are directed at the team member perceived to have expertise in a particular area. This is known as differentiation (Gupta and Hollingshead, 2010, Hollingshead, 2001). For differentiation structures to develop there must be cognitive interdependence and team members must have convergent expectations that others will learn in areas of relative expertise. As team members' beliefs relating to each other's knowledge become more accurate and complete over time this produces consensus between teammates, resulting in convergent expectations. This means that, as well as developing differentiated knowledge structures, the team will also be developing a shared or integrated set of knowledge relating to "who knows what" (Brandon and Hollingshead, 2004). However, according to Faraj and Sproull (2000), the mere presence of differentiation is insufficient, accurately knowing the location of knowledge is essential. Accurate recognition of a differentiated knowledge base facilitates sharing of tasks, reduction of workload and more effective accomplishment of team goals.

There is some dispute over the benefits of differentiation. Some research shows that diverse teams take longer and have more difficulty in solving problems and sharing information (Rico et al., 2008). Rico et al argue that if knowledge is similar then team members develop similar opinions and attitudes to the task and task processes but if knowledge is very diverse and discrete then the unique perspectives may introduce an unwillingness to accept new ideas or to exchange task critical information. An alternative, more prevalent view suggests that performance advantages deriving from TMS are greater when group members have higher differentiation (Austin, 2003, Larson, 2007, Lewis, 2004, Littlepage et al., 2008). Austin posits that differentiation means that it will be easier to recognise experts and as such the team members are more likely to agree on the distribution of expertise in the team. In support of this Lewis found that teams with initially diverse knowledge were better able to develop TMS than teams with overlapping knowledge and that a stronger TMS was positively related to team performance.

One study found that, as a team worked together, differentiation increased such that their roles became increasingly specialised resulting in lower levels of interaction within the team (Levesque et al., 2001). Levesque et al. suggest that for temporary teams this divergence may be highly functional and lower interaction reduces losses from coordination and team processes. This, however, may not be the case for teams with longevity where low levels of differentiation, at least early in the project, may encourage interaction and improve team processes. It has been suggested that teams need to build a TMS early in the project lifecycle, during the planning phase, and develop a mature TMS during the implementation phase, in order to reap the rewards of an effective TMS later in the project (Lewis, 2004). An expectation of continuing to work together has been shown to increase coordination and communication in a team, and an expectation of increased task difficulty affected how the team coordinate their efforts (Baumann and Bonner, 2011) suggesting that groups do not choose coordination strategies blindly; it is possible that they may consciously, or unconsciously choose good coordination strategies based on their expectations of the team.

The situation regarding differentiation may not be as simple as described above. Hollingshead (2001) found that the optimum level of differentiation for highest performance is dependent on the task requirements, the anticipated project outcome, work relationships, the motivation of the team members to learn different or similar knowledge and various other antecedent factors.

Some previous studies have started to look at the antecedent factors to the development of transactive memory systems and hence team performance. Akgun et al. (2005) looked at task complexity, team stability, team member familiarity and interpersonal trust. Others include team structure and resources (Ren et al., 2006), task interdependence, cooperative goal interdependence and support for innovation (Zhang et al., 2007), experience (Reagans et al., 2005) and team process skills training (Prichard and Ashleigh, 2007). It has been found that team size has a negative relationship with TMS; smaller groups benefit more from TMS in terms of performance than do larger groups (Palazzolo et al., 2006). This list is not exhaustive and it demonstrates the complexity of the interdependencies that impact on transactive memory systems and ultimately team performance.

Numerous studies have shown that a good TMS is positively related to team performance (Akgun et al., 2005, Alge et al., 2003, Austin, 2003, Jackson and Moreland, 2009, Lewis, 2004, Littlepage et al., 2008, Maruping et al., 2009, Zhang et al., 2007) as TMS allows groups to utilize member expertise more effectively to achieve higher levels of performance. More specifically the aspects of accuracy and completeness of TMS appear to have an important positive influence on team performance (Austin, 2003, Lewis, 2004).

In order to evaluate TMS research one has to be aware of the variation in its definition and measurement across different studies, and compare appropriately. Three methods have been utilised to determine its constitution: those of recall, observed behaviour and self-report about team members' expertise. Recall tests infer levels of TMS by measuring quantity, content and structure when team members are asked to remember information individually or as a team (Hollingshead, 1998). Other researchers claim that TMS can be measured from three factors observed from behaviour namely, specialisation (the differentiation of members' knowledge), credibility (the levels of belief in the reliability of members' knowledge) and the coordination of the team processes (Liang et al., 1995). More recently self-report measures have been developed that ask questions relating to perceived knowledge and processes within a team however these have to rely on accuracy and veracity.

Using self-report measures Lewis (2003) and Austin (2003) measured TMS albeit each defining different elements. Focussing on the structure of the team knowledge, Lewis defined transactive memory systems within the three dimensions of specialised expertise, credibility and coordination; Austin, looked more at content and defined TMS as knowledge stock, knowledge specialization, transactive memory consensus and transactive memory accuracy. Both Lewis (2003) and Austin (2003) found that transactive memory systems have a positive relationship with group performance and Austin found that transactive memory accuracy is the most significant predictor of team performance.

Finally, it has been suggested that studies have portrayed an overly optimistic view of the benefits of TMS as much of the focus has been on performance. Less research has been carried out on the mechanisms that prevent TMS from forming and functioning in teams (Peltokorpi, 2008).

3.5 Focus on software engineering teams

The cognitive processes that take place in teams are well studied (Salas and Fiore, 2004) but less work has been carried out relating to the internal cognitive processes that are utilised by software development teams when solving complex problems.

The work carried out by software engineering teams requires very specialist skills and is often creative, complex and unstructured, requiring teams with high levels of coordination and shared cognition. In fact, expertise coordination has been found to be more important to team performance than input characteristics, presence of expertise and administrative coordination in software development teams (Faraj and Sproull, 2000). This is an important finding implying that internal cognitive processes have a greater impact on team performance than demographic factors. This is supported in a study by Kang et al. kang 2006 who found that cognitive factors in software development teams had more impact on team effectiveness than the demographic factors of age, tenure and gender.

The importance of teams in organisations that rely on creativity, knowledge and innovation has been documented (Akgun et al., 2005, Ancona and Caldwell, 1992, Faraj and Sproull, 2000, Kang et al., 2006, Lewis, 2004). Team collaborative processes can influence the relationship of team processes, performance and team member satisfaction with creative skills. Creative processes during development need less collaboration but during discussion and elaboration they need more collaboration. This would indicate that team processes should include less collaborative sections to optimise the creative process (Hoegl and Gemuenden, 2001, Hoegl and Parboteeah, 2007).

Software development projects can vary from highly innovative when designing and developing new products to relatively routine tasks such as maintenance and upgrades. There is some support to suggest that the innovativeness of the task determines whether collaboration is beneficial to performance (Hoegl et al., 2003), with high levels of innovativeness demanding high quality collaboration for high performance. In fact the quality of early collaboration may be an early indicator of the success of the project. Hoegl and colleagues go so far as to suggest that there may even be a negative relationship between teamwork quality and performance for routine projects with low levels of innovativeness.

Levesque (2001) looked at newly formed student software development teams and found that over time interaction and communication was reduced and each team member's role became increasingly specialised. They propose that the team adopted a black box approach where each team member had their own specialised deliverable for the project. This was identified as characteristic of temporary, task focused work teams who have not had the opportunity to develop strongly shared cognition, whereas teams with longer tenure develop increased levels of communication and interaction. This may be particularly applicable in the software industry where teams are often reformed for each project.

There is a clear link between team cognition and team performance with processes including communication, coordination and cohesion being positively linked to team performance (Espinosa et al., 2007b, Hoegl and Gemuenden, 2001). In fact, more specifically it has been found that awareness of expertise location, shared task understanding and frequency of meetings all have positive and significant effects on team performance in software development teams (He et al., 2007), however, the benefits of team familiarity can be reduced with increasing task complexity (Espinosa et al., 2007a). Moreover, the effects of familiarity fade over time with familiar teams and unfamiliar teams achieving the same levels of cognition (He et al., 2007). This implies that any TMS developed prior to forming the team due to the members being known to each other is superseded and that the TMS catches up in the unfamiliar teams. This is an important finding for the software development industry where teams are very often formed in a transient way for each particular project. Due to this common practice in the industry there is a need for more research into how the transient nature of software development teams affects the development of TMS and subsequently performance, particularly in highly innovative environments.

There is very little research into the cognitive processes that take place in teams utilising different software development methodologies. Maruping et al. (2009) investigated how Agile development practices facilitate the coordination of expertise in software development teams, specifically focusing on two elements of XP. They found that knowing the location of expertise weakens with increasing levels of collective ownership of code and so argued that collective ownership of code substituted for expertise coordination (TMS). They also found that coding standards enhanced the effectiveness of expertise coordination (TMS). They question the work by Faraj and Sproull (2000) who reasoned the coordination is especially important when teams perform complex and unpredictable tasks. Maruping et al. suggest that Faraj and Sproull did not consider internal team processes of roles and responsibilities and also how expertise coordination (TMS) is affected by standardisation of work. Their paper contributes to the TMS literature by arguing that critical contextual factors, such as XP practices, affect the efficacy of TMS in software development teams. This study provides some initial insight into the effect of using different software development techniques on TMS and knowledge structures within a team but more research is required.

3.6 Conclusion

In summary, there is a complex and not fully understood relationship between TMS, differentiation and team performance. The presence of differentiation is insufficient as it can only be useful if there is a well-developed TMS, but there is dispute regarding the optimum levels of differentiation for best performance.

As much of TMS research has been carried out in routine, simple team and task environments it is particularly important to more fully understand the role of TMS and differentiation in the novel, more innovative environments of software development teams. Additionally the nature of the development of TMS, differentiation and performance of software development teams over time is of interest as a particular feature of the software industry is that teams are often transient and newly formed for particular projects. By understanding the mechanisms of cognition within these teams over time some insight can be gained to allow greater efficiency of team formation and operation. Additionally, the study of TMS and differentiation is scarce in the specific area of eXtreme programming. XP techniques may introduce factors that change the behaviour of TMS and differentiation that will, in turn, impact on performance; the agent-based models used in this research aim to investigate this.

Chapter 4: Agent-based modelling

4.1 Introduction

Agent-based modelling is increasingly being used as an analytical tool in the social and cognitive sciences, and it is the methodology used for this research. This chapter gives an overview of the method, describes the property of emergence and provides some insight into the iterative process of agent-based modelling. Also, this chapter provides a basic review of some of the modelling environments that are being used in the literature, with a brief introduction to FLAME, the modelling environment of choice for this research. Finally, this chapter considers previous research using agent-based modelling to investigate social systems, cognition and transactive memory systems, and discusses the issue of cognitive realism in agent-based modelling.

4.2 Background

For a system or phenomenon that is too complex or inaccessible to observe in detail directly, often a simple model can allow a scientist to understand how it works. Examples include European economics (Deissenberg et al., 2008), social behaviour (Goldstone and Janssen, 2005), cognition (Palazzolo et al., 2006), and biological systems (Coakley et al., 2006a). Mathematical models have been used extensively to predict and describe a system and increasingly computational models are being used that can simulate behaviour in a system allowing it to be observed and measured over time. The use of agent-based modelling (ABM) allows experiments to be designed and the results interpreted through examining the way the model behaves.

All ABMs work by synthetically constructing virtual versions of social phenomena from low-level descriptions of the individual agents (Goldstone and Janssen, 2005).

Multi agent systems typically are defined as (Fan and Yen, 2004):

- Having no explicit global control.
- Having distributed resources, expertise, intelligence, and processing capabilities.

- Working in an open environment full of uncertainties.
- Emphasising social agency and social commitments.

An agent is a small autonomous software program that has internal memory and decision-making functions, which communicates with other similar agents. As they communicate with each other, agents exhibit behaviour that follows the rules encoded within their functions. The rules have to be defined carefully as the choice of rule selection scheme can have a large impact on the way the model works (Boero et al., 2008, Juran and Schruben, 2004, Phelan, 2002). As the individual agents operate, the system as a whole may exhibit emergent behaviour.

The individual agents are said to be acting at the lower, micro level of a system and the emergent behaviour is said to be acting at the higher, macro level. Agents only act in their immediate environment, in their own interests. Agents may also experience a lifecycle comprising being created, learning, adapting, reproducing and dying.

Agents are often described as having four features (Gilbert, 2008):

- Autonomy: There is no global controller operating the agents, they operate independently within the parameters provided to the agent and its immediate environment.
- Social ability: It is able to interact with other agents.
- Reactivity: It can react appropriately to stimuli from its environment.
- Proactivity: It has its own goals that it pursues according to its own rules.

There are many benefits to ABMs (Fum et al., 2007). By modelling a system the modeller is forced to decompose the system being modelled and by extension needs to fully understand that system. Computers will not tolerate any vague or imprecise instructions therefore the model will be based on logical reason producing clarity and completeness. As a product of this any theoretical output will also be based on logical, clear statements (Gilbert, 2008). Secondly, ABMs enable researchers to experiment with highly complex or inaccessible systems. This can take place in a controlled manner without the messiness of the real world allowing isolation of the particular process being analysed (Gilbert, 2008). Additionally a model can provide data at both the agent level and the system level. And finally, the property of emergence is perhaps the major benefit to agent-based modelling, allowing new ways of understanding complex systems or phenomena that may have been unexpected.

There are some limitations of ABMs that should be highlighted. Of course they are abstractions of human phenomena and the extent to which we believe the results depends on how well we feel the model simulates reality (Hutchins, 1992). Sometimes small changes to rules, seemingly irrelevant to the theory, will make large differences to the output of a model (Phelan, 2002) and of course, one must always be aware of Bonini's Paradox (Fum et al., 2007). This is the tendency of models, as they become more realistic, to become more and more complex, until they are so complex that they are as difficult to understand as the real world system being modelled. For this reason a trade-off has to be made between simplicity and realism, by creating a model with sufficient realism to provide a useful outcome but simple enough to allow analysis and interpretation of the results. Often complex models can produce huge amounts of output that are difficult to analyse, and it can take a long time to execute a simulation if you have many complicated agents. For this reason parsimony is crucial; it is important to select simple solutions that address the questions without additional complexity (Bryson et al., 2007).

Agent-based models are highly specialised and therefore can be difficult to understand by, not only those unfamiliar with agent-based modelling, but potentially anyone not closely involved in the development. Nearly everyone can understand narrative but only those familiar with programming languages and environments may have the ability to understand the intricacies of a model. It is argued that in translating conceptual verbal theories into a computational model there are a number of degrees of freedom (Sun, 2009). All models have to be logically specified, with decisions made for every stage to ensure the model runs correctly. For this reason the implementation of a model from a verbal conceptual theory may not be a direct mapping of theory to model. Often there are gaps that require assumptions to be made resulting in minor 'enhancements' to the theory. Two models written independently using the same theory may very easily have differing assumptions built in culminating in different simulation results. Equally, a verbal conceptual theory can include terms such as trust that could be interpreted differently by different model designers. This is also likely to result in differing models and subsequently different output. Awareness of this risk is essential when comparing the output from two models purportedly modelling the same theory.

ABMs have been used to simulate, amongst others, ants (Bicak, 2010, Gheorghe et al., 2001, Jackson et al., 2004), cells, (Fisher and Henzinger, 2007, Pogson et al., 2006, Walker et al., 2004, Walker et al., 2006) economics (Farmer et al., 2005) and the human immune system (Baldazzi et al., 2006).

4.3 Emergence

Often models are used to investigate emergent behaviour. Emergence is the way that complex systems and patterns emerge out of many simple interactions with no central command or control. These complex patterns have their own behaviour that is characteristic of an autonomous entity. The emergent behaviour is grounded in, yet transcends, the behaviour of the contributory agents.

The many simple entities that interact to generate emergent behaviour will operate according to simple rules. As the number of agents increases the number of potential interactions becomes very large indeed. Sometimes top down feedback takes place and the emergent behaviour will influence the individual agents to adapt, in turn affecting the emergent behaviour causing growth or evolution of the system. A simple demonstration of emergence was The Life Rules invented by John Conway in 1970 (Resnick, 1996). This was a simple board based environment where counters were given simple rules on how to act. This gave rise to life-like structures and complex behaviours such as moving, growing, reproducing and evolving.

There are examples of emergent behaviour everywhere, including nature, physics, biology, economics, and technology. The emergent effect of a Mexican wave at a football match is the individual actions of many people acting according to one simple rule; lift my arms up and down once, after the person to my left does the same. Termite mounds illustrate emergent behaviour; forms develop as a result of the actions of many, many termites; and there are examples of emergent behaviour online. The rising popularity of certain videos on YouTube, the links created to certain pages on the Internet, emerging trends on Amazon, and the maintenance and development of Wikipedia all exhibit properties of emergence where there is no central control but patterns arise out of the actions of many individuals. Other examples of emergent behaviour can be seen in ant colonies, flocks of birds, traffic behaviour, economic systems, consumer behaviour and weather.

4.4 Cognitive realism

A topic that is hotly discussed in agent-based modelling circles is cognitive realism. It has been argued that cognitive realism is important for modelling psychological processes in order to more closely capture human behaviour (Sun and Naveh, 2004). There are instances where complex models that purport to be cognitively realistic have produced results that are close to empirical results. Sun and Naveh (2004) created an agent-based model to model the growth of academic science by simulating authors of academic papers and the environment around the publication of journal papers. The agents in this model attempt to publish papers by using previously published papers, cognitive processes, learning and decision making to refine ideas and generate new work for publication.

Successful agents publish many papers and unsuccessful ones fade away, with measures of cognitive ability based on number of published papers. By varying factors relating to cognition the effect on performance can be said to be a function of cognition.

It can be argued that a model can never realistically model human cognition in all the complexity, unpredictability and variety that it contains. It is not possible to model all the external factors relating to cognition in addition to the traditional factors of memory, language, judgement, creativity, emotion and perception (Eysenck and Keane, 2000). This would be an example of Bonini's Paradox (Fum et al., 2007), a model so complex that it would be as difficult to understand as the real world system being modelled.

Relevancy can be obtained with agent-based models that are not necessarily close to reality but a reasonable abstraction, where the focus must be the relevancy of the questions being asked. Although realism in computational modelling is germane, parsimony is preferred for a number of reasons. A model should be as complex as the question being asked requires and no more. Additional complexity introduces additional cost in financial and temporal terms, provides extraneous and overly complex output, and is more difficult for others to understand (Bryson et al., 2007, Burton and Obel, 1995).

It is asserted here that it is not necessary to model highly complex models and that it is more productive to start with a simple model and gradually increase complexity and sophistication over subsequent development cycles of the model. It is possible to interpret and understand the cognitive and social properties of agents without developing, from the start, complex cognitive architectures, which are difficult to evaluate, use and analyse (Boero et al., 2008, Deng and Tsacle, 2006, Marks, 2000).

4.5 The modelling process

At the start of the modelling process various parameters have to be identified to specify the agents, the environment, rules and data structures. Also, the behaviour of the agents, their goals and actions need to be defined. In order to do this the modeller requires a detailed understanding of the system being modelled and it is normal to start with an extremely simplistic model (Grimm, 1999). This allows the modeller to check that the basic concept is sound and gives some scope for initial tests of validity of the model. A simple model is easier to test and validate than a complex one.

A simple hypothesis can be defined with a prediction of how the first simple model will behave. If the results are not as expected then either the model is faulty or there is some genuine emergent behaviour being exhibited. The reason for this can be pinned down by subtly altering some aspect of the model to try and find the cause of the unexpected behaviour.

From here the modelling process is an evolutionary one. Further complexity can be added to the model and it can be tested, hypotheses defined and results gathered. The next steps are to systematically vary the agent parameters to establish the sensitivity of the model and how it will behave under different circumstances over time. This builds an increasingly detailed picture of the system being modelled and allows experimentation to take place under known conditions.

This process is iterative and as it is repeated the model will become more detailed and refined. In fact:

> *"The model can serve as a kind of rough draft of the theory allowing a quick preliminary check on many of the consequences of a theory. If the model makes predictions that are obviously incorrect, that difficulty can often be uncovered in a less painful and time consuming fashion when simulating" (Lehman as cited in (Fum et al., 2007))*

The relationship between model and empirical or academic research is not one-way. Empirical or academic data will validate the model but data from the model may open up new lines of research, which will in turn feed back into the model design. The relationship is a circular one.

A model cannot generate facts but can test a hypothesis from which the model was developed. It allows the derivation of predictions that answer 'what if' questions, with alternative studies needed to validate the model. So the aim is to produce a dynamic simulation of a finite set of parameters that interact via a group of agents. This allows the researcher to identify dependencies and effects, and go on to further refine the model.

It is important to recognise that the model is not an end point to research but a tool that can generate phenomena and pose questions leading to further investigation. Computational models can highlight those aspects of a theory that would benefit from further theoretical development and investigation.

4.6 Modelling social systems

Agent-based modelling is useful in many domains but is increasingly being used for the analysis of complex social systems (Axtell et al., 2001, Bonabeau, 2002, Deng and Tsacle, 2006, Goldstone et al., 2008, Panzarasa and Jennings, 2002, Purnomo et al., 2005, Rouchier et al., 2001, Smith and Conrey, 2007, Sun, 2006). These applications include crowd behaviour, traffic management, stock market staff behaviour, operational risk in banking, social networking, social influence, population growth in disadvantaged societies, and forest management in Indonesia.

The ability of agent-based models (ABM) to demonstrate and deconstruct emergent behaviour is attractive to social scientists as this is often the focus of their work (Gilbert, 2008). ABMs are particularly useful for modelling social behaviour when:

- It is unethical or illegal - for example riot behaviour or the spread of disease, in order to study behaviour.
- It is not possible to examine behaviour due to financial or political constraints - for example social behaviour worldwide or in war torn countries.
- What-if scenarios are not possible to create in reality - for example organisational behaviour under different extreme conditions that would jeopardise the business if attempted in reality.
- When the population being studied is heterogeneous, complex and nonlinear - agent-based models can isolate the particular attributes being addressed and simplify the analysis whereas studying the population in reality would be difficult due to the complexity.

Agent-based models of social behaviour do not attempt to model humans entirely as that would firstly, be impossible due to the complexity of humans and social networks, and secondly, the model would not be useful as it would be as complex as the system being modelled (Deng and Tsacle, 2006). So, models concentrate on the particular parameters or aspects that are the focus of the researcher's work (Gilbert, 2008).

Models can simulate knowledge, communication, beliefs, attitudes, intentions, desires and opinion. Examples of agent-based models simulating social systems are:

- Axelrod's culture model (Goldstone and Janssen, 2005) simulated cultural imitation or social influence and found explanations for bandwagon effects, clustering of opinions and spontaneous division of culture into subcultures. This model may be too simplistic to model real world patterns but it generates questions and suggests avenues for further research.
- Another agent-based computation by Goldstone and colleagues (Goldstone et al., 2008) used an Internet based platform in real time to look at foraging behaviour in terms of beliefs, goals and cognitive capacities. They also found the emergence of bandwagon effects along with population waves and spontaneous specialisation.
- Computational modelling of communication between networks (agents) by Hutchins

> (Hutchins, 1992) found that even when the cognitive properties of the individuals in a group are held constant the group may display different cognitive properties if the communication within the group is organised in different ways. He concludes that the cognitive properties of a group are not only fashioned by the cognitive properties of the individuals in the group but also by the properties of the interactions between group members.

Traditionally, psychologists support the scientific method of causation and explanation seeking covariation between variables through the application of statistics. Agent-based modelling can often provide a more detailed explanation and a deeper understanding of the specific mechanisms that cause an effect (Smith and Conrey, 2007). This is often preferable to social psychologists that are used to thinking conceptually of the underlying cognitive processes giving rise to behaviour or interpersonal activities that underlie social or group phenomena.

Agent-based modelling is well suited to addressing some of the problems that social psychologists encounter in the following processes:

- Intrapersonal - decision-making, memory, personality.
- Interpersonal - social influence, liking, conflict, emotional contagion, social influence.
- Group– leadership, commitment, status.
- Intergroup – bias, discrimination, anxiety.
- Social and cultural – social roles, influence, contagion.

In fact one of the advantages of agent-based modelling is that it can bridge these theoretical levels and reveal interdependencies between them. This is particularly useful when looking at the relationships between individual and group cognition.

There are criticisms of the ability of agent-based models to effectively model adaptive human behaviour. Although agent-based models have been, and continue to be, used successfully for analysing social systems it has to be recognised that there are issues related to modelling human beings. Unlike more predictable entities such as cells or economics, human beings have complex psychology and often exhibit irrational behaviour or make subjective choices. In this respect, modellers must be aware of the varying degrees of completeness of the input to the model in relation to the output of the model. This often cannot be quantitative and must be viewed qualitatively.

4.7 Modelling cognition

For some time social scientists studied social systems using social simulation and psychologists studied cognition using cognitive modelling. More recently the two disciplines have integrated and ABMs have been developed that simulate social situations incorporating agents that are more cognitively realistic (Sun and Naveh, 2004). This can extend the advantages of cognitive modelling to allow the influences of social interaction to be studied alongside agents' internal processes.

As it is not possible to observe cognition directly research has to observe behaviour and make inferences relating to the processes operating. This is relevant to cognitive science as traditionally cognition has been viewed as a property of an individual mind and there is a growing realisation that cognition can be a social phenomenon resulting in individuals' interaction with others. One of the strengths of agent-based modelling of collective cognition is the ability to find connections between individual cognition and interaction patterns, and then investigate the conditions under which these take place (Hazy, 2007, Panzarasa and Jennings, 2002, Phelan, 2002). ABMs of collective cognition allow the researcher a clearer picture of the relationships between individual cognition and collective cognition (Sun, 2006).

Team cognition has been the subject of agent-based modelling (Dionne et al., 2010, Fan and Yen, 2004, Ioerger and Johnson, 2001, Kang, 2007, Larson, 2007, Zhuge, 2003). Larson developed an agent-based model investigating the relationship between problem solving ability diversity and team performance mediated by levels of communication. He found that communication improves the performance of diverse groups and worsens the performance of homogeneous groups because, according to Larson, diverse groups are able to build on one another's success via communication but knowledge of another's success in homogeneous groups narrows the range of solution alternatives. Dionne et al. (2010) modelled team and leadership properties including social network structure, heterogeneity of agents' domains of expertise and levels of mutual interest. Results indicated that participative leadership only had positive effects on team performance under certain conditions and under other conditions leader-member exchange produced better team performance.

A further team cognition model was developed simulating agents operating in the scenario of assessing the threat level of incoming aircraft (Kang, 2007). The hypothesis predicted that team interaction styles affect communication and team performance through enhancing or hindering knowledge sharing. Findings indicate that levels of cooperativeness and activeness have an effect on team efficiency and also that there are interaction effects of cooperativeness, activeness and information redundancy on team efficiency. Although the study was not focussed on transactive memory systems, Kang's model included agent meta-knowledge relating to the knowledge of other team members so, in essence, a transactive memory system.

4.8 Modelling transactive memory systems

As discussed previously, the knowledge in a team and the accuracy of information relating to who knows what in a team is directly related to the performance of the team. As knowledge becomes more differentiated and dispersed amongst greater numbers of people it becomes increasingly difficult to know the location of knowledge. The paradigm of transactive memory systems allows us to understand how this process operates and to improve knowledge and expertise sharing in teams and thus organisations.

There is a dearth of research using agent-based modelling to simulate transactive memory systems however there are at least two (Palazzolo et al., 2006, Ren et al., 2006). Palazzolo et al. explored, using Blanche, how initial knowledge profile, initial accuracy of expertise recognition (TMS accuracy), and network (team) size affected the development of the TMS as mediated by communication. They simulated teams with high and low TMS accuracy, high and low initial knowledge, and team sizes of 4 and 20 members. The transactive memory development was measured as the degree to which team members accurately perceived each other's knowledge and the extent of the differentiation of the team's knowledge.

The hypotheses considered the relationships between the three antecedent factors and communication and how communication affected the final differentiation and TMS accuracy. They expected that low initial differentiation, high TMS accuracy and small team size would result in higher communication density and that high communication density would result in higher TMS accuracy and differentiation.

Their hypotheses were all supported and in addition they modified their theoretical model to incorporate additional relationships that were identified by the computational model. Firstly, that there is a direct positive relationship between the level of starting knowledge and the final TMS accuracy meaning that high initial knowledge results in better TMS accuracy regardless of communication density; and secondly, they found direct negative relationships between both the starting TMS accuracy and team size with final differentiation. In other words larger teams are less likely to differentiate than smaller teams and teams that start off with high TMS accuracy are less likely to finish with differentiated knowledge, both regardless of communication.

This highlights a paradox in their results. The computational model showed a positive relationship between initial TMS and final differentiation mediated by communication, however, it also showed a direct negative relationship between initial TMS and final differentiation. So does higher initial TMS result in higher or lower final differentiation? Palazzolo et al (2006) do not posit an answer to this question but suggest that further investigation is required.

The Palazzolo et al. model (2006) investigates a simple set of relationships and reveals some additional unexpected ones. It shows the impact of initial conditions on a TMS and in addition demonstrates some of the benefits of computational modelling to logically validate verbally described TMS theories. The authors do point out, however, that the model was based on three knowledge areas, which they assert, is overly simplistic but does provide a good starting point for research in this area.

The other agent-based model exploring transactive memory systems (Ren et al., 2006) aimed to confirm previous research on TMS and extend it to apply further to more dynamic and diverse group settings. It is a more sophisticated model using ORGMEM, which is specifically designed to study how communication influences transactive memory and how the existence of transactive memory impacts group performance. This study used the parameters of low, medium or high task volatility (how frequently the tasks change), low, medium or high knowledge volatility (decay rates of knowledge) and as did Palazzolo et al. (2006), group size. The model simulated groups with little, moderate and extensive overlapping knowledge, and group size ranged from three members to 35. In addition they included conditions where group members had blank or complete initial TMS. This resulted in 108 different conditions for their model.

The paper does not identify their expectations but they found that the effects of transactive memory are contingent on group characteristics such as group size and environment, and the measurement that is used for performance. Results differed when performance was measured by time than when measured by quality. The study found that a good TMS is generally beneficial on performance but for advantages of time it is more beneficial for large groups, groups in a volatile knowledge environment and groups in a volatile task environment. For better decisions, a good TMS is more beneficial for small groups. They suggest that groups such as software development teams, which operate in an environment of high task and knowledge volatility, a high quality TMS would be beneficial. The results from this study demonstrate that agent-based modelling can provide some useful insights into the mechanisms of TMS and provide avenues for further research.

The two agent-based models focussing on TMS described above demonstrate initial forays into modelling TMS and each selected very specific avenues of research. Each paper took pains to point out that their model was a simple initial model and that it did not capture many aspects of TMS such as subjective judgement (Ren et al., 2006) and the ratio between numbers of knowledge topics and people (Palazzolo et al., 2006). Both papers assert that although their models were low in cognitive realism they are a starting point to using agent-based modelling to take the research into TMS further and develop models with increasing cognitive realism.

4.9 Conclusion

This chapter has briefly described agent-based modelling, the modelling process and identified some popular agent-based modelling frameworks in use. It has also outlined previous social and cognitive research using agent based modelling and argued for parsimony in modelling cognitive systems.

Increasingly agent-based models are successfully being used to simulate social and cognitive environments, however, modelling of transactive memory systems has attracted little interest with only two apparent previous models.

The two models described investigated the development of TMS, differentiation, communication, group size and external environmental factors, and looked at some simple relationships between them. They provided some initial investigations into TMS theory and produced some interesting results but had limited applicability in reality. Both models highlighted a number of questions relating to the many factors that affect a group's TMS and how that impacts on the group performance but there is much scope for further modelling of TMS and for that to be put into a more practical context. Unlike the previously described models, the agent-based model developed for this research places transactive memory systems in the realistic context of software development teams and looks at how TMS, differentiation and performance differ for teams using two different software development methodologies.

Chapter 5: Modelling environment

5.1 Introduction

This chapter will introduce the framework chosen, FLAME, and describe it in more detail with the reasons for its use in this research.

5.2 Some ABM frameworks

There are many agent-based modelling frameworks available with a number of researchers developing idiosyncratic systems being used for simulating a variety of cognitive phenomena. These include: ACT-R (ACT-R), Soar (Soar), (Kang, 2007), Clarion (Clarion), Blanche (Blanche), (Palazzolo et al., 2006), ORGMEM (ORGMEM), (Ren et al., 2006), Seleste (Phelan, 2002), Swarm, Repast, Mason, NetLogo (Macal and North, 2006), SIGMA, Brahms (Sun, 2006), ValSeek (Larson, 2007), CAST (Yen et al., 2003), EPICURE (Goldstone et al., 2008) and StarLogo (Resnick, 1996). For a review of a number of modelling environments see Ashworth and Carley (Ashworth and Carley, 2007).

5.3 FLAME

FLAME stands for Flexible Large-scale Agent-based Modelling Environment (Coakley, 2011) and is a generic, highly flexible environment, which is being developed at the University of Sheffield, in collaboration with the Science and Technology Facilities Council (Coakley et al., 2006b). Modellers from various disciplines have used FLAME for subjects as diverse as cellular biology, ant foraging behaviour and the European economy (Bicak, 2010, Coakley et al., 2006a, Deissenberg et al., 2008, Gheorghe et al., 2001, Jackson et al., 2004, Kiran et al., 2008, Pogson et al., 2006, Walker et al., 2004, Walker et al., 2006).

FLAME is based on a formal X-machines architecture (Coakley et al., 2006b, Holcombe et al., 2006). An X-machine is simply a device or model to describe a system that manipulates entities of type X, so for example, a word processor is an *electronic-document*-machine, and a calculator is a *number*-machine (Stannett, 2005). X-machines are general computational machines that were introduced by

Eilenberg (Eilenberg, 1974) and extended by Holcombe (Holcombe, 1988). They employ a diagrammatic approach of modelling and are capable of modelling both the data and the control of a system. A specific type of X-machine has formed the basis for a specification and modelling language to define and validate software systems. It is defined as follows (Coakley et al., 2006b):

Definition: A stream X-machine is an 8-tuple

$$X = (\Sigma, \Gamma, Q, \Phi, F, q_0, m_0)$$

where:

- Σ and Γ the input and output alphabets respectively.
- Q is the finite set of states.
- Φ, the *type* of the machine X, is a set of partial functions ϕ that map an input and a memory state to an output and possible different memory state, $\phi : \Sigma \times M \to \Gamma \times M$
- F is the next state partial function, $F : Q \times \phi \to Q$, which given a state and a function from the type ϕ determines the next state. F is often described as a state transition diagram.
- q_0 and m_0 the initial state and initial memory respectively.

(Coakley et al., 2006b)

Further, communicating stream X-machines have been defined (Balanescu et al., 1999) and it was found that communicating X-machines could be used to formally model multi-agent systems (Eleftherakis et al., 2005). Describing agents as autonomous communicating X-machines allows agents to work in parallel, an essential component of multi-agent systems, and have the ability to process information and communicate. Figure 3 represents an X-machine agent. It describes its memory, system states, input and output. The figure shows the transition between system state S1 (the initial state q_0) and S2 via the transition function F1. This transition function contains rules that can change the agent's memory, M to M', and send and receive messages via the input and output ports. These in turn are connected to message boards used for communication between agents. Agents send messages to global message boards that can be read by all X-machine agents in the system, and by giving each message the ID of the targeted agent each message is directed to the agent to which it was intended. There is a different message board for each type of message.

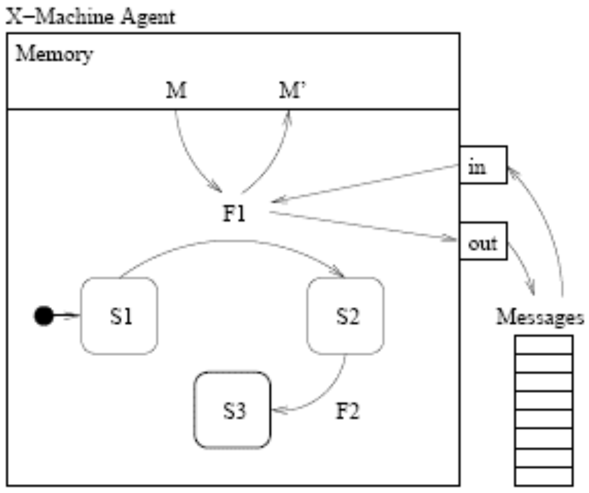


Figure 3: X-machine agent

The runs of a model are based on discrete time steps and each time step includes one transition function. The agents are processed in a random order so that no agent has priority over any other. The time step function receives any intended messages and after the function is processed, sends messages to respective message lists. Effectively the start and end of transition functions act as a way to synchronise the processing of X-machine agents and the communication of messages (Coakley et al., 2006b).

FLAME uses simple nested XML tags to describe the structure of the X-machine agents. This describes the environment with any constants in use in the model, the function files, and data types. It also includes a description of each agent with information relating to the transition function, start and end states, and input and output messages. Finally, it also includes a description of the message structures within the model. A simple structural example can be seen below:

```
<xmodel>
<environment>
        <constants></constants>
        <functionFiles></functionFiles>
        <dataTypes></dataTypes>
</environment>
<agents><xagent>
        <name></name>
        <memory></memory>
        <functions><function>
                <name></name>
                <description></description>
                <currentState></currentState>
                <nextState></nextState>
```

```
                    <inputs><input><messageName></messageName></input> </inputs>
                    <outputs><output><messageName> </messageName></output> </outputs>
          </function></functions>
</xagent></agents>
<messages><message>
          <name> </name>
          <description></description>
          <variables></variables>
</message></messages>
</xmodel>
```

The transition functions are written in C. In addition a start-up file is required to specify each agent's initial condition before a run of the model can start. This is also an XML file and defines the memory state of agents m_0. The core of FLAME constitutes of a specially developed parser that takes the X-machine description, and the transition functions, and produces an executable C program that can process a start-up XML file, run the X-machine agents and implement the message boards (Coakley et al., 2006b). Information, case studies, publications, installation files and instructions for FLAME can be found online at www.flame.ac.uk (Coakley, 2011).

The FLAME framework consists of a number of user-defined files, as follows:

- Model.xml: This file contains an XML description of the X-machine agent, model descriptions, memory variables, and definition of functions and messages.

- Functions.c: This file contains the c code that consists of the transition functions defined within the model.xml file.

- 0.xml: This is the start-up that file contains the initial information defining each agent at the start of the simulation. It contains information such as the initial contents of agent memory and state details.

This framework can be seen in Figure 4.

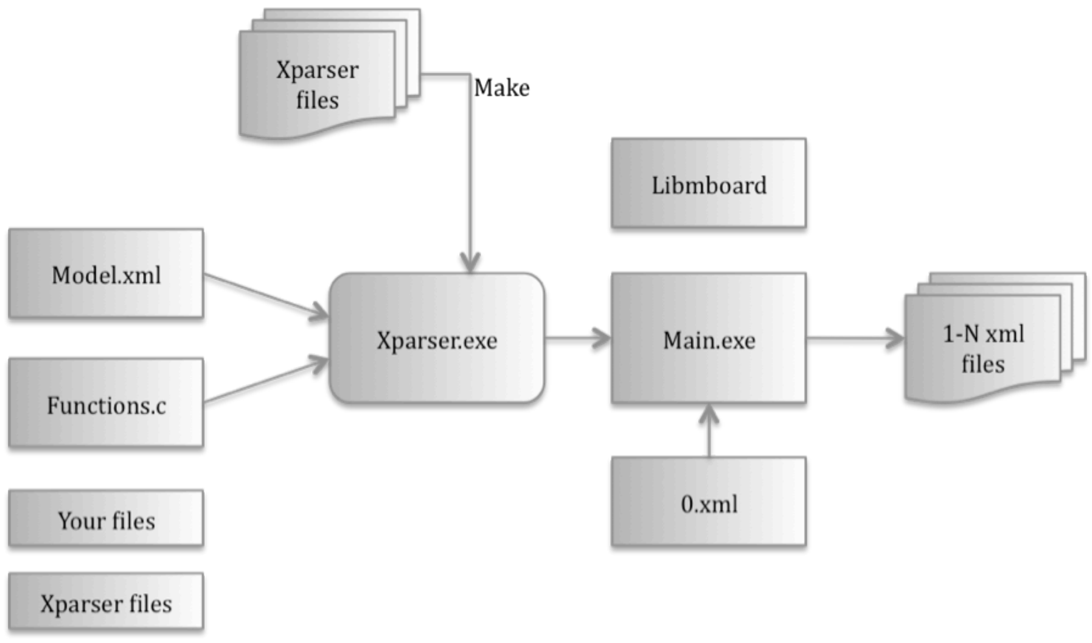


Figure 4: The FLAME framework (Bicak, 2010)

The model runs through a prescribed number of iterations where the agents interact with each other. Every iteration produces an automatically generated XML file containing all the memory data and state information for every agent (Kiran et al., 2008).

FLAME is the modelling environment of choice for this project for a number of reasons (Bicak, 2010).

- It has an extremely versatile testing environment with the facility to use various techniques.
- It is also a highly configurable and generic environment that allows high levels of innovation, unlike other systems that can restrict the user.
- As it is based on a formal X-machines method, specification and validation are inherently provided.
- Stategraph output showing states and transitions in each agent cycle are generated automatically.
- The syntax is based mainly on XML therefore it is easily extendable.
- It was developed at the University of Sheffield so the developers are on site, and there are a

significant number of experienced users that can provide expert advice and assistance with its use.

The conventional paradigm for using FLAME is to computationally outline a physical environment, where the agents are defined spatially, which then operate within three dimensions. In addition, the convention is to define large numbers of agents, potentially millions, for example cells, which can often require the use of High Performance Computing systems.

The model described in this thesis uses FLAME in a novel manner. Firstly, the agents are not defined within a three dimensional environment, they are defined with memory depicting human memory and knowledge, and the functions describe less tangible cognitive processes that take place in human teams. This model defines very small numbers of agents, the number of members in the team, in this case four. The model developed, however, has the capability to define larger numbers of team members.

Chapter 6: The Model

6.1 Introduction

This chapter will give a description of the model with a worked example to illustrate its operation. It will also detail a more formal structural description of the model in the form of tables including information held, messages sent and the logical flow of the functions. This chapter will also cover details of the verification and testing of the model.

The model is defined such that every simulation models a single software project with one team. There are two types of agents, a controller agent that manages administration within the model and player agents, which represent the team members carrying out the work. The player agents can hold elements of knowledge and levels of expertise for that knowledge, and in addition can hold beliefs relating to the knowledge and expertise of the other player agents. Work is issued to the player agents in the form of tasks where one task is issued to each agent per iteration of the simulation. Agents develop their knowledge and expertise as a result of processing the tasks and this can be done as solo agents or working in pairs. During the processing of the tasks agents also develop their beliefs relating to the other agents knowledge and expertise, their TMS, by sharing information and subsequently the distribution of the tasks is influenced by the TMS of the player agents. The output value of each iteration is a function of the level of expertise of each agent processing its task, and other measures of TMS and differentiation are also calculated.

6.2 Model description

Various parameters are defined that determine the initial general parameters of the simulation. They are:

- Task size - the number of elements of 'work' that are delivered to each agent per iteration.
- Team size - the number of player agents in each simulation.
- Random range - the range from which the tasks are randomly constructed from 1 to the random range.

- Working mode - pair or solo programming

Each player agent has the ability to have pairs of numbers in their memory. In each pair one number represents knowledge in a particular subject, and the other represents the level of expertise in that knowledge. The breadth of knowledge can grow as the player agents learn and the expertise levels can increase as they gain experience. Each player agent's memory also holds information relating to the knowledge and expertise of the other player agents in the simulation in the same way. This memory represents transactive memory and this can also develop over the course of a simulation as each player agent learns about other player agents' knowledge and associated expertise. The range of knowledge that a player agent can possess is limited by the random range; the task elements can be drawn from 1 to the random range. The player agents can start a simulation with no knowledge, expertise and transactive memory or can start with existing knowledge, expertise and transactive memory as predefined in the 0.xml file.

Each task presented to a player agent comprises a randomly generated series of numbers within the knowledge range determined by the random range. This represents the knowledge required to complete the task. Before the task is fulfilled the tasks are redistributed around the team to fulfil the task for the greatest value to the team according to the beliefs of the team. This process utilises the transactive memory; player agents will offer their task to other player agents that they believe will fulfil the task to a higher value than them. So as transactive memory develops the tasks should be redistributed more efficiently around the team.

For each player agent the task is fulfilled by calculating the task value, which is generated as a function of their expertise relating to the elements of the task.

The value is calculated as follows:

$$v = \sum_{i=1}^{ts} x_{k_i}$$

where:

- v is the value returned from an agent doing a task
- x is the experience value associated with knowledge element k in the task at position i in the task
- ts is task size

This means that high expertise player agents and those with expertise in more elements of the task will generate higher values towards the team output. This contributes to the output value for the team. For conditions where pair programming is utilised, agents will work in pairs and pool their knowledge and expertise to fulfil the task. For pair programming each simulation will be run for double the number of iterations as only half the tasks are delivered per iteration. This ensures that the same amount of work is delivered for the entire simulation.

As a task is fulfilled the agent increases expertise in the areas presented by the task. A number of methods of incrementing the expertise were tested including using percentages and various ranges of random numbers. Using a random number of thousandths between 0.001 and 1 produced sufficient randomness, and increments were not so large as to produce unmanageably large numbers by the end of a simulation. It also produced results that were plausible in terms of validating the model.

For each task fulfilled the player agent is given some information relating to other player agents' knowledge and expertise and updates its transactive memory accordingly. In pair programming conditions, if a player agent's partner is more expert in a task element the agent with the lower expertise will increase its knowledge by half the difference between the two levels of expertise. Again, a number of methods were tested and using half the difference provided results that were sensible in terms of validation. In reality this seems plausible, as working with a more expert person will increase skills substantially but is unlikely to improve skills sufficiently to match the expert.

Each iteration produces measures of output, knowledge differentiation and TMS. The controller agent aggregates the output from each agent and calculates team level results. The task output values for team members are aggregated to give a score per iteration. The optimum output score is also calculated as if the tasks were perfectly distributed in the team for maximum output. The ratio of aggregate score and optimum aggregate score is used as the measure of success. This ratio is a measure of learning as well as team efficiency in matching tasks to agents.

Differentiation is the variance of the knowledge within the team for each iteration. This relates to all team members across all knowledge streams and is a measure of the level of specialism in the team. High differentiation indicates the presence of specialists in the team and low differentiation indicates heterogeneous knowledge across the team.

Differentiation is calculated as follows:

$$diff = \frac{\sum_{i=1}^{t\times r}\left(x_i - \bar{x}\right)^2}{t \times r}$$

where:

- $diff$ is the differentiation: the variance of all expertise across all knowledge streams for all agents for one iteration
- x is expertise
- $\bar{x}$ is the mean of expertise for all knowledge streams for all agents in one iteration
- t is team size
- r is random range

TMS accuracy is the level of agreement of the team TMS with the actual knowledge of each team member. It is calculated by dividing the value of each belief in TMS by the actual knowledge to which it relates. The mean of these values is then calculated as the TMS accuracy for the team.

TMS accuracy is calculated as follows:

$$acc = \frac{\sum_{i=1}^{n}\frac{b_i}{x}}{n}$$

where:

- acc is TMS accuracy
- b is the belief in TMS of an agent relating to another agents' expertise
- x is the actual expertise to which the belief pertains
- n is the number of elements of belief held in TMS in the team

TMS completeness is a measure of the extent by which transactive memory in the team is filled. It is calculated by dividing the number of entries in TMS by the maximum possible number of entries.

Completeness is calculated as follows:

$$c = \frac{n}{\left(t^2 - t\right) \times r}$$

where:

- c is completeness of TMS
- n is the number of elements of belief held in TMS in the team
- t is team size
- r is the random range

6.3 Worked example

This is an example of how one iteration of a simulation may take place. For this example, there are four team members, task size is also four and the random range is between 1 and 10. Each of the agents has already developed some knowledge and expertise and some TMS.

1. The controller agent generates four tasks, each including four randomly generated numbers between 1 and 10.

2. The controller agent distributes the tasks randomly to each agent. So:

    - *Agent A receives – 1,4,6,5*
    - *Agent B receives – 5,7,10,2*
    - *Agent C receives – 3,4,8,3*
    - *Agent D receives – 9,5,1,4*

3. Each agent may decide to offer the current task to another agent if he believes that the other agent will fulfil the task to a higher value to the team.

- *Following agent A, the knowledge, expertise and TMS is as follows*:

Knowledge	1	2	3	4	5	6	7	8	9	10
Expertise	3.04	8.12	35.678			11.304	2.009	4.1		15.6
TMS – Agent B				11.67	10.12		1.5			
TMS – Agent C			3.66	4.29			17.005		23.5	
TMS – Agent D	12.4				2.23					

Table 1: Knowledge, expertise and TMS of Agent A

- *Agent A's current task is 1,4,6,5*
- *The value of agent A's current task is (3.04+11.304) = 14.344.*
- *The value of agent B doing agent A's task according to the beliefs of agent A is (11.67+1.120) = 21.79.*
- *The value of agent C doing agent A's task according to the beliefs of agent A is 4.29.*
- *The value of agent D doing agent A's task according to the beliefs of agent A is (12.4+2.23) = 14.63.*
- *Agent A offers the current task to agent B as there is more value to the team if agent B has the task.*
- *Agent A is offered the task from agent C (3,4,8,3), which has a value to agent A of 75.456 (35.68 + 4.1 + 35.678).*

4. Each agent assesses the value of each of each offer from other agents based on actual levels of knowledge. Each agent can accept one task offer, rejecting any others, keeping the task with the highest potential value to the team.

   - *Agent B rejects the offer from agent A as they may have a better current task or have a better offer from another agent.*
   - *Agent A accepts the offer from Agent C as that has a higher value than the current*

task.

- *Agent A has a surplus task so sends the current task to the controller for redistribution in the team.*
- *The current task for agent A is now 3,4,8,3 giving a value of 75.456 (35.68 + 4.1 + 35.678) to the team.*

5. The controller distributes any remaining unallocated tasks randomly to agents with no remaining task. After this optimisation of tasks all agents will have one task.
6. Each agent calculates the value of their task by taking the sum of their expertise relating to each element of the task.
    - *The value for the current task for agent A is 75.456 (35.68 + 4.1 + 35.678).*
    - *Likewise the other agents will calculate the value of their current task.*
7. Each agent increases its knowledge by increasing the value of expertise associated with each element of the task. The increase is a randomly generated value between 0.001 and 1.
    - *The expertise for agent A is increased for knowledge streams 3, 4 and 8 randomly as follows:*

Knowledge	1	2	3	4	5	6	7	8	9	10
Expertise	3.04	8.12	36.058	0.872		11.304	2.009	4.92		15.6

Table 2: New knowledge and expertise of Agent A

8. Each agent shares their expertise with other agents by sending a message to a randomly selected agent including a randomly selected piece of knowledge and associated expertise. This allows agents to update their transactive memory.
    - *Agent A sends a message to Agent D with the information – Knowledge stream 10, expertise level 15.6.*

- *Agent A receives a message from agent B with the information – Knowledge stream 2, expertise level 2.5.*

- *Agent A can now update TMS to reflect the new information relating to agent B.*

9. The results of this iteration are sent to the controller agent who calculates the team measures for analysis.

   - *Agent A sends the value of the task and the contents of knowledge, expertise and TMS to the controller agent.*

10. If pair programming pairs of agents will receive the same task. The redistribution to optimise the tasks will still take place and then agents with the same tasks will partner up. The partner agents will share information exclusively with each other during the iteration and can thus update their TMS.

6.4 Model verification

The model was developed in an Agile fashion in that a small amount of functionality was developed initially. This was tested by examining the code manually, and the model was then unit tested by stepping through the code operation examining the values of variables at every stage (Pfleeger and Atlee, 2010, Sommerville, 2004). As the model produces random numbers it is not possible to predict the values of variables, only to check that the code is manipulating the variables correctly.

All unit tests were carried out a number of times to carry out:

- Statement testing – Every statement is in the component is executed at least once in at least one test.

- Branch testing – For every decision point in the code, each branch is chosen at least once in at least one test.

- Path testing – Every discrete path in the code is tested at least once in at least one test

(Pfleeger and Atlee, 2010, Sommerville, 2004)

The model was developed by gradually adding small pieces of additional functionality and continuing to test in the above manner. Throughout the verification process the model was checked for expected or unexpected behaviour.

6.5 Detailed model definition

The following tables define the elements of the model in detail and describe the flow of the operation of each part of the model.

6.5.1 Constants

Name	**Type**	**Description**
task_size	int	Number of elements in the task
team_size	int	Number of 'player' agents
random_range	int	Range of random task numbers generated from 1 to random_range
agile	int	1 = pair programming, 0 = solo programming

6.5.2 Agent types

Name	**Description**
controller	Controlling agent that issues tasks and holds team information
player	Team member agent

6.5.3 Data types

Name		**playertask**
Description		A team id and the task they are assigned
	Elements	
Name	**Type**	**Description**
playertask_id	int	ID for agent
playertask_task	int array	Array containing task elements for agent. Array size is 'task_size'.

Name		**kande**
Description		Unit containing one knowledge element and associated expertise level
Elements		
Name	**Type**	**Description**
kande_know	int	Knowledge element
kande_expertise	float	Expertise level for knowledge element

Name		**means**
Description		Unit containing one knowledge element and associated calculated value
Elements		
Name	**Type**	**Description**
means_know	int	Knowledge element
means_expertise	float	Calculated value associated with knowledge element

Name		**playermeans**
Description		Unit containing agent id, knowledge and calculated values for one team member
Elements		
Name	**Type**	**Description**
playermeans_id	int	ID of agent
playermeans_means	means_array	Dynamic array of data type 'means'

Name		**playerknow**
Description		Unit containing agent ID and a dynamic array of knowledge and expertise for one agent
Elements		
Name	**Type**	**Description**
playerknow_id	int	ID of agent
playerknow_know	kande_array	Dynamic array of data type 'kande' (units of knowledge and associated expertise)
Name		**knowexp**
Description		Unit containing knowledge and a dynamic array of expertise from agents for that knowledge
Elements		
Name	**Type**	**Description**
knowexp_know	int	Knowledge element
knowexp_expertise	float_array	Dynamic array of expertise values

Name		**playertms**
Description		Unit containing agent ID and a dynamic array of knowledge relating to other agents
Elements		
Name	**Type**	**Description**
playertms_id	int	ID of agent
playertms_tms	knowexp_array	Dynamic array of data type 'knowexp'.

Name		**playerperf**
Description		An agent and its values for all tasks
Elements		
Name	**Type**	**Description**
playerperf_id	int	ID of agent
playerperf_val	float array	Array containing values for this agent doing each task. Array size is 'team_size'.

Name		**perf_vals_low**
Description		Array of performance values
Elements		
Name	**Type**	**Description**
vals_low	float array	Array containing performance values. Array size is ‘team_size’.

Name		**perf_vals_high**
Description		Array of performance values
Elements		
Name	**Type**	**Description**
vals_high	perf_vals_low array	Array containing perf_vals_low arrays of performance values. Array size is ‘team_size’.

6.5.4 Agents

6.5.4.1 Agent name: controller

This agent is used to issue the tasks and manage the redistribution of tasks when tasks are moved around the team. All player agents report information to the controller agent and it holds the individual and team level results.

Controller memory

Name	**Type**	**Description**
team_ids	int array	Array containing ‘player’ agent IDs. Array size is ‘team_size’.
tasks_alloc	playertask array	Agents and their allocated tasks. Array size is ‘team_size’.
player_know	playerknow array	The knowledge of each ‘player’ agent’. Array size is ‘team_size’.
tms_calcs	playermeans array	Agents and associated measures of TMS accuracy. Array size is ‘team_size’.
tms_density	playermeans array	Agents and associated measures of TMS density. Array size is ‘team_size’.

cum_team_val	float	Cumulative output value for the team
team_val	float	Output value for this iteration for the team
cum_optimum_val	float	Cumulative optimum output value for the team
optimum_val	float	Optimum output value for this iteration for the team

Controller messages sent

Name		**controller_task**
Description		Randomly generated task sent to each agent.
Name	**Type**	**Description**
task_to_agent	playertask	One target agent ID and a randomly generated task

Name		**new_task_to_agent**
Description		A team id and the task they are assigned
Name	**Type**	**Description**
new_task_to_agent	playertask	One target agent ID with the reallocated task

Controller functions

Name	**controller_generate_task**
Description	Randomly generate a task for each ‘player’ agent
Current state	start
Next state	1
Flow	
(Pair programming) ↙	↘ (Solo programming)
Randomly generate ‘team_size’/2 number of tasks of size ‘task_size’ ↓ Randomly allocate each task to two ‘player’ agents ↓ Output message	Randomly generate ‘team_size’ number of tasks of size ‘task_size’ ↓ Randomly allocate each task to a ‘player’ agent ↓ Output message

Inputs	
Message name	**From agent**
/	/
Outputs	
Message name	**To agent**
controller_task	player

Name	**settle_tasks**
Description	Reallocate spare tasks to 'player' agents without tasks
Current state	1
Next state	2
Flow	
Read all 'id_to_controller' messages and put IDs into ptask array ↓ While reading 'task_to_controller' message ↓ Match incoming task to unique ID in ptask array ↓ Output 'new_task_to_agent' message ↓ loop ends	
Inputs	
Message name	**From agent**
task_to_controller	player
id_to_controller	player
Outputs	
Message name	**To agent**
new_task_to_agent	player

Name	**optimum_perf**
Description	Read in perf messages and calculate optimum output
Current state	2
Next state	3

Flow	
While reading ‘perf_to_controller’ message ↓ Add output values for each ‘player’ doing each task to two dimensional array ↓ (loop ends) Calculate optimum total output value for iteration ↓ Add optimum total output for iteration to controller memory ↓ Calculate cumulative optimum total output value ↓ Add cumulative optimum total output to controller memory	
Inputs	
Message name	**From agent**
perf_to_controller	player
Outputs	
Message name	**To agent**
/	/

Name	**update_tms_means**
Description	Read in perf messages and calculate optimum output
Current state	3
Next state	4
Flow	
While reading ‘know_to_controller’ message ↓ Update knowledge for each ‘player’ in ‘controller’ memory ↓ (loop ends) While reading ‘tms_to_controller’ message ↓ Update tms for each ‘player’ in ‘controller’ memory ↓ (loop ends) Calculate team TMS accuracy ↓ Calculate team TMS completeness ↓ While reading ‘val_to_controller’ message ↓ Calculate team output ↓ Update team outputs in ‘controller’ memory ↓ loop ends	

Inputs	
Message name	**From agent**
know_to_controller	player
tms_to_controller	player
val_to_controller	player
Outputs	
Message name	**To agent**
/	/

6.5.4.2 Agent name: player

This agent is a member of the team carrying out the work. A ‘player’ agent may work as a pair, carries out tasks and reports knowledge, TMS and output to the controller.

Player memory

Name	**Type**	**Description**
myid	int	This agent’s ID
my_know	kande array	A dynamic array of units of knowledge and associated expertise.
tms	playerknow array	Transactive memory relating to the knowledge and expertise of other ‘player’ agents. Array size is one less than ‘team_size’.
tasks_alloc	playertask array	The task allocated to each agent. Array size is ‘team_size’.
current_task	int array	The task allocated to this agent. Array size is ‘task_size’.
my_task_val	float	This agent’s output for current task
id_given	float	ID of agent given this agent’s task
new_tasks	playertask array	Dynamic array of tasks given to this agent
taskcount	int	Number of current tasks
pair	int	ID of partner for pair programming. myid or 0 if solo programming

Player messages sent

Name		task_to_expert
Description		Offer task to better qualified 'player'.
Name	**Type**	**Description**
sent_id	int	ID of sending 'player' agent
target_id	int	ID of target 'player' agent
task	int array	Array containing task offered. Array size is 'task_size'

Name		task_to_controller
Description		Excess task sent to controller for redistribution
Name	**Type**	**Description**
task	int array	Array containing task sent back to controller. Array size is 'task_size'

Name		feedback
Description		Acceptance or rejection of task
Name	**Type**	**Description**
id	int	ID of agent receiving feedback
result	int	acceptance = 1, rejection = 0

Name		id_to_controller
Description		Send ID to controller if 'player' agent does not have a task.
Name	**Type**	**Description**
id	int	ID of 'player' needing a task

Name		**find_partner**
Description		For pair programming, find 'player' agent with the same task as this agent.
Name	**Type**	**Description**
find_p	playertask	ID and task of this 'player' agent

Name		**perf_to_controller**
Description		Send values of this agent doing all tasks to 'controller'
Name	**Type**	**Description**
perf	playerperf	This agent's ID and the values for this agent doing each task
Name		**tms_to_controller**
Description		Send contents of this agent's TMS to 'controller'
Name	**Type**	**Description**
id	int	ID of this agent
tms_know	int	Knowledge element
tms_expertise	float	Associated expertise

Name		**know_to_controller**
Description		Send knowledge and expertise of this agent to 'controller'
Name	**Type**	**Description**
id	int	ID of this agent
know_know	int	Knowledge element
know_expertise	float	Associated expertise

Name		**share_know**
Description		Knowledge shared with partner for pair programming
Name	**Type**	**Description**
target_partner	int	ID of partner
sent_partner	int	ID of this agent
partner_know	kande array	Knowledge and expertise to be shared with partner. Array size is 'task_size'

Name		**reshare_know**
Description		Knowledge reshared with partner for pair programming
Name	**Type**	**Description**
target_partner	int	ID of partner
sent_partner	int	ID of this agent
partner_know	kande array	Knowledge and expertise to be shared with partner. Array size is 'task_size'

Name		**fill_tms**
Description		Send information for updating other agents' TMS
Name	**Type**	**Description**
target_id	int	ID of another 'player' agent
sent_id	int	ID of this agent
fin_tms_info	kande	One random unit of this agent's knowledge and associated expertise

Name		**val_to_controller**
Description		Send this agent's output value to 'controller'
Name	**Type**	**Description**
val	float	Value of task output

Player functions

Name	**player_get_task**	
Description	Pick up task from 'controller'	
Current state	1	
Next state	2	
Flow		
While reading 'controller_task' message ↓ Add all tasks to 'player' memory ↓ (loop ends) Identify task for this agent and add to 'player' memory		
Inputs		
Message name	**From agent**	
controller_task	controller	
Outputs		
Message name	**To agent**	
/	/	

Name	**assess_give_task**	
Description	Assess task potential task outputs and possibly offer task to another 'player' agent	
Current state	2	
Next state	3	
Flow		
Find value of this agent doing this task ↓ Using TMS assess value of each other 'player' agent doing this agent's task ↓ If higher than this agent doing task, offer it to agent that will produce best output for this agent's task ↓ Send offer message to 'player' agent		
Inputs		
Message name	**From agent**	
/	/	
Outputs		
Message name	**To agent**	
task_to_expert	player	

Name	**optimise_tasks**	
Description	Receive tasks, decide on best and accept or reject offers	
Current state	3	
Next state	4	
Flow		
While reading 'task_to_expert' message ↓ If task is offered to this agent add task to memory ↓ (loop ends) Decide on task that will produce best output ↓ Send acceptance or rejection messages for tasks offered ↓ If this agent has a surplus of tasks send them back to 'controller' for redistribution		
Inputs		
Message name	**From agent**	
task_to_expert	player	
Outputs		
Message name	**To agent**	
task_to_controller	controller	
feedback	player	

Name	**feedback**	
Description	Receive feedback and process acceptance or rejection	
Current state	4	
Next state	5	
Flow		
While reading 'feedback' message ↓ If task was offered to another 'player' process acceptance or rejection message ↓ (loop ends) If task offered was accepted and this agent had no offers send ID to controller to be issued a surplus task		
Inputs		
Message name	**From agent**	
feedback	player	

Outputs	
Message name	**To agent**
id_to_controller	controller
feedback	player

Name	**final_get_task**
Description	Obtain surplus tasks for agents with no task
Current state	5
Next state	6
Flow	
While reading 'new_task_to_agent' message ↓ If this agent has asked controller for new task add it to memory ↓ (loop ends) If pair programming send message to find partner	
Inputs	
Message name	**From agent**
new_task_to_agent	controller
Outputs	
Message name	**To agent**
find_partner	player

Name	**do_task**
Description	Update knowledge and send share information to partner if pair programming
Current state	6
Next state	7
Flow	
Calculate output value of this agent doing task ↓ Calculate output value of this agent doing all other tasks ↓ Send message to controller with value of this agent doing all tasks ↓ Send message to controller with contents of TMS ↓ Send message to controller with contents of knowledge	

↓	
(Pair programming) ↙	↘ (Solo programming)
While reading 'find_partner' message ↓ Match partner and add partner ID to memory ↓ (loop ends) Send message to partner with this agent's knowledge and expertise for current task ↓	
↓ Update knowledge and expertise for this agent for current task elements	
Inputs	
Message name	**From agent**
find_partner	player
Outputs	
Message name	**To agent**
perf_to_controller	controller
tms_to_controller	controller
know_to_controller	controller
share_know	player

Name	**share**
Description	Send selected knowledge and expertise to team and if pair programming update knowledge with partner knowledge.
Current state	7
Next state	8
Flow	
(Pair programming) ↙	↘ (Solo programming)
While reading 'share_know' message ↓ If partner is more expert than this agent increase knowledge and expertise for this agent ↓ (loop ends) Send message to partner with new knowledge and expertise ↓ Send message to partner with this agent's knowledge	
Send message to random 'player' agent with random information relating to this agent's knowledge and expertise	

Inputs	
Message name	**From agent**
share_know	player
Outputs	
Message name	**To agent**
reshare_know	player
fill_tms	player

Name	**tms**
Description	Update TMS
Current state	8
Next state	9
Flow	
While reading 'fill_tms' message ↓ Update TMS with information relating to another 'player' agent ↓ (loop ends)	
(Pair programming) ↙	↘ (Solo programming)
While reading 'reshare_know' message ↓ Update TMS with information relating to partner ↓ (loop ends)	
↓ Send message to 'controller' with the output value of this agent's current task	
Inputs	
Message name	**From agent**
fill_tms	player
reshare_know	player
Outputs	
Message name	**To agent**
val_to_controller	controller

For a detailed stategraph please refer to Appendix 1.

To examine the model in greater detail please refer to the code in the digital resources described at Appendix 2.

6.6 Model application

The application of the model described above is used to investigate hypotheses relating to the behaviour of TMS, differentiation and performance under conditions simulating the use of two eXtreme programming techniques. The application of the models starts with a simple simulation and successively introduces greater complexity and cognitive realism with the following two simulations. It is suggested that the use of XP techniques changes the cognitive mechanisms within a team, which may contradict the established literature relating to TMS. Utilising agent-based modelling allows these questions to be explored.

The first simulation, Model Tabula Rasa, utilises agents with no initial knowledge or TMS and simulates task work in an environment with a low knowledge range of 10. This simulates routine and repetitive work, as the same task elements will occur repeatedly. The simulation compares teams using the XP techniques versus traditional Waterfall techniques, namely, pair versus solo programming, and small versus large task sizes. The aim of this model is to examine the use of XP techniques in a simple environment allowing some comparison of XP and Waterfall techniques but also to set some benchmarks for the subsequent models.

The second simulation, Model Novelty, introduces the concept of task novelty and complexity by increasing the knowledge range utilised by the model to 100, reducing the likelihood of task elements being repeated. The same conditions of no TMS and no initial knowledge were used, as was the comparison of XP versus traditional parameters. The results of this model were compared with the results of the previous model in order to test hypotheses relating to teams working in novel and complex environments.

The final simulation, Model Differentiation, retains the high knowledge range simulating software development teams working in a novel environment. This simulation introduces yet more cognitive realism by using agents that have existing knowledge and TMS with varying characteristics. The initial knowledge simulates heterogeneous and homogenous initial knowledge in the team, and the presence or absence of a complete TMS simulates team members that have prior knowledge, or not, of each other. The same comparisons of pair versus solo programming, and small versus large tasks were carried out to test how various initial conditions relating to knowledge and TMS affect the team in terms of TMS, differentiation and performance over time.

The results from these simulations will allow some preliminary insight into optimum team construction for software development teams by understanding how initial conditions, the use of XP techniques and the development of TMS and differentiation affect team performance.

Before those simulations were accomplished, an attempt to validate the model by aligning it with a previous paper (Ren et al., 2006) that describes an agent-based model simulating TMS was carried out. This is described in the following chapter.

Chapter 7: Model Validation

7.1 Introduction

This chapter describes a validation exercise where the model is run with one simple condition to compare results with the behaviour of TMS, differentiation and team output as described in the literature.

7.2 Theoretical framework

There are many challenges associated with validating agent-based models (Gore and Reynolds, 2007, Louie and Carley, 2008, Roth, 2007). As a basic requirement verification has to take place; the developer has to be sure that there are no errors in the code of the model and the model has to be confirmed as operating as designed. Arguably the most difficult, the validation of the model has to show that the design accurately models the real life process that is being modelled. These operations are addressed with an iterative process. A simple model is developed and compared with known data, the results of which feed into the ongoing development of the model design. Validation is not simply a dichotomous attribute of a model - one cannot say that a model is either validated or not validated. It is a continuous scale whereby the model improves as it develops (Bryson et al., 2007). According to Sun (2009) assessing the absolute validity of a computational cognitive model is impossible. One can only aim for empirical adequacy and only when a more accurate one replaces the current model will the validity no longer stand.

The potential benefits of a TMS for a team's performance are clear as members can anticipate rather than react to other members' behaviour. An accurate expertise recognition system and knowledge differentiation are essential characteristics of a well developed TM system and are necessary for knowing who knows what expertise, and bringing that expertise to bear. Additionally, accurately identifying expertise and having differentiated knowledge facilitates the sharing of tasks and completion of goals, reducing the workload by not having to duplicate knowledge and effort in a team (Faraj and Sproull, 2000).

There is evidence to suggest that the presence of a developed TMS improves a group's performance

(Austin, 2003, Faraj and Sproull, 2000, Lewis, 2004, Liang et al., 1995, Moreland, 1999, Ren et al., 2006). Simply, longevity in a team and thus getting to know team members better will increase the quality of the TMS. A study by Reagans et al. (2005) considered teams of health professionals in a hospital carrying out joint replacement procedures. Findings suggested that experience working together was a significant predictor of team performance and moreover, those team members have more accurate and more sophisticated knowledge of who knows what on the team than their less experienced counterparts.

Specialists emerge in a team as a result of a well-developed understanding of each other's knowledge (Wegner, 1995). As the TMS becomes more efficient the knowledge in a team will become more differentiated (Hollingshead, 2000), as tasks will be directed to those team members with the most appropriate expertise thus enhancing relevant expertise further. Levesque et al (2001) found that, while studying teams of software development students, team members roles became increasingly specialised, in other words differentiation increased over time. In turn, an increase in differentiation is believed to have a positive impact on team performance due to tasks being directed to team member with the best knowledge for completing the task (Lewis, 2004, Littlepage et al., 2008)

Austin (2003) looked at performance, differentiation and TMS accuracy in mature teams in different departments of an apparel and sporting goods company. Results showed that accuracy and differentiation have an influence on performance with the interaction between differentiation and accuracy having a significant effect on performance. This indicates that if the team members know accurately what expertise is in the team then tasks are efficiently directed to the most expert team member, having a positive impact on both performance and differentiation.

Over time a TMS in a team will develop with the optimal state of a TMS being convergence. This is a state where all members of a team have the same, complete, accurate representations of the actual knowledge in the team (Brandon and Hollingshead, 2004). For TMS structures to develop Hollingshead (2001) found that individuals required cognitive interdependence, be aware of the expertise of others in the team and have convergence. In reality, it is unlikely that any team will achieve perfect convergence, however, this may not be necessary for effective and improving performance of a team. Additionally, the TMS will always be adapting as the knowledge in a team is constantly changing and this will always have some time lag, impacting on performance. It is also possible that the TMS in a team reaches a 'good enough' state of convergence and unless there is some team upheaval it will remain at a steady level. In summary it would seem that in general terms developing an accurate TMS and increasing differentiation contribute to better outcomes for a team over time.

This model aims to simulate a single team process in order to establish that the behaviour of the model mirrors the behaviour of a TMS as described in previous research.

7.3 Method

The model used for this experiment was as described in previous chapters. There was one simple condition for this simulation as described in table 3 below.

Number of agents	4
Task size	20
Work mode	Solo
Knowledge range	1 to 10
Iterations	200

Table 3: Validation: Parameters for model

Team size is one factor that has been shown to have an effect on TMS (Palazzolo et al., 2006) and smaller groups have been found to gain greater benefit from TMS than larger groups (Ren et al., 2006). There are numerous variables that can be modelled and decisions have to be taken to choose a manageable set or, as previously discussed, the data can become unwieldy and difficult to analyse. For this work it was decided to use a constant team size of four throughout and model other factors that are of interest. Four was chosen as being an even number it lends itself to pair programming and being a small team the data can be easily analysed. Larger teams may be modelled in later work.

An arbitrary task size of twenty was chosen to model a fairly detailed, normal task. The range of knowledge was between 1 and 10 meaning that each task contained random numbers between 1 and 10 and each agent could develop a maximum of ten knowledge streams. This was chosen as a number that would provide enough diversity of knowledge but not so much that the analysis of the data would be unwieldy.

At the start of the simulation all agents had no knowledge and no associated transactive memory. The simulation was carried out 50 times with each simulation running for 200 iterations. A number of trials were carried out with varying numbers of iterations and 400 was judged to provide enough detail; and with greater numbers of iterations the model did not provide any further useful information.

7.4 Results

The results show the behaviour of the model over time in terms of mean output ratio, mean TMS accuracy and mean differentiation. Error bars show standard error of the mean (SEM) where:

$$SEM = \sigma / \sqrt{n}$$

where σ is standard deviation and n is sample size.

Figure 5 shows a steady increase of the output ratio over time indicating that the tasks are being directed increasingly efficiently to the agent with the most appropriate knowledge resulting in a higher mean ratio between actual output and optimum output over successive iterations. The increase in output also indicates that agents are gaining specialism, so not only are tasks being directed to the best agent, but also the agents are becoming greater experts and thus delivering greater task output.

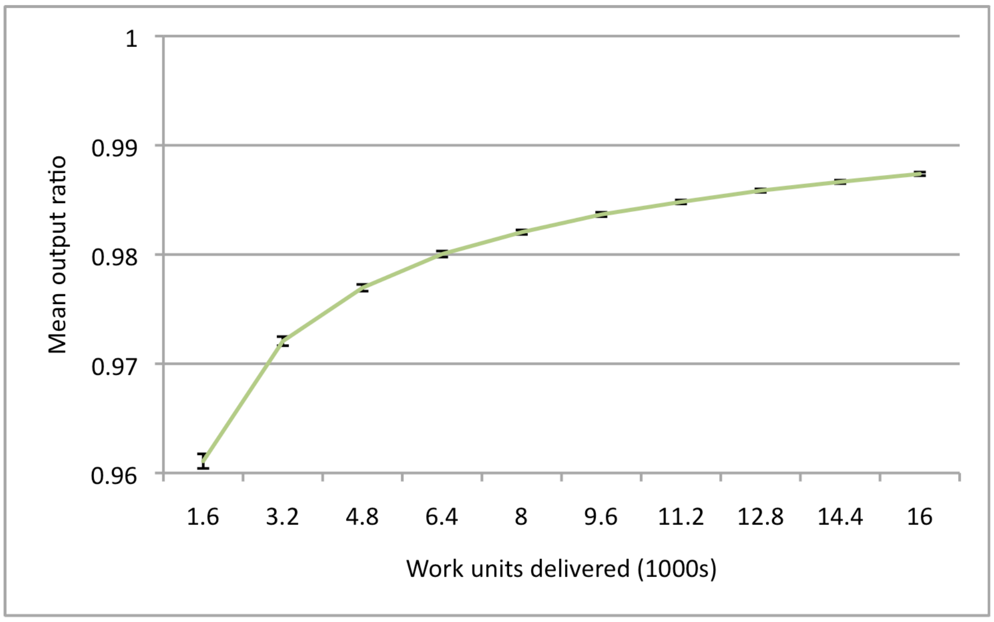


Figure 5: Validation: Mean output ratio over time

Figure 6 shows the mean TMS accuracy. The graph shows that TMS accuracy increases over subsequent iterations. This indicates that agents are learning about the knowledge of other agents over subsequent iterations. This is allowing tasks to be directed more efficiently to the agent with the best knowledge for the task.

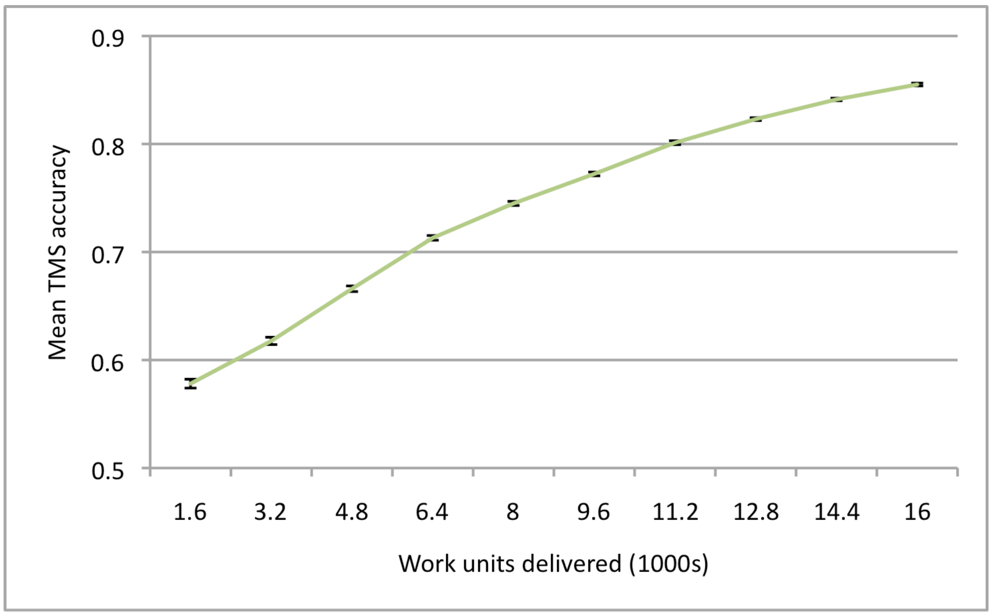


Figure 6: Validation: Mean TMS accuracy over time

Figure 7 shows the differentiation over time and shows that differentiation increases over time. This demonstrates that specialisation is taking place in the model with agents developing greater knowledge in fewer areas. The TMS is causing tasks to be directed more often to the agent with the most knowledge in that area thus increasing specialism further.

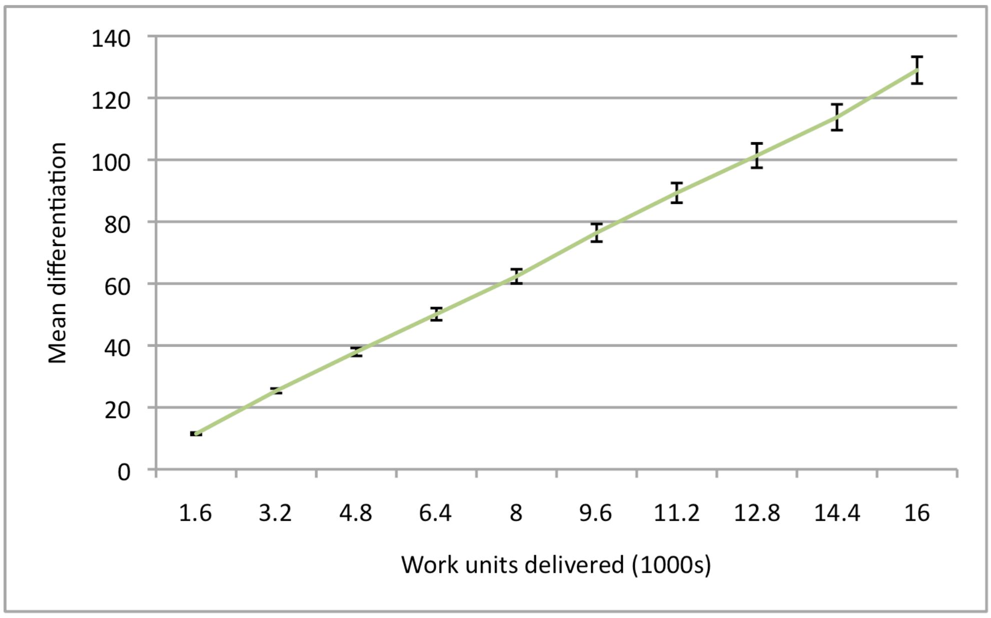


Figure 7: Validation: Mean differentiation over time

The results show that as expected, output, differentiation and TMS accuracy increase over time. The

TMS is serving the team by directing tasks in a more efficient manner to the agent that will produce the best task output for the team, thus in turn increasing differentiation.

7.5 Discussion

The aim of this simulation was to establish that the model simulates the behaviour of a TMS in order that further simulations can be carried out to investigate the research questions relating to the comparison of Agile and Waterfall methodologies.

The model simulated a team of four carrying out a series of tasks with each team member having no initial allocated knowledge and no associated transactive memory of other team members. The model produced increasing mean output of the team over time, increasing mean differentiation, and increasing TMS accuracy.

This supports the literature positing that accurately identifying expertise and having differentiated knowledge facilitates the sharing of tasks and completion of goals, reducing the workload by not having to duplicate knowledge and effort in a team (Faraj and Sproull, 2000). The model showed increasing output over time supporting Reagans (2005) who studied established teams of health professionals.

It is known that specialists emerge in a team as a result of a well-developed understanding of each other's knowledge (Wegner, 1995) as tasks are directed to those team members with the most appropriate expertise thus enhancing relevant expertise further. This simulation showed differentiation developing over time, which in turn positively influenced the output of the team being modelled. This supports the literature on differentiation (Austin, 2003, Levesque et al., 2001, Lewis, 2004, Littlepage et al., 2008) indicating that if the team members know accurately what expertise is in the team then tasks are efficiently directed to the most expert team member, having a positive impact on both performance and differentiation.

Over time a TMS in a team will develop with the optimal state of a TMS being convergence. This is a state where all members of a team have the same, complete, accurate representations of the actual knowledge in the team (Brandon and Hollingshead, 2004). The TMS accuracy in this validation model was found to increase with the curve flattening out slightly over time indicating that the speed to complete convergence is slowing down. This would support the idea that perfect convergence is unlikely and that a 'good enough' state of convergence may ensue.

7.6 Conclusion

In summary, supporting the literature, this simple model simulates a simple team over time and demonstrates that developing an accurate TMS and increasing differentiation contribute to better outcomes for a team.

Chapter 8: Aligning the model

8.1 Introduction

This chapter describes a further exercise in validation carried out on the model prior to running the simulations relating to this research. Alignment can be used to determine equivalence of models where the results of two models given the same simulation requirements can produce comparable results.

8.2 Theoretical framework

Alignment, or validation, of a model is the process used to compare a model with another model to determine if they produce the same results under the same tests. Determining equivalence of models can be a useful tool for validating models (Axtell et al., 1996b) or providing insight into the reasons for any differences, however the question of equivalence can be fraught with difficulty. Determining equivalence of two models is not to prove that two models are identical but whether, when claiming to model the same phenomena, they produce the same results.

Axtell (1996a) carried out an alignment exercise by comparing two models that had been used for quite different aims, a simple model and a model that was much more complex. To do this the more complex model was asked to process a very simple scenario that was akin to one that the simple model could simulate. In this way they could compare the two models to establish if the models were equivalent in that that they produced the same results. Of course only the simple model would be validated as only a small element of the complex model was being tested.

The alignment exercise produced mixed results, however, the results highlighted a number of issues with alignment of different models. Sensitivity of parameters can make a difference in the success of alignment; meaning that small changes in input parameter values can make large differences in the success or failure of alignment of the models. Secondly, the study identified that the problem of specifying 'equivalent' model behaviour is not trivial. Do the models produce distributions of results that cannot be distinguished statistically? Can the models be shown to produce the same internal relationship between their results? The study concluded that alignment is a useful methodology for validation but care must be taken over interpretation of the results and awareness of the sensitivity of the models must be taken into account.

Alignment with a prior model of transactive memory was chosen to carry out some initial validation of this model, however, only two previous models of transactive memory systems have been found Palazzolo et al. (2006) and Ren et al. (2006).

Palazzolo et al. (2006) used Blanche to look into how different initial conditions influenced the development of the TMS. The model simulated initial knowledge, initial accuracy of TMS and network size, and how these parameters affected final accuracy of TMS and differentiation. Interestingly this model resulted in only a modest fit between the theoretical model and the computational model results, so they explored alternative explanations and modified their theoretical model based on the results. This is a demonstration of the cyclic nature of agent-based modelling where the model generates further questions that may assist with the adaptation of the theory. This can then in its turn influence the refinement of the agent-based model.

Another example of an agent-based model of transactive memory systems was developed by Ren et al. (2006) who validated it by comparing it with a previous experiment into transactive memory (Liang et al., 1995). The original study (Liang et al., 1995) tested hypotheses that training a group together will have a positive affect on transactive memory and in turn the performance of the group. The measures of differentiation, task coordination and task credibility were used to infer the strength of the transactive memory system. Measures of performance included recall of information, assembly accuracy and time taken. The results suggested that groups that are trained together have greater recall and assembly accuracy but that there was no effect on time taken to complete the task. Also, the results indicated that group training has a positive effect on transactive memory, and that transactive memory mediates the effects of group training on performance.

Ren et al., (2006) used ORGMEM to create a model to test the contingent effects of transactive memory on group performance and used the Liang et al. study to validate it. The independent factors used were group size, task volatility and knowledge volatility. The dependent factors were TMS accuracy and density, and performance measures of time and quality. Quality was assessed as a combination of the match between individual requirements and resources, and a group assessment of how knowledgeable each agent was. The agents were simulated through a training stage, starting with no initial knowledge, for two conditions - training individually and training within the team. Each team was then put through the testing stage under each of the training conditions. The results corresponded to the previous study and the researchers judged their model effectively validated.

It was decided to use the Ren et al. study for aligning models as they had used empirical data from a previous study to validate their model unlike the Palazzolo et al. model, which was purely theoretical.

The model for this research was developed to explore the relationship between TMS activity and team performance with regard to various input factors, so departing from the research focus of the comparison of software development methodologies, this model was utilised in a training scenario, in the same manner as Ren et al. They simulated the training scenario in order to demonstrate some equivalence with the original Liang et al. (1995) paper.

8.3 Method

The independent factor used for Ren et al.'s validation exercise was training condition, where members of groups were trained as individuals or as a team. The dependent factors were TMS accuracy and density, differentiation and performance, where time and quality was measured. Quality was assessed as a combination of the match between individual requirements and resources, and a group assessment of how knowledgeable each agent was.

In order to align the model with the Ren et al. model a number of model parameters were defined to emulate the Liang et al. and Ren et al. papers but as the modelling environments had some differences assumptions were made to produce the best reproduction of both experiments. So in that spirit, task size was defined as five, as with the Ren et al. model and the range of knowledge was 1 to 10 meaning that each task contained five random numbers between 1 and 10, and each agent could develop a maximum of 10 knowledge streams. As with the Ren et al. model there were 50 three-agent teams for each condition, the team training condition and the individual training condition. Again mirroring the Ren et al. model, the training phase was run for 50 iterations and the testing phase for 100 iterations. At the start of the training phase all agents had no knowledge and no TMS. For the testing phase the teams who had trained together retained their knowledge and their TMS, but the teams where the agents had been trained individually retained their knowledge and their TMS was empty.

As with Ren et al. the three hypotheses were adapted from the Liang et al. paper as follows:

1. Groups whose members are trained together rather than alone will perform better than groups whose members are trained alone.

2. Groups whose members are trained together will develop higher quality TMSs than groups whose members are trained alone.

3. The development of TMSs will mediate the effects of group training on team output.

The measures of team output, differentiation and TMS accuracy and density were taken at ten intervals over the 100 iterations.

The two conditions were compared by reviewing graphs of the behaviour of the model over time and by repeated-measures ANOVAs. The mediation was measured using the methodology as defined by Baron and Kenny (1986) as used by both Liang et al. and Ren et al.

8.4 Results

The following graphs show the behaviour of the models over time under the different conditions. All results show the means for all 50 runs of the simulation. Error bars show standard error of the mean (SEM) where:

$$SEM = \sigma / \sqrt{n}$$

where σ is standard deviation and n is sample size.

The following graph (Figure 8) shows the mean for output ratio over the course of the 100 iterations of the testing phase of the simulation. It can be seen that output ratio for those groups that were individually trained is substantially reduced at the start of the simulation however output ratio increases rapidly until it almost converges with that for groups trained together. Output ratio for groups trained together also shows a steady increase throughout the simulation.

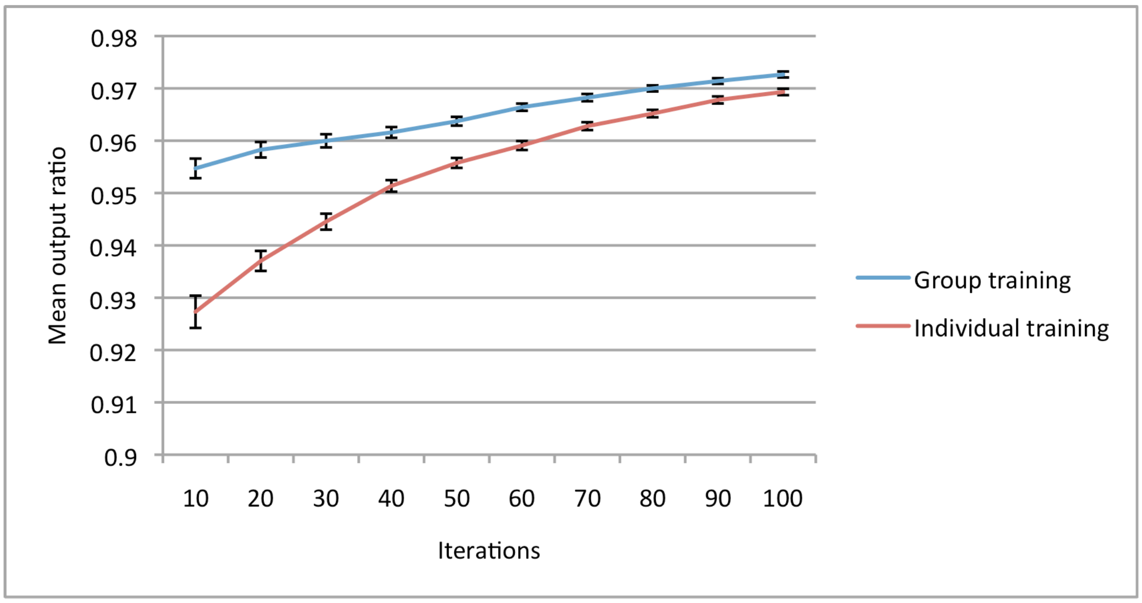


Figure 8: Alignment: Mean output ratio over time

The following graph (Figure 9) shows the mean for TMS accuracy over the course of the 100 iterations of the testing phase of the simulation. The elevated initial accuracy for individually trained groups is likely to be due to the TMS starting empty and the initial contents of the TMS being more accurate than a more mature, populated TMS; as it becomes more populated the accuracy drops then as the volume of content of the TMS approaches the group training condition the accuracy for both conditions becomes very similar.

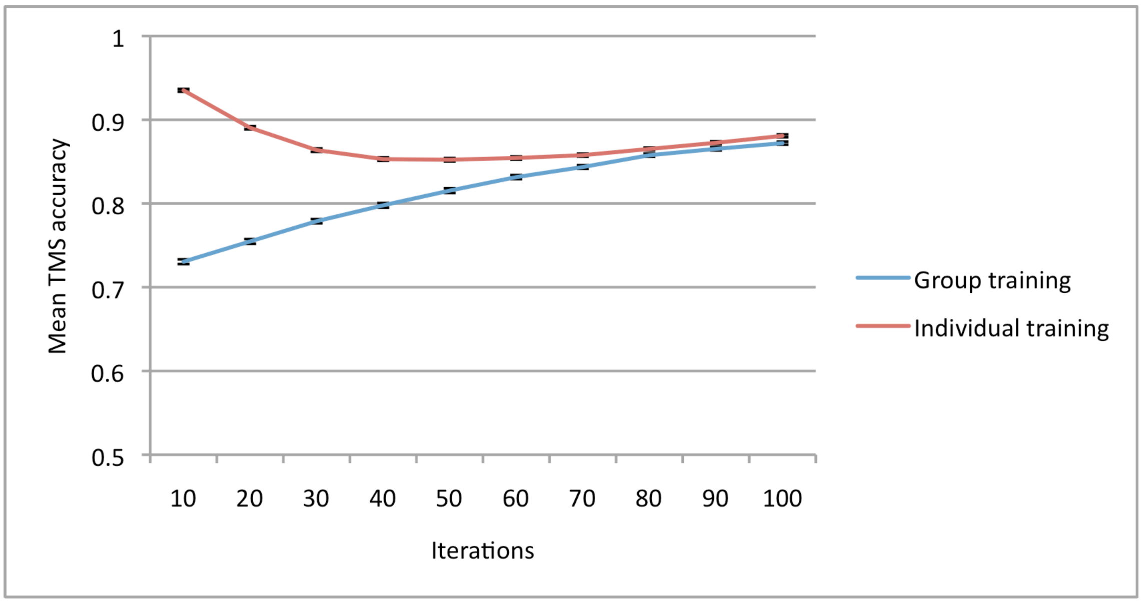


Figure 9: Alignment: Mean TMS accuracy over time

Figure 10 shows that the groups that are trained together start the simulation with a TMS that is almost complete in volume although not necessarily accurate (Figure 6, TMS accuracy). For those groups that are trained individually the TMS increasingly becomes populated to join the volume of TMS of the trained together condition.

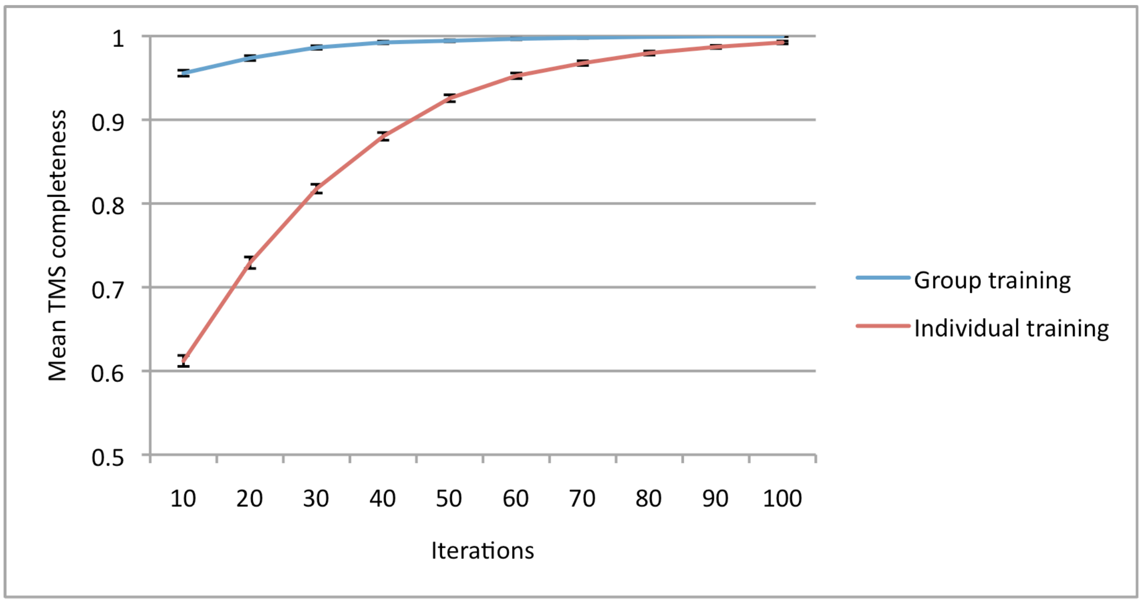


Figure 10. Alignment: Mean TMS completeness over time

In addition to the parameters as measured by the Ren et al. model the measures for differentiation for this model have been recorded in line with the Liang et al study. Differentiation and TMS have a close relationship and this measure may help to explain the results. The following graph, Figure 11, Mean differentiation over time shows the differentiation over the course of the simulation. The rate of increase of differentiation increases over the course of the simulation with differentiation for groups that were trained together greater than the groups trained individually.

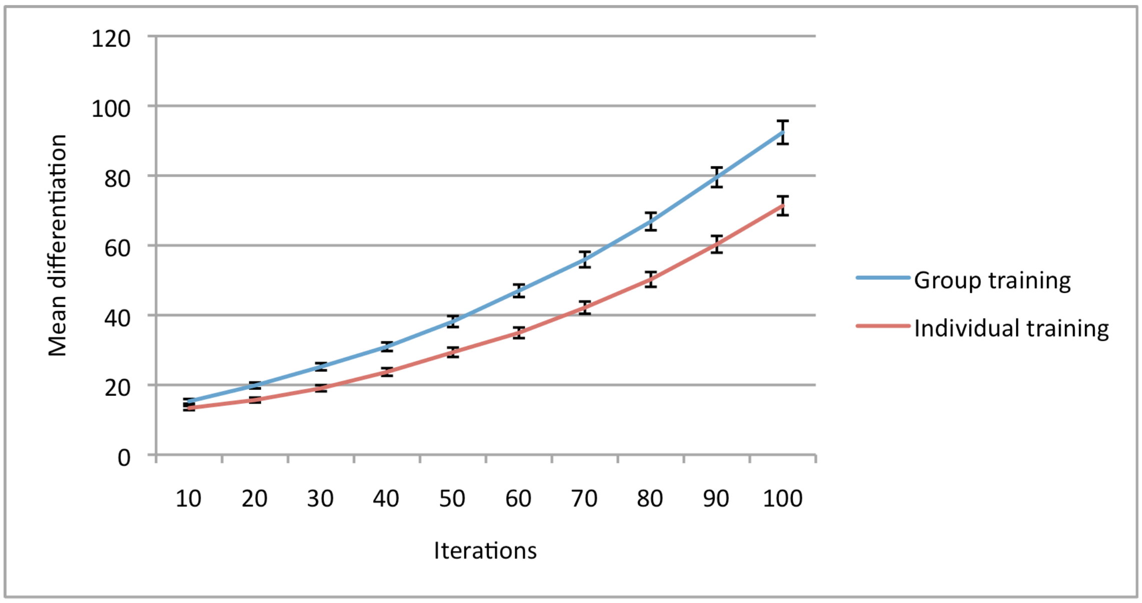


Figure 11: Alignment: Mean differentiation over time

Ren et al. carried out t-tests to investigate whether the results were drawn from significantly different populations, however, this takes no account for the behaviour of the model over time, it only analyses a snapshot of the model at one point. Although it is not clear from their paper it is assumed that the results at the end of the simulation are used. In order to look at the behaviour of this model over time in addition to comparing conditions, repeated measures ANOVA was used in preference.

Where a t-test is used to compare the means of two sets of data to test whether statistically they originate from the same or different populations, the ANOVA (or ANalysis Of VAriance) test provides a statistical test of whether or not the means of several groups are all equal, and therefore generalizes the t-test to more than two groups. A repeated measures ANOVA is used when all members of a sample are measured under a number of different conditions. As the sample is exposed to each condition in turn, the measurement of the dependent variable is repeated. Repeated measures ANOVA tests are typically used in longitudinal experiments where the repeated measure is time.

Sphericity is an important assumption of a repeated measures ANOVA. It relates to the condition where the variances of the differences between all possible pairs of groups are equal. The violation of sphericity occurs when the variances of the differences between all combinations of the groups are not equal. If sphericity is violated, then the variance calculations may be distorted, which would result in an F-ratio (used to determine whether the variances in two independent samples are equal) that would be inflated. Mauchly's test tests the hypothesis that the variances of the differences between conditions are equal and if the results of Mauchly's test are significant then sphericity has been violated and the F‑ratios produced by the statistical software, in this case SPSS, cannot be trusted. Fortunately, if data violate the sphericity assumption there are several corrections that can be applied and all of these involve adjusting the degrees of freedom associated with the F‑value. In all cases the degrees of freedom are reduced based on an estimate of how 'spherical' the data are; by reducing the degrees of freedom we make the F‑ratio more conservative. The Greenhouse‑Geisser correction is the correction to use when $\varepsilon < 0.75$ (as stated on the SPSS output), and when $\varepsilon > .75$ then the Huynh‑Feldt correction is to be used. For further explanation see Field (2005).

Hypothesis 1 states that groups whose members are trained together will perform better than groups whose members are trained individually.

In order to analyse the operation of the model in more detail over time a repeated-measures ANOVA was carried out on output ratio over the five iteration periods. Mauchly's test indicated that the assumption of sphericity had been violated ($X^2(9) = 345.85$); therefore degrees of freedom were corrected using Greenhouse-Geisser estimates of sphericity (= .39).

The results show that over time overall output increased significantly, F(1.58, 154.43) = 325.17, p< .001. For the effect of training condition there was a significant overall effect indicating that output differed by training condition, F(1,98) = 66.43, p < .001. In terms of interaction effects there was significant interaction between training condition and subsequent iterations indicating that over subsequent iterations output was affected by training condition F(1.58, 154.43) = 47.87, p < .001. These results can clearly be seen on the graph shown above at Figure 5. Hypothesis 1 is supported; members who are trained together will produce greater output than groups whose members are trained individually. In addition, output generally increases over time, and output differed by condition over time.

Hypothesis 2 states that groups whose members are trained together will develop a higher quality TMS than groups whose members are trained alone. As differentiation and TMS accuracy are both considered to be indications of strength of TMS repeated measures ANOVAs were carried out for both variables.

Taking TMS accuracy initially, Mauchly's test indicated that the assumption of sphericity had been violated ($X^2(9) = 46.63$); therefore degrees of freedom were corrected using Huynh‑Feldt estimates of sphericity (= .83). The results show that over subsequent iterations there is a significant increase in TMS accuracy, F(3.48, 341.13) = 267.72, p < .001. For the effect of training condition there was a significant overall effect indicating that there was a difference in TMS accuracy between training conditions, F(1, 98) = 768.93, p < .001. Lastly, interaction effects show that there was also a significant interaction between training condition and subsequent iterations indicating that over subsequent iterations TMS accuracy was affected by training condition F(3.48, 341.13) = 360.34, p < .001. Hypothesis 2 is partially unsupported; these results show that groups that are trained individually have a more accurate TMS than groups that are trained together, however, reviewing figure 7, this could be due to the completeness of the TMS. Groups that are trained together have a more complete TMS at the start of the simulation, albeit perhaps less accurate, compared to a less complete, more accurate TMS for groups that are trained individually. The graphs for accuracy and completeness show that over time, completeness and accuracy converge and accuracy continues to increase.

Secondly, addressing differentiation, Mauchly's test indicated that the assumption of sphericity had been violated ($X^2(9) = 513.49$); therefore degrees of freedom were corrected using Greenhouse-Geisser estimates of sphericity (= .30).

The results show that over subsequent iterations overall differentiation increased significantly, F(1.21, 118.75) = 1135.84, p < .001. For the effect of training condition there was a significant overall effect indicating that training conditions affected differentiation, F(1,98) = 25.54, p < .001 with groups being trained together having greater differentiation. In terms of interaction effects there was significant interaction between training condition and subsequent iterations indicating that over subsequent iterations differentiation was affected by training condition F(1.21, 118.75) = 20.26, p < .001. Again, this can clearly be seen on the graph at Figure 8 showing a significant increase in differentiation over time and by condition. Assuming that increased differentiation is a measure of a strong TMS then by measures of differentiation hypothesis 2 is partially supported.

Hypothesis 3 states that the development of a TMS will mediate the effects of group training on team output. Following the procedure of Baron and Kenny (1986), as did both Liang et al. and Ren et al. the mediation effect of both TMS accuracy and differentiation were tested. No mediation effects were found.

In summary the results are mixed. Hypothesis 1 was supported, hypothesis 2 partially supported and hypothesis 3 was unsupported. Training condition did have an effect on output, differentiation and TMS accuracy. In addition the picture over time is of note. Output increases generally over time and this is affected by training condition, with output over time greater for groups that are trained together but for groups trained apart output rises faster to converge with group trained together.

The results do appear plausible. It would seem sensible that groups with knowledge of each others' skills will initially have higher output than groups that have no knowledge of each other, and that after some time the output of the groups trained individually will 'catch up' due to the development of a TMS and differentiation.

8.5 Discussion

The results of this model show some agreement with both the Liang et al. and Ren et al. work. The independent variable of training condition had a significant effect on output. There was also significance for differentiation and TMS accuracy for training condition, over time and a significant interaction effect was found.

The model does show significance in differentiation, which increased over time for both groups with it increasing faster over time for the groups that were trained together. This result indicates that the model is successfully simulating a TMS in the team over time by, as they learn more about other agents' expertise, directing tasks to agents that are most expert in that subject area thus increasing that agent's expertise further. This has the effect of increasing TMS and differentiation.

It is asserted that there are some inconsistencies in the translation of the Liang et al. study to the Ren et al. one and also to this one. One of the problems with modelling the Liang et al. study is the translation of factors into the modelling environment and the interpretation of those factors. For example, Liang et al. measure TMS in terms of differentiation; task coordination and task credibility while Ren et al. measure TMS in terms of TM density and TM accuracy.

Likewise, the measures for team performance differ in all three cases; Liang et al. measure accuracy of memory and errors made during assembly, Ren et al. measure group performance as a combination of the match between individual requirements and resources, and a group assessment of how knowledgeable the agent is, and finally this model measures output by comparing the optimum team output based on knowledge and expertise against the actual team output. Ren et al. measure performance partially by a group assessment of how knowledgeable an agent is and it is argued here that this is a function of a transactive memory system, and thus blurring the distinction between TMS and performance.

There have been numerous studies of TMS, many differing methods for measuring it, and some discussion around its constitution (Akgun et al., 2005, Austin, 2003, Brandon and Hollingshead, 2004, Brauner and Becker, 2006, Faraj and Sproull, 2000, Griffith and Neale, 2001, Gupta and Hollingshead, 2010, Hollingshead, 2000, Hollingshead, 2001, Jackson and Moreland, 2009, Lewis, 2003, Prichard and Ashleigh, 2007, Zhang et al., 2007). Some of the factors used to measure TMS include differentiation, coordination, knowledge stock, TMS consensus, TMS accuracy, credibility and communication. The developers of a model of transactive memory necessarily have to select and interpret measures of transactive memory that can be translated into the logical world of a modelling environment.

These problems demonstrate the issues relating to translating a verbal theory into a model that is accurate. The definitions, assumptions, processes and parameters all have to be written in English alongside the code. Due to the nature of the English language this version will be a woollier and less precise version of the model code, and is also potentially open to inconsistent interpretation by the reader due to ambiguity. In contrast the model version of the theory necessarily has to be defined with rigour and cannot include any ambiguity. It is also possible that due to the model having to be defined in every detail during the development process, omissions or inconsistencies in the English language version may be discovered and assumptions made (Sun, 2009).

The modelling environment is logical, numerical and absolute with no room for vague or imprecise parameters. The real world however, is not, and this can be especially true for the behaviour of people who can be vague, imprecise and inconsistent. This presents problems for the modeller who by necessity has to make assumptions relating to value or process. In illustration of this point the Liang et al. paper tested task credibility, which was a measure of the level of trust each agent had for each other's knowledge and measurement of this variable was carried out by third party judges observing behaviour and making an assessment. Ren et al. chose to measure TM in terms of TM density and accuracy. The mapping of Liang et al.'s measure onto Ren et al.'s measure is difficult to evaluate and likewise onto the measures for this model of TM in terms of completeness, accuracy and differentiation.

In addition to showing some of the issues with the translation of theory into a model, this exercise has also highlighted some of the issues with comparing different modelling environments. Different modelling environments vary in their detail and construction with different treatment of variables, assumptions, and processes. In a model, the interaction of input and processes produces emergent behaviour. In different modelling environments the input variables and the processes will be different in some respects, otherwise the models would be identical. It is the mapping of these variables and processes between modelling environments that can make it difficult to compare models.

The Ren et al. model appears to be a more complex model than ours. This should not be a drawback as simple and complex models have been successfully aligned before (Axtell et al., 1996a). This model and the model described in the Ren et al. paper were clearly designed for different purposes and both were utilised in a simple manner in order to carry out this alignment. The comparison of the output and any emergent behaviour will allow some comparison but any differences could be as a result of differences in the design of each model or the mapping of factors between them. It is possible that the equivalence of some variables is problematical due to semantic or conceptual reasons.

In addition small changes to agents can have very different effects at the macro level and this is difficult to identify in more complicated models (Boero et al., 2008). This sensitivity of the model is important when making assumptions. If the parameters or values change will the model still reliably predict known behaviour and how much do they have to change to render the model invalid? One of the strengths of agent-based modelling is the variability of the combinations of parameters and variables, each variation potentially delivering different emergent behaviour. It is these nuances in behaviour as a result of small changes in input that establish agent-based modelling as a powerful tool. So, simple models, which address the question, are preferred (Burton and Obel, 1995). An additional factor in agent-based modelling that is often absent in traditional empirical experimentation is the continuous observation of model behaviour over time. The Ren et al. model did not evaluate any results over time; had they done so, they may have garnered different results.

In terms of transactive memory research this exercise has produced some initial results relating the behaviour of agents under particular initial conditions. It has introduced some further questions for subsequent simulations and development of the model. As stated earlier the primary focus for this model is on how TMS and performance is affected in Agile environments and this exercise has provided some initial data with which to progress the research focus.

Regarding the validation exercise, it is dangerous to make assumptions of validation based on such a simple test such as this. Variation of variables can make large changes to the behaviour of a model and the factor of time has to be taken in to account, which Ren et al. did not do. However, as discussed earlier, the translation of verbal theory and subjective, vague, potentially ambiguous concepts into a logical, precise model is open to error, and great care must be taken to ensure validity at this low level. For this exercise the mapping of the variables used in the Liang et al. study onto the Ren et al. model and also onto this model was questionable.

8.6 Conclusion

In conclusion, the validation carried out by Ren et al. was simplistic, using only one previous study and with no testing for sensitivity of changes in values or parameters. In hindsight, there was not enough detailed information relating to the Ren et al. model in order to carry out an effective alignment. It was also perhaps naive to expect this exercise to succinctly validate this model, but it has highlighted some questions and provided a test of the model resulting in some interesting output.

This exercise has highlighted that validation has to be more than just a few tests for statistical significance. There has to be an iterative process where the model becomes more plausible as it develops. The relationship between model and empirical or academic data is not one-way. Research will validate the model but data from the model may open up new lines of research, which will in turn feed back into the model design. The relationship is a circular one resulting in ongoing exploration of the domain of research, with successful validation allowing this cycle to continue.

Chapter 9: Model Tabula Rasa

9.1 Introduction

Previous chapters have discussed team and knowledge processes, Transactive Memory Systems (TMS), eXtreme programming (XP), and agent-based modelling (ABM). This chapter describes the first simulation, Model Tabula Rasa, which utilises agents with no initial knowledge or TMS and simulates task work in an environment with a low knowledge range of 1 to 10. This simulates routine and repetitive work, as the same task elements will occur repeatedly.

The aim of this simulation is to examine the use of XP techniques in a simple environment allowing some comparison of XP and Waterfall techniques but also to obtain some benchmarks for subsequent simulations.

9.2 Theoretical framework

Specialists emerge in a team as a result of a well-developed understanding of each other's knowledge (Wegner, 1995). As the TMS becomes more efficient the knowledge in a team will become more differentiated (Hollingshead, 2000), as tasks will be directed to those team members with the most appropriate expertise thus enhancing relevant expertise further.

There is likely to be an optimum level of differentiation for a particular set of circumstances and conditions. Very low or non-existent levels of differentiation produce dysfunctional teams, as there will be no requirement for interaction if all members have the same knowledge (Klimoski and Mohammed, 1994). Less differentiated knowledge will inhibit the need for team members to work together thus rendering the TMS less useful (Brandon and Hollingshead, 2004). One view suggested by the literature states that performance advantages deriving from TMS are greater when group members have higher differentiation (Austin, 2003, Larson, 2007, Lewis, 2004, Littlepage et al., 2008). Austin suggests that differentiation means that it will be easier to recognise experts and as such the team members are more likely to agree on the distribution of expertise in the team. In support of this Lewis found that teams with initially diverse knowledge were better able to develop TMS than teams with overlapping knowledge and that a stronger TMS was positively related to team performance.

So, it is essential to have some differentiation in a team, but at the other extreme of very high differentiation there is an essential requirement for some overlap in team knowledge to allow effective communication, coordination and shared expectations (Cooke et al., 2000).

It has been proposed that tasks divided into smaller chunks and higher differentiation of team knowledge result in higher team performance (Littlepage et al., 2008). The Littlepage et al. study, however, was carried out on staff doing relatively routine work and the findings may not apply in an innovative environment such as software development where tasks are often more novel, complex and non-routine (Brandon and Hollingshead, 2004). There is an alternative and contradictory view that there is a positive relationship between task size and performance when looking at familiar tasks in software development (Espinosa et al., 2007a). Support for this is scarce, however, endorsing the view of Littlepage et al.

XP in software development has become more widespread and it may be useful to understand how its elements affect team processes and outcomes. The importance of TMS is well recognised as positively influencing the performance of software development teams by ensuring that expertise is effectively utilised (Akgun et al., 2006, Faraj and Sproull, 2000). As discussed in a previous chapter there is also much empirical work on the benefits of XP (Hannay et al., 2009, Macias, 2004, Nosek, 1998, Williams et al., 2000), however, there is a scarcity of work looking at how different development methodologies facilitate expertise coordination and knowledge processes. This was addressed in a study by Maruping et al. (2009) who examined the relationship between eXtreme programming practices, expertise coordination and software quality in software project teams. This study found that collective ownership of code attenuates the relationship between expertise coordination and software product quality. This study focussed on collective ownership of code and coding standards but their theoretical model indicates that studies of other elements of eXtreme programming may also yield similar results. It could be argued that pair programming and collective ownership of code will have similar effects on TMS.

Hypothesis 1a. The eXtreme programming practice of pair programming moderates the relationship between TMS accuracy and output. TMS accuracy will be lower for pair programming conditions, as by working together closely and having overlapping knowledge of the project, as suggested by Maruping (2009), there is reduced interdependency in the team and thus less reliance on the TMS.

TMS is especially important when teams perform complex and unpredictable tasks and it can be shown that when teams face complexity they resort to team familiarity (TMS accuracy) to reduce the complexity of the task (Espinosa et al., 2007a). Reduced task size is one element of eXtreme programming and it is proposed that large task size is one characteristic that can render the task complex, demanding higher reliance on TMS for larger tasks.

Hypothesis 1b. The eXtreme programming practice of small task size moderates the relationship between TMS accuracy and output. TMS accuracy will be lower for small task conditions.

As found by Maruping et al., knowing the location of expertise in a team becomes less important when there is collective ownership of code, which can substitute for expertise coordination mechanisms such as TMS (Maruping et al., 2009). Whether this argument could be extended to pair programming is an interesting question. If team members are working in pairs, knowledge is more likely to be distributed evenly around the team resulting in lower differentiation.

With pair programming, the risk from losing key programmers is reduced as long as the pairs are changed frequently, because there are multiple people familiar with each part of the system. Also, the pair approach is better at leveraging expertise by pairing experts with less skilled partners (Williams et al., 2000). Again this would aid the distribution of knowledge around the team reducing differentiation.

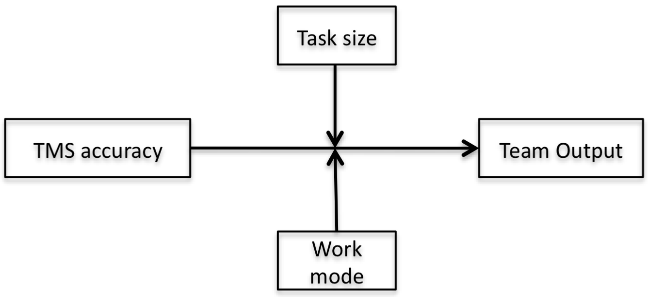


Figure 12: Tabula Rasa: Diagram of moderation relationships hypotheses 1a and 1b

Hypothesis 2a. The eXtreme programming practice of pair programming moderates the relationship between differentiation and output. Differentiation will be lower for pair programming conditions.

As mentioned earlier it is proposed that tasks divided into smaller chunks and higher differentiation of team knowledge result in higher team performance (Littlepage et al., 2008). This does not fit well with the previous hypothesis 2.a. as the two eXtreme programming practices would appear to conflict with each other. However, it is proposed to test it with the model.

Hypothesis 2b. The eXtreme programming practice of small task-size moderates the relationship between differentiation and output. Differentiation will be lower for small task conditions.

Overall software quality is affected far more profoundly by improvements to developer knowledge, which reduces future defect creation, than by simply removing defects from current individual projects. By increasing the overall knowledge stock and reducing the differentiation of knowledge in the team pair programming can contribute strongly to this optimum defect removal efficiency (Williams et al., 2000).

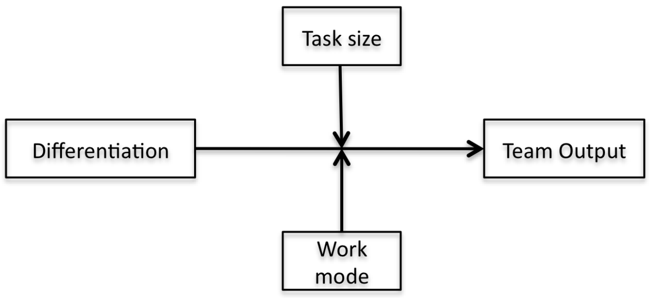


Figure 13: Tabula Rasa: Diagram of moderation relationships hypotheses 2a and 2b

There is much research looking at the benefits of pair programming and to summarise some of it Hannay et al. (2009) carried out a meta-analysis on the effectiveness of pair programming finding a small positive overall effect of pair programming on software quality. It is not unreasonable to assume, however, that there are more complex relationships influencing the outcomes of pair programming projects. With pair programming, programmers with low expertise benefit with increased software quality, intermediates from decreased time spent and the experts generally have no overall benefits at all (Arisholm et al., 2007). Theory predicts that experts perform better on complex tasks and that novices perform better on non-complex tasks (Chi et al., 1981, Chi et al., 1982, Chomsky, 1957, Ericsson and Charness, 1994) but by implementing pair programming the novices can be elevated to almost expert status (Hannay et al., 2009). Pair collaboration compensates for lack of understanding so it would seem sensible to employ pair programming when task complexity is low and time is short or when task complexity is high and correctness is important.

The literature would seem to suggest that for optimum performance in traditional team environments the TMS develops hand in hand with differentiation with the emergence of specialists (Austin, 2003, Brandon and Hollingshead, 2004, Hollingshead, 2000, Wegner, 1995). The introduction of XP practices, it is argued, changes the way that TMS is utilised in a team and the mechanisms for the emergence of specialists, or in other words, differentiation. For optimum performance in XP teams, differentiation, and the reliance on TMS is reduced, as there is elevated communication and sharing of task work.

Hypothesis 3a. The eXtreme programming practice of pair programming is positively related to output of the team.

Hypothesis 3b. The eXtreme programming practice of small task-size is positively related to output of the team.

This first model was developed to start the exploration of the relationships between these themes. For this initial model it was decided to use a limited knowledge set and thus model simple routine tasks. Also, all agents started the simulation with no knowledge and no transactive memory simulating novice team members with no knowledge of each other. The introduction of more complex, non-routine tasks, and agents with more expertise will be addressed in a later chapter.

One of the strengths of agent-based modelling is the ability to analyse behaviour over time. As described above, there is plenty of literature relating to TMS and to XP but a scarcity relating to the effects of such over time. It would be expected that the effects of the above hypotheses would be amplified over time but this model will illustrate if that indeed takes place.

9.3 Method

The model used for this experiment was as described previously. The two dichotomous variables of task size and work mode were manipulated to produce four conditions for this simulation. Task size was 4 or 20 and work mode was working solo or working in pairs.

The range of knowledge was between 1 and 10 meaning that each task contained random numbers between 1 and 10 and each agent could develop a maximum of 10 knowledge streams. At the start of the simulation all agents had no knowledge and no transactive memory.

Number of agents	4
Task size	4 or 20
Work mode	Solo or in pairs
Knowledge range	1 to 10
Initial knowledge	None
Initial transactive memory	None

Table 4: Tabula Rasa: Parameters for model

Condition no.	Task size	Work mode	No. of iterations
1	4	Pair	2000
2	4	Solo	1000
3	20	Pair	400
4	20	Solo	200

Table 5: Tabula Rasa: Description of conditions

Each condition was simulated 50 times. The volume of work delivered for each simulation defined the number of iterations for each condition. For each condition 16000 task elements were delivered to the team. After a number of trials, this number of task elements was chosen as it was deemed sufficient to show emergent behaviour and any more did not reveal any further information.

9.4 Results

The results will describe firstly the descriptive statistics, look at the behaviour of the model over time and then look more closely at the behaviour of the model at the final iteration. Table 6 provides the descriptive statistics and the correlations among the constructs in the model. It shows that working mode (coded 0 for solo programming; 1 for pair programming) is positively significantly correlated to output and TMS accuracy and negatively significantly correlated with differentiation. Task size (coded 0 for large tasks; 1 for small tasks) is negatively, and significantly correlated with output and positively correlated with both differentiation and TMS accuracy. TMS accuracy is significantly, positively correlated with both differentiation and output and differentiation is negatively correlated with output. These results provide preliminary evidence of the importance of TMS, differentiation and eXtreme programming practices.

	Variable	n	Mean	SD	1		2		3		4	
1	Work mode											
2	Task size											
3	Differentiation	200	1.8E3	2.69E3	-.550	***	.598	***				
4	TMS accuracy	200	0.96	0.06	.712	***	.505	***	.189	***		
5	Output ratio	200	0.99	0.01	.899	***	-.303	***	-.843	***	.344	***

Notes: ***p<0.001, n = number of runs of the simulation. 50 runs for each of the 4 conditions

Table 6: Tabula Rasa: Descriptive statistics and correlations

Firstly graphs of the behaviour of the model over time will be considered and secondly the behaviour of the results of the final iteration of the model will be examined more closely with correlation and t-tests.

The following graphs illustrate the means of the model data over time. Error bars show standard error of the mean (SEM) where:

$$SEM = \sigma / \sqrt{n}$$

where σ is standard deviation and n is sample size. Where error bars are not apparent on the graphs, they are too small to be visible.

Figure 14 clearly shows that performance is consistently greater for pair programming conditions. For solo conditions output increases faster over time with output being consistently better for large tasks. This result supports hypothesis 3a; pair programming is positively related to output. There is also an effect of task size on output relating to hypothesis 3b, “small task size is positively related to output”. This graph shows that large tasks are associated more with higher output, especially for solo programming conditions, contradicting hypothesis 3b.

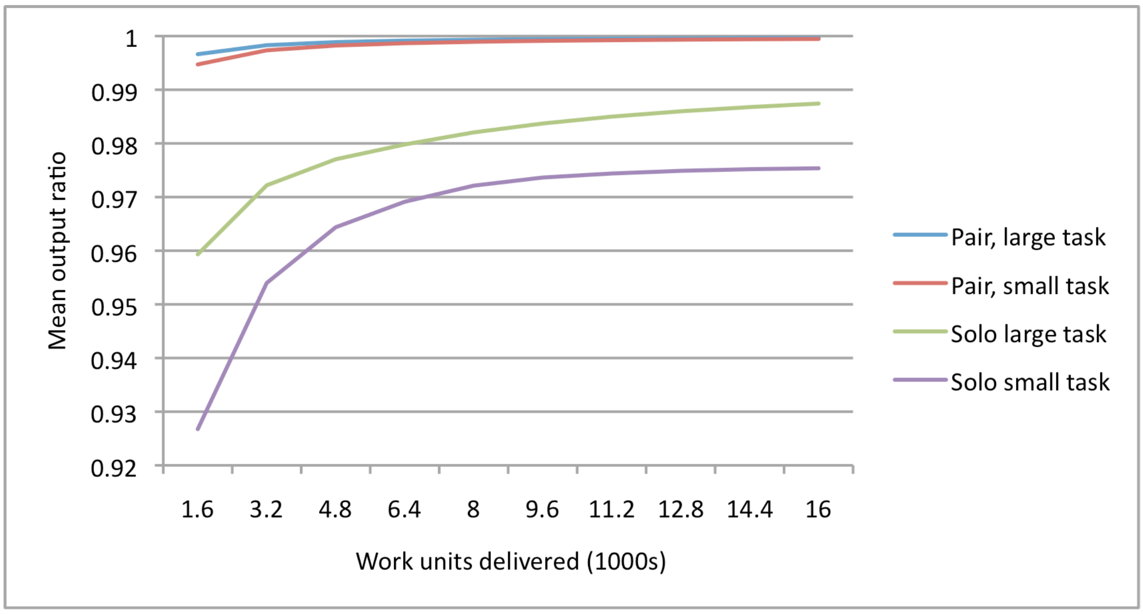


Figure 14: Tabula Rasa: Mean output ratio over time

The graph for differentiation over time (Figure 15) illustrates a dramatic increase in differentiation for solo programming with small tasks. For solo programming with large tasks the graph shows a smaller but increasing differentiation. The conditions with pair programming show low differentiation for both task conditions. When comparing this graph with the output, Figure 14 above, it would indicate that high differentiation produces lowest performance and that work mode would seem to moderate the relationship between differentiation and output; hypothesis 2a is supported. Additionally, the graph shows that large tasks result in lower differentiation, thus hypothesis 2b is unsupported.

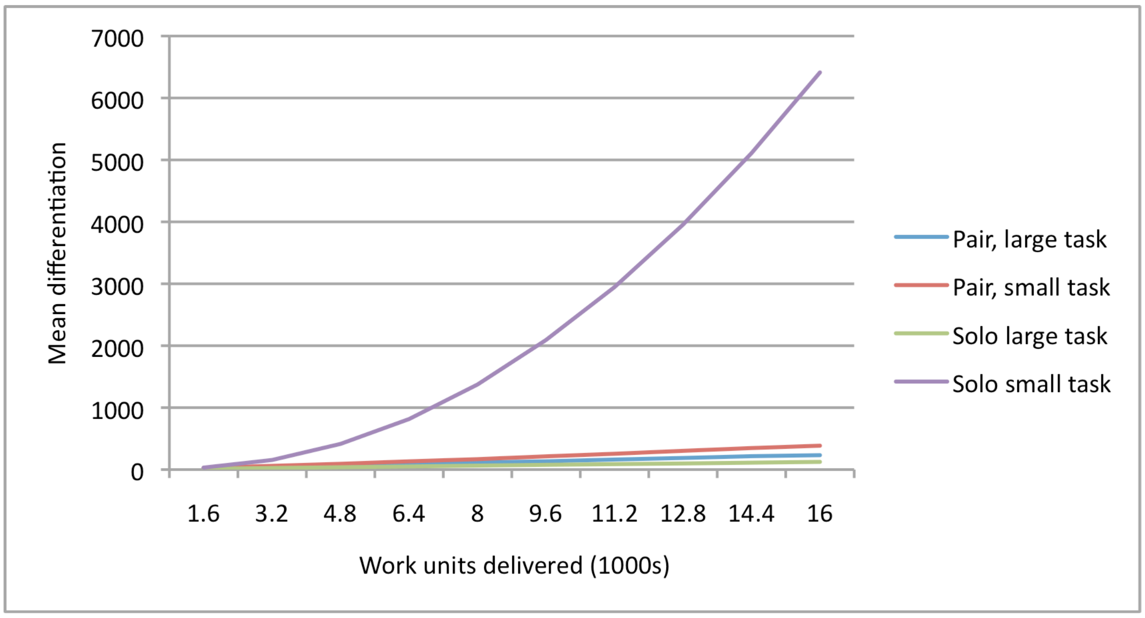


Figure 15: Tabula Rasa: Mean differentiation over time

The graph below (Figure 16), TMS accuracy, shows that TMS accuracy is consistently higher for pair programming conditions with small task conditions producing slightly higher TMS accuracy. Solo conditions with small tasks have higher TMS accuracy than with large tasks and the difference for solo programming is larger than for pair programming. By reviewing the graph of TMS completeness (Figure 17), it can be seen that the solo programming with large tasks condition took longer to build a TMS than the other conditions. As the simulation is separated into large tasks there are fewer opportunities to exchange information relating to each other's knowledge and by working solo this exacerbates this effect. Comparing this graph with the output graph (Figure 14) it would indicate that the relationship between TMS accuracy and output is moderated by work mode in support with hypothesis 1a however contradicting hypothesis 1a TMS accuracy is higher for pair programming. In support of hypothesis 1b by comparing the TMS accuracy graph with the output graph, task size does moderate the relationship between TMS accuracy and output but contradicting hypothesis 1b there is a negative relationship between task size and performance with TMS accuracy higher, and output lower for small tasks.

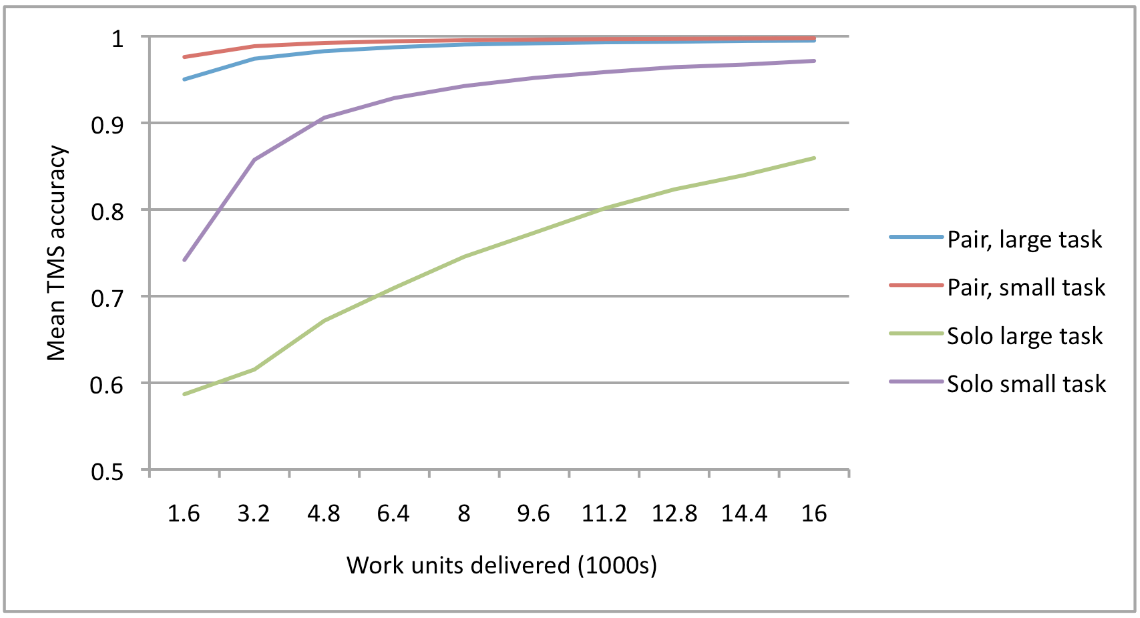


Figure 16: Tabula Rasa: Mean TMS accuracy over time

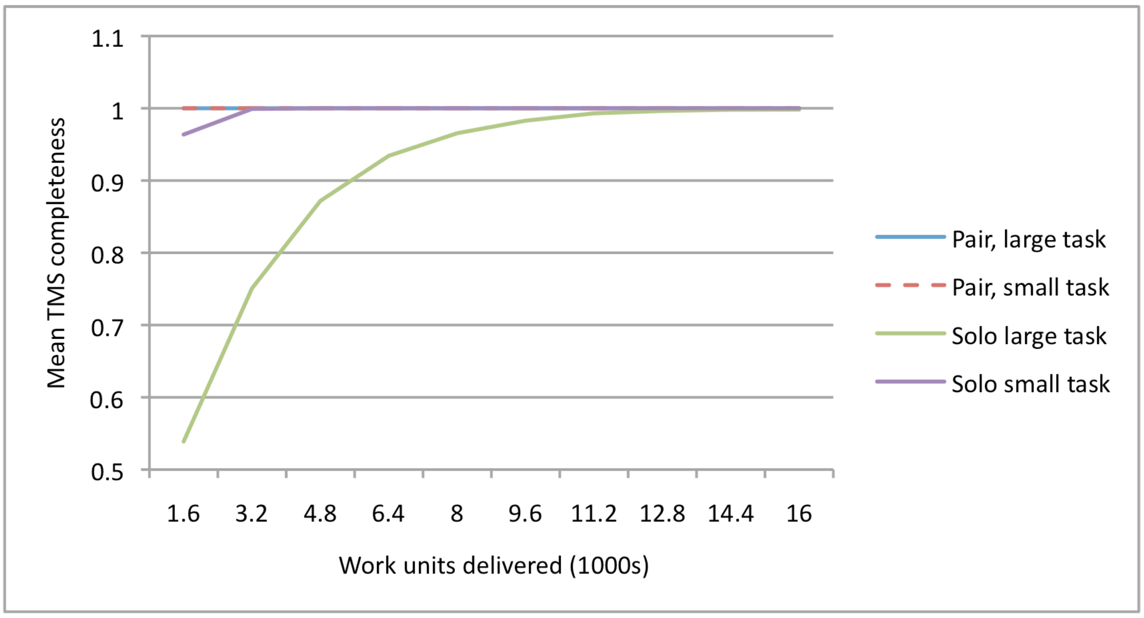


Figure 17: Tabula Rasa: TMS completeness over time

The following table shows the independent t-tests for work mode and task size for output ratio, TMS accuracy and differentiation. These tests confirm the results shown on the graphs above and give some statistical verification of the graphs.

	Variable	Mean	SE	*t*	df	
Output ratio	**Pair programming**	0.999	9.8 E-6	-28.87	99.05	***
	Solo programming	0.981	6.3 E-4			
TMS accuracy	**Pair programming**	0.996	2.1 E-4	-14.25	99.26	***
	Solo programming	0.915	5.7 E-3			
Differentiation	Pair programming	322.36	18.27	9.26	99.65	***
	Solo programming	3.268E3	317.63			

Notes: n=200, ***p<0.001, (n = number of runs of the simulation. 50 runs for each of the 4 conditions)

Table 7: Tabula Rasa: Independent t-tests for work mode.

	Variable	Mean	SE	*t*	df	
Output ratio	Small tasks	0.987	1.2 E-3	4.48	146.65	***
	Large tasks	0.994	6.2 E-4			
TMS accuracy	**Small tasks**	0.985	1.3 E-3	-8.23	106.31	***
	Large tasks	0.927	6.8 E-3			
Differentiation	**Small tasks**	3.40E3	304.70	-10.51	99.54	***
	Large tasks	191.64	16.05			

Notes: n=200, ***p<0.001, (n = number of runs of the simulation. 50 runs for each of the 4 conditions)

Table 8: Tabula Rasa: Independent t-tests for task size.

To test hypotheses 3a and 3b independent t-tests were carried out to determine if work mode and task size had significant effects on output. Pair programming conditions (M = 0.999, SE = 9.8 E-6), resulted in significantly higher output than solo programming conditions (M = 0.981, SE = 6.3 E-4). This difference was significant, t(99.05) = -28.87, p>0.001; This supports hypothesis 3a.

Large task conditions (M = 0.994, SE = 6.2 E-4), resulted in significantly higher output than small task conditions (M = 0.987, SE = 1.2 E-3). This difference was significant t(146.65) = 4.48, p>0.001. This contradicts hypothesis 3b, which predicted that small tasks would result in significantly higher output.

T-tests were also carried out for the effects of pair programming and task size on TMS accuracy and differentiation, which show significant positive main effects for all relationships apart from the effect of working mode on differentiation. This relationship showed a positive relationship between large tasks and differentiation. To assess the moderation effects hypothesised the graphs below show the relationships between both TMS accuracy and differentiation with output and each other.

Below are graphs derived from data from the final iteration of the model, which show the relationships hypothesised. Figure 18 shows the relationship between output and TMS accuracy. The blue lines show the interaction effects of task size and the red lines show the interaction effects of the work mode, showing some support for hypotheses 1a and 1b. The graph shows the strong effect of pair programming on output and that TMS accuracy is higher for pair programming conditions than for solo programming conditions. In addition it shows an advantage for large tasks for both solo programming conditions as the results of the t-test showed. It also shows a larger difference in output and TMS accuracy relating to task size for solo programming conditions than for pair programming conditions. This demonstrates the moderating effects of both work mode and task size on the relationship between TMS accuracy and output but contradicts hypotheses 1a, as TMS accuracy is high for pair programming conditions.

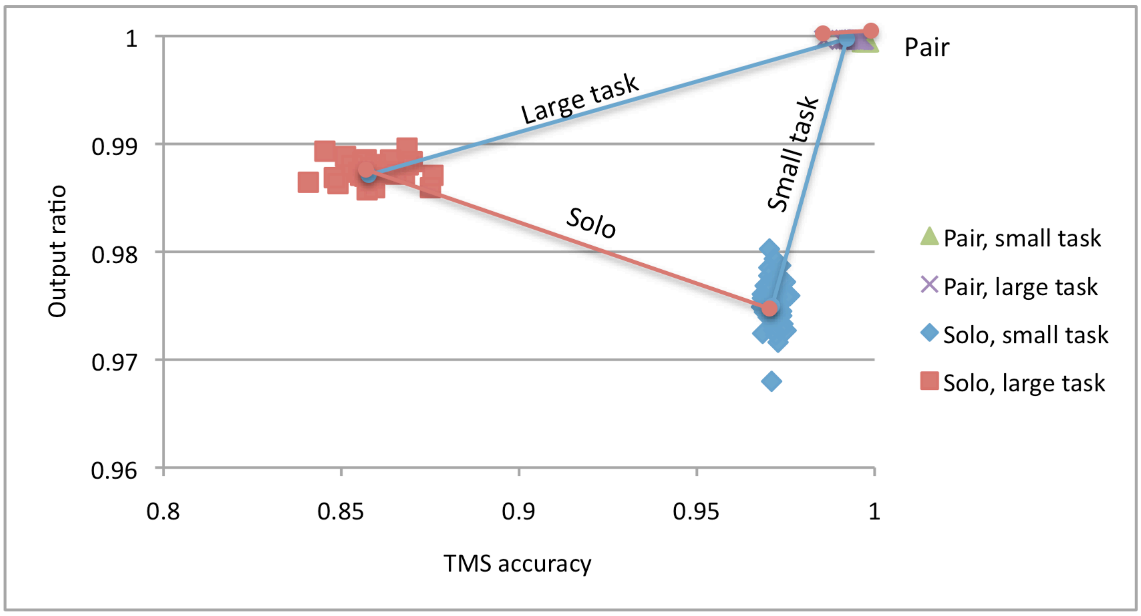


Figure 18: Tabula Rasa: output by TMS accuracy

Hypotheses 2a and 2b assert that the relationship between differentiation and output is moderated by XP conditions. Figure 19 depicts this, showing that, for solo programming, small task conditions, differentiation is high for lowest output and large task size seems to have a small advantage. The blue lines show the interaction effects of task size and the red lines show the interaction effects of the work mode, showing some support for hypotheses 2a and 2b. However for pair programming, performance is highest with task size appearing to have little effect. This demonstrates the moderating effect of XP practices on the relationship in support of hypotheses 2a and 2b, however, differentiation is lower for pair programming and large tasks.

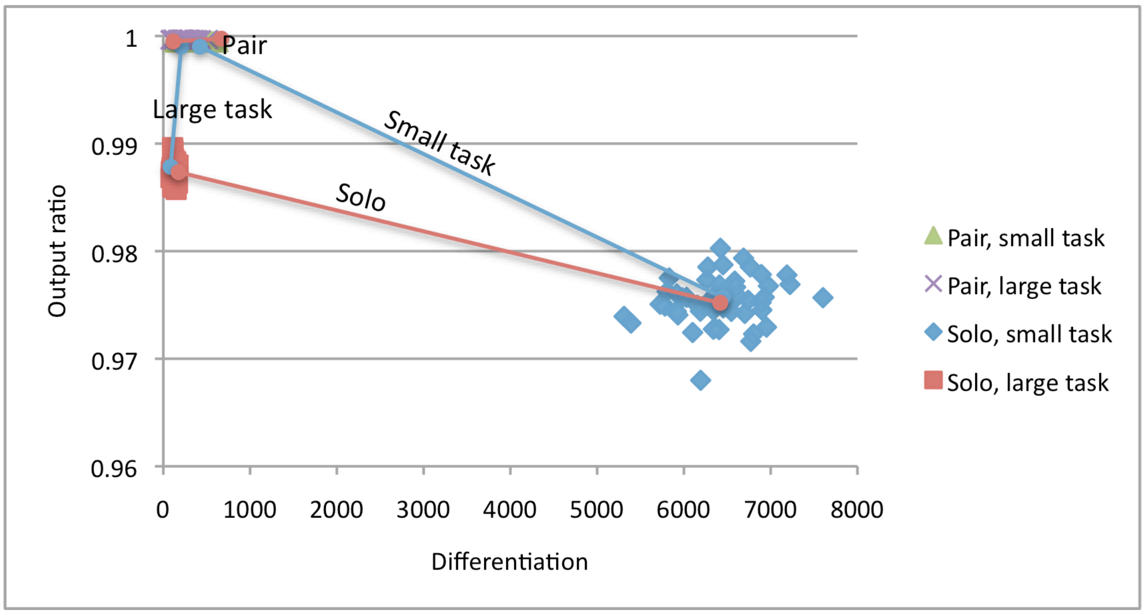


Figure 19: Tabula Rasa: output by differentiation

The graph at Figure 20 shows the relationship between TMS accuracy and differentiation. The blue lines show the interaction effects of task size and the red lines show the interaction effects of the work mode. It shows a negative relationship for pair programming with TMS accuracy high and differentiation being low. For solo programming TMS accuracy is lower with TMS accuracy and differentiation being higher for small tasks.

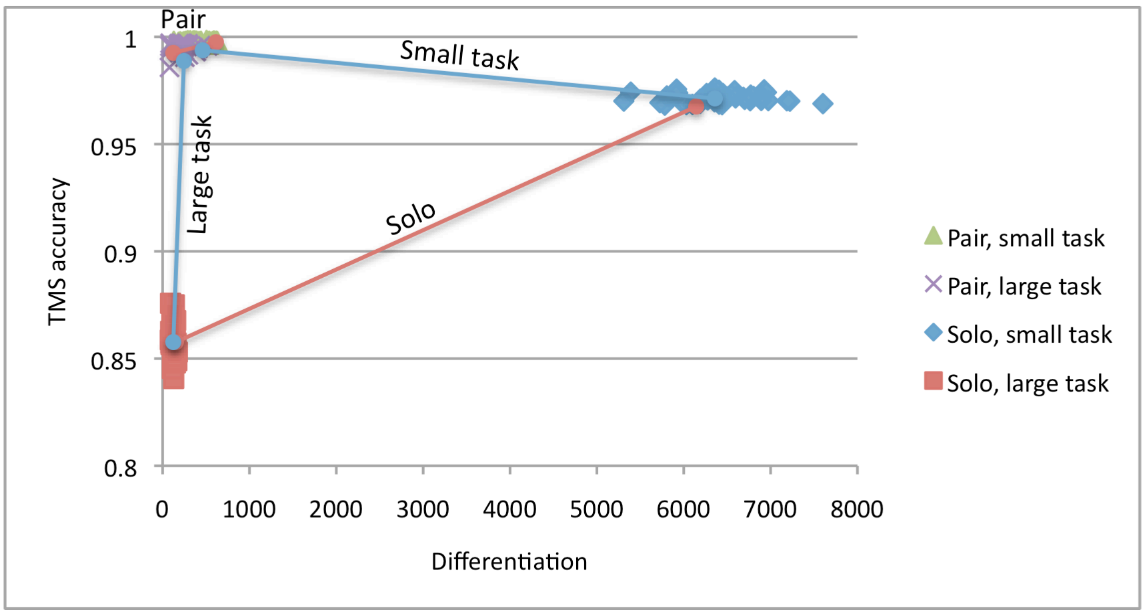


Figure 20: Tabula Rasa: TMS accuracy by differentiation

9.5 Discussion

The hypotheses for this model were based on the literature relating to team performance, transactive memory systems, differentiation, tasks and XP techniques. There are many dependencies, complex relationships and some potential disagreement. The aim of this model was to try and clarify how the different elements relate to one another with a view to deeper investigation in future models.

The hypotheses considered the relationships between work mode and task size factors, TMS, differentiation and output. Hypotheses 1a, 1b, 2a and 2b all predicted a moderating effect of work mode and task size on relationships with output, and 3a and 3b predicted that pair programming and small tasks would have a positive effect on output.

The independent t-tests and the graphs show a strong main effect of working mode on output demonstrating that pair programming has a positive relationship with output in support of hypothesis 3a. The t-test for task size showed a significant positive main effect between task size and output contradicting hypothesis 3b. T-tests were also carried out for the effects of pair programming and task size on TMS accuracy and differentiation, which show significant positive main effects for all relationships apart from the effect of working mode on differentiation. This relationship showed a positive relationship between large tasks and differentiation.

The results of the t-tests provide some information but of course the dynamics of the results are more complex as these results show main effects and do not take into account any moderation for work mode or task size. The graphs, Figures 18 to 20, can give some insight into these moderation effects. It was expected that differentiation would be reduced for pair programming and small task conditions and this was partially found. Differentiation remained very low for pair programming conditions and for solo conditions with large tasks, however, differentiation rose quickly for solo conditions with small tasks.

It was also expected that TMS accuracy would be reduced for pair programming and small task conditions however this was not found. In contradiction of the hypotheses, TMS accuracy was higher for pair programming conditions and small tasks produced higher TMS accuracy for both programming conditions.

In summary, these results suggest that in pair programming conditions, with low differentiation, TMS accuracy is important for output. According to this model, pair programming generates high performance, high TMS accuracy and low differentiation, with large tasks having a weaker but positive effect on output. Solo programming produces lower performance and lower TMS accuracy, with an advantage for large tasks; however, differentiation with small tasks produces substantially increased differentiation.

There is some discrepancy in the literature relating to work mode and task size factors and differentiation. Small tasks are expected to increase differentiation whereas pair programming is expected to reduce it. Littlepage et al. (2008) predicted high differentiation with smaller tasks, resulting in higher performance but pair programming is expected to reduce differentiation and raise performance (Hannay et al., 2009, Maruping et al., 2009, Williams et al., 2000). Some explanation for this apparent discrepancy was sought by this model to identify the cumulative effects of the combination of XP factors. This model did indeed find that small tasks increased differentiation in solo conditions however in pair programming conditions differentiation was the lowest. The model also found in support of the literature (Hannay et al., 2009, Maruping et al., 2009, Williams et al., 2000) that pair programming reduced differentiation and increased output.

It is clear from the model that pair programming has a dominant effect but the results for solo programming can be viewed in respect of the literature. The solo programming condition with small tasks produced the lowest output yet substantially higher differentiation than any other condition. Austin (2003) suggests that it is easier to recognise experts when differentiation is higher and thus TMS accuracy will be higher. This model demonstrates that, for solo conditions, this is the case however this does not translate into higher output in contradiction of the literature that suggests that performance advantages deriving from TMS are greater when group members have higher differentiation (Austin, 2003, Larson, 2007, Lewis, 2004, Littlepage et al., 2008). The results of this model also conflict with the findings of Lewis who found that teams with initially diverse knowledge were better able to develop TMS than teams with overlapping knowledge and that a stronger TMS was positively related to team performance. So, for solo programming this model found that higher output was attained for large tasks but with lower differentiation and lower TMS accuracy.

Austin (2003) found that TMS accuracy is the most significant predictor of team performance and this model supported that finding**.**

Whether pair programming creates high or low interdependence has not been addressed by the literature. It would seem plausible that as cognitive interdependence relates to the outcome of an individual being dependent on other team members then working in pairs would create high cognitive interdependence. If this is so then pair programming may also foster high levels of TMS accuracy, as it is dependent on cognitive interdependence. Maruping et al. (2009) found that collective ownership of code attenuates the relationship between expertise coordination and software product quality and it could be argued that pair programming will have similar effects on TMS. By working together closely and having overlapping knowledge of the project, there is reduced interdependency in the team and thus less reliance on the TMS. The results of this simulation contradict Maruping finding that pair programming produced high levels of output and corresponding high levels of TMS accuracy indicating that even with high levels of shared knowledge, i.e. low differentiation, TMS accuracy remains important.

The range of knowledge used in this model was very small, only 1 to 10, and it could be construed as representing routine, repetitive tasks. Indeed, the same elements will appear many times in the tasks presented to the agents in the model, as there are so few of them. This may explain why the differentiation for working in pairs produces such high differentiation; by working together and sharing knowledge in such a narrow range of knowledge team members' knowledge grows very quickly.

A number of studies have suggested that task novelty and complexity would introduce higher levels of task interdependence (Brandon and Hollingshead, 2004, Faraj and Sproull, 2000, Hoegl and Gemuenden, 2001) however Hoegl and Gemuenden found limited support for the idea that task innovativeness affects the relationship between team coordination and performance. Lui and Chan (2006) in a study looking at expert versus novice programmers suggest that pair programming works best when a pair encounters challenging programming problems.

Due to the very small range of knowledge over which this model operated, namely 1 to 10, the TMS was able to 'get up to speed' very quickly. Each agent was able to learn within a few iterations the knowledge of the other agents. It is suggested that in a model using a much larger knowledge range the TMS would take some time to develop.

9.6 Conclusion

The hypotheses for this simulation predicted that the XP techniques of pair programming and small task size would improve output and this was partially supported with pair programming generating higher output. Large tasks were associated with higher output.

It was expected that pair programming would alter the relationships between TMS accuracy and differentiation with output and this was found. Pair programming improved output and there was generally a positive relationship between TMS accuracy and output, and, as expected pair programming generated lower differentiation. The graphs indicated some moderating effects of work mode and task size on the relationships between output, TMS accuracy and differentiation. For solo programming this model found that higher output was attained for large tasks but in contradiction of the literature, with lower differentiation and lower TMS accuracy.

This model revealed some interesting insights into the relationships between work mode and task size factors, TMS, differentiation, and performance. The model used agents with no initial knowledge or TMS, which is unrealistic, and used a very simple knowledge range. Subsequent chapters will describe further simulations using a substantially larger knowledge range, 1 to 100, simulating higher task complexity and novelty, and agents will be furnished with expertise and TMS for simulations to investigate the effects in more realistic settings.

Chapter 10: Model Novelty

10.1 Introduction

The previous chapter focussed on a simple initial model with a small range of knowledge simulating repetitive routine tasks. The model found that work mode and task size to a lesser degree moderate the relationships between output and TMS accuracy, and differentiation. Pair programming generated higher output with low differentiation and high TMS accuracy, while solo programming produced lower output with an advantage for large tasks generating lower TMS accuracy and lower differentiation.

Previously the model has used a knowledge range of only 1 to 10 and representing routine and repetitive tasks. This chapter describes the same model but replacing the knowledge range with 1 to 100. As the task sizes for the model are 4 or 20 in size this larger knowledge range will deliver tasks that are more novel and complex; the intention being to simulate the environment of software developers where they face less routine tasks that include novel, complex problems.

10.2 Theoretical framework

Software development projects are not usually repetitive and routine; they often vary considerably in levels of innovativeness with development of new software solutions presenting higher levels of task complexity, novelty and technical uncertainty (Hoegl et al., 2003).

Increases in complexity and novelty of tasks will always have a negative effect on performance. (Espinosa et al., 2007a, Gibson, 1999). There are contradictory views on the moderating influence of task complexity on the relationship between TMS and performance and since complex tasks introduce more demand for team members it may be argued that more complex tasks increase team interdependency.

Low cognitive interdependence can work for routine simple tasks but when tasks are complex, innovative and non-routine there is a high need for cognitive interdependence (Brandon and Hollingshead, 2004). A study by Faraj and Sproull (2000) considered team coordination by looking at

TMS and performance of software development teams. They intentionally used homogenous tasks but tested for the moderating effect of task uncertainty and found no significant relationships. Drawing on previous research however they propose that task uncertainty and complexity would moderate the relationship between expertise coordination (TMS) and team performance in a more heterogeneous task set. There is limited support, however, for the notion that task innovativeness moderates the relationship between the TMS and team performance (Hoegl and Gemuenden, 2001).

It has been found that groups with lower member turnover have higher performance (Argote et al., 1995). The TMS of a team is damaged by the replacement of a member and it takes time for it to adapt to the new team constitution. However the impact of turnover on performance is lower for complex tasks than for simple tasks, implying that innovation lowers the deleterious effect of turnover on the team. If the work of the team is novel and requires less established knowledge then damage to the TMS will have less effect. So higher complexity and novelty will require lower dependence on TMS (Argote et al., 1995, Espinosa et al., 2007a).

A contradictory view has been proposed; if a task is simple or repetitive team members can rely on existing knowledge and standard procedures. For higher complexity or non-routine tasks the requirements for knowledge and interdependency in the team are unknown to a greater or lesser extent. If the knowledge is already in place or the task requires new knowledge a complex task brings along ambiguity and greater difficulty. Complex tasks require greater interdependency and thus the impact on of the TMS on performance is increased with increased task complexity (Akgun et al., 2005, Ren et al., 2006). Increased task complexity and novelty enhances team knowledge by bringing new knowledge into the team and calls for more cooperation and coordination within the team.

The work carried out by software development teams is often highly innovative. In projects of high innovativeness collaboration and communication is more essential to interpret and manage complex information, and share skills and information. For this reason the TMS is more strongly related to team performance in highly innovative software projects which supports the fundamental notion that it is the nature of the task that determines whether collaboration is beneficial (Hoegl et al., 2003).

With high levels of collaboration pair programming responds to the needs generated by task complexity and novelty. Pair programming is more effective for complex tasks but not as useful in simple routine tasks (Lui and Chan, 2006) but this advantage applies mainly to novices and to a lesser extent experienced programmers. However, it is possible that the benefits of pair programming will increase for larger, more complex tasks and if the pair programmers have a chance to work together over a longer period of time, thus developing a stronger TMS (Arisholm et al., 2007, Hannay et al., 2009).

It has been proposed that tasks divided into smaller chunks and higher differentiation of team knowledge result in higher team performance (Littlepage et al., 2008). The Littlepage et al. study, however, was carried out on staff doing relatively routine work and the findings may not apply in an innovative environment such as software development where tasks are often more novel, complex and non-routine (Brandon and Hollingshead, 2004). There is an alternative and contradictory view that there is a positive relationship between task size and performance when looking at familiar tasks in software development (Espinosa et al., 2007a). Support for this is scarce, however, endorsing the view of Littlepage et al.

The results of the previous chapter showed that with low differentiation, high levels of TMS were important for high performance for pair programming conditions. This contradicts traditional research on TMS which states that there is a positive relationship with TMS and differentiation but supports a study by Espinosa (2007a) who found that team familiarity (TMS) and task familiarity (expertise) appear to substitute for each other. In other words, having more task knowledge makes one less dependent on colleagues, whereas having knowledge of your team members' abilities makes one less dependent on task knowledge.

With the higher levels of task novelty and complexity introduced in this model, and the contradictions in the literature relating to task novelty and complexity the following hypotheses are proposed:

Hypothesis 1. Output is lower with novel complex tasks for all conditions than for routine tasks.

This is fairly intuitive but it seems sensible that for unfamiliar tasks there is likely to be lower output. It is expected that this will apply for all conditions.

Hypothesis 2. Output is higher for pair programming and small task conditions.

As seen in the last model, it is expected that output will be higher for pair programming. In addition it is expected that small tasks will be more successful than in the routine tasks model.

Hypothesis 3. The reduction in output for complex novel tasks is attenuated by the use of pair programming and small tasks.

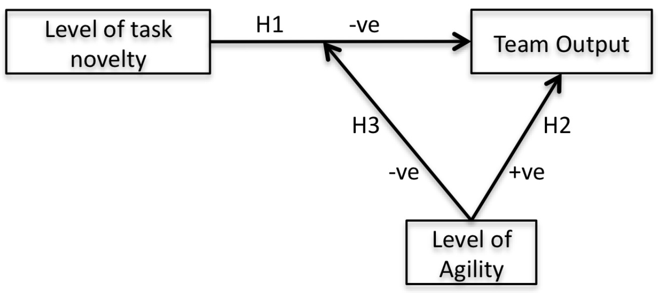


Figure 21: Novelty: Diagram of hypotheses relationship 1,2 and 3

As pair programming is more successful for more complex tasks and knowledge is more evenly distributed around the team allowing a greater ability to address unfamiliar tasks, the reduction in output associated with more complex and novel tasks will be lower for pair programming conditions. The model will also show if task size has an influence on the relative output for routine and complex tasks.

Hypothesis 4. Pair programming and small task practices affect the relationship between TMS and output.

As stated, the literature is contradictory on the benefits of pair programming with novel and complex tasks. This model will help to determine the relationship between TMS and output with more novel and complex tasks.

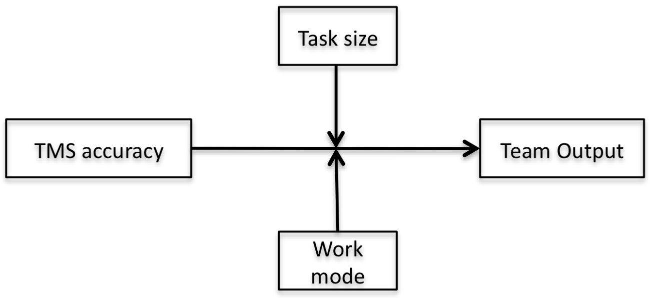


Figure 22: Novelty: Diagram of hypothesis relationship 4

Hypothesis 5. XP practices affect the relationship between differentiation and output.

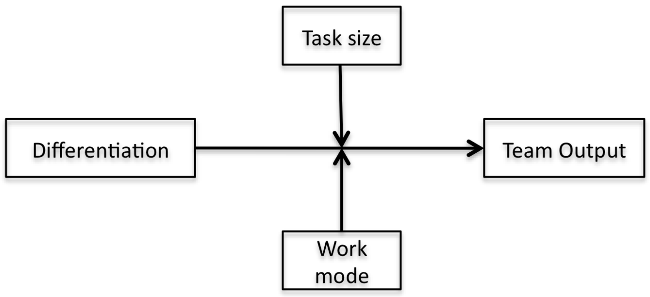


Figure 23: Novelty: Diagram of hypothesis relationship 5

With this model being operated with higher novelty this will result in lower differentiation for XP conditions. The knowledge that each team member is gaining is being distributed about a larger range; therefore specialism is less likely with lower repetition of task content.

10.3 Method

The model described in this chapter increased the range of knowledge in the model from 1 to 100 to simulate more innovative tasks, so all tasks could contain numbers up to 100 and that the knowledge that an agent could develop will range from 1 to 100. This presents tasks to the agents, during each simulation, which will contain novel elements more often.

The three dichotomous variables of task size, work mode and knowledge range were manipulated to produce four conditions for this simulation. Task size was 4 or 20; work mode was working solo or working in pairs and knowledge range was from 1 to 100.

At the start of the simulation all agents had no knowledge and no transactive memory.

Number of agents	4
Task size	4 or 20
Work mode	Solo or in pairs
Knowledge range	1 to 100
Initial knowledge	None
Initial transactive memory	None

Table 9: Novelty: Parameters for model

Condition no.	Task size	Work mode	No. of iterations
1	4	Pair	2000
2	4	Solo	1000
3	20	Pair	400
4	20	Solo	200

Table 10: Novelty: Description of conditions

Each condition was simulated 50 times.

The volume of work delivered for each simulation defined the number of iterations for each condition. For each condition 16000 task elements were delivered to the team.

10.4 Results

The results will describe firstly the descriptive statistics, look at the behaviour of the model over time and then look more closely at the behaviour of the model at the final iteration. Table 11 provides the descriptive statistics and the correlations among the constructs in the model. It shows that working mode (coded 0 for solo programming; 1 for pair programming) is again positively, significantly correlated to output indicating that pair programming is highly positively correlated with output, supporting hypothesis 2. Conversely, task size (coded 0 for large tasks; 1 for small tasks) is negatively, significantly, correlated with output indicating that large tasks produce higher output. With regard to TMS accuracy, it is significantly, positively correlated with output. Notably, in contrast to the results in the previous chapter, differentiation is significantly and positively correlated with output and working mode, indicating that for pair programming conditions differentiation is high for high output. This gives preliminary evidence in support of hypothesis 5.

	Variable	n	Mean	SD	1		2		3		4	
1	Working mode											
2	Task size											
3	Differentiation	200	25.76	8.43	.558	**	.694	**				
4	TMS accuracy	200	.813	.176	.932	**	.262	**	.765	**		
5	Output	200	.967	.032	.900	**	-.326	**	.203	**	.682	**

Notes: **p<0.01, n = number of runs of the simulation. 50 runs for each of the 4 conditions

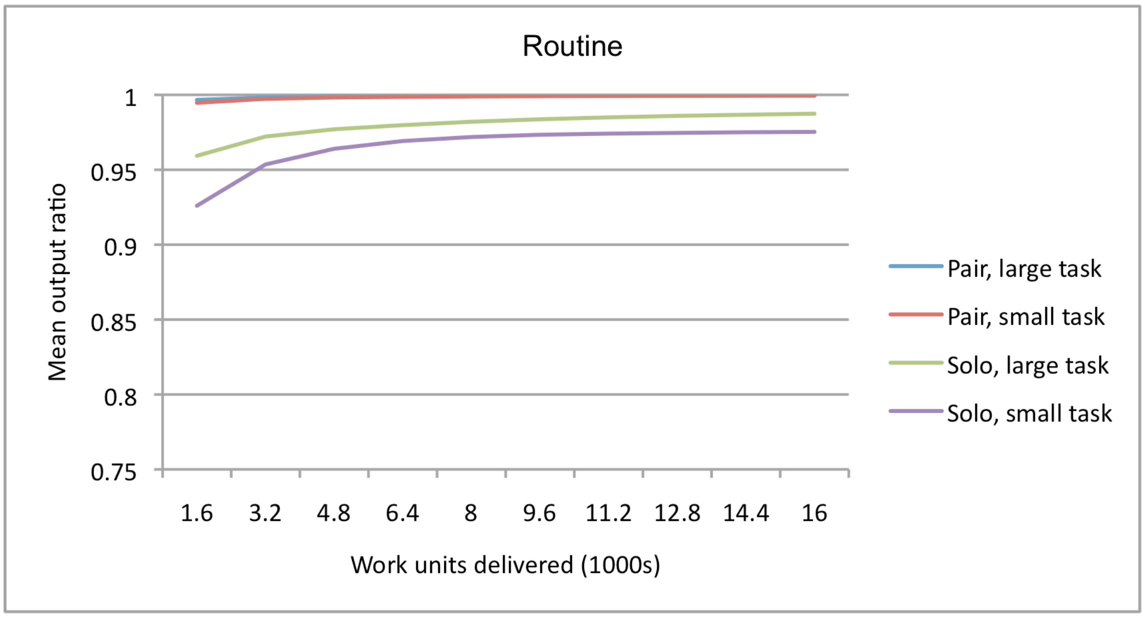
Table 11: Novelty: Descriptive statistics and correlations

The following graphs show the results of the model over time. Error bars show standard error of the mean (SEM) where:

$$SEM = \sigma / \sqrt{n}$$

where σ is standard deviation and n is sample size. Where error bars are not apparent on the graphs, they are too small to be visible.

For comparison, the graphs from the previous chapter, for routine tasks, have been placed alongside the graphs relating to these results. The graph in Figure 24 shows output over time and it can be seen that in general, output is lower than for routine tasks supporting hypothesis 1. As with the previous model pair programming produces higher output than for solo conditions (supporting hypothesis 2) and it can be seen that the reduction in output due to task novelty is greater for solo conditions, in support of hypothesis 3.



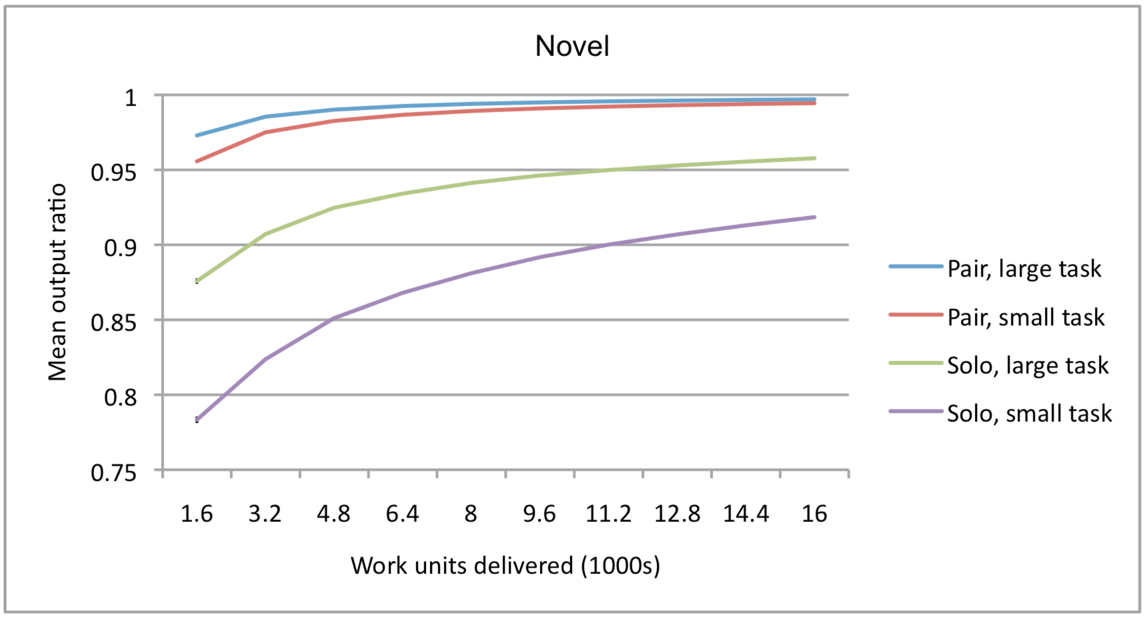
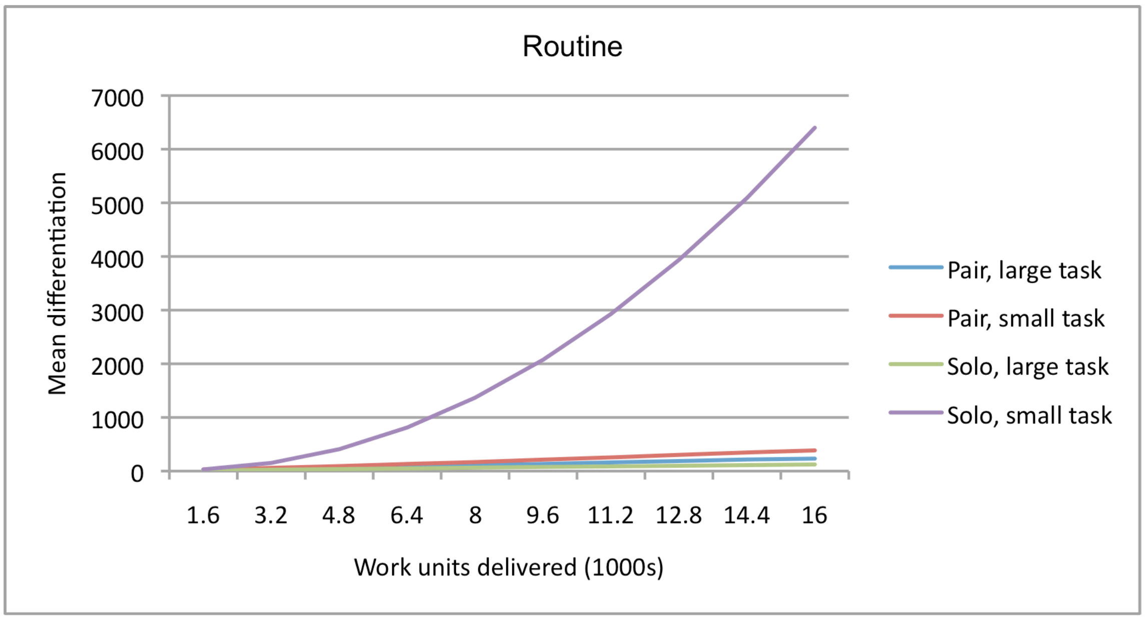
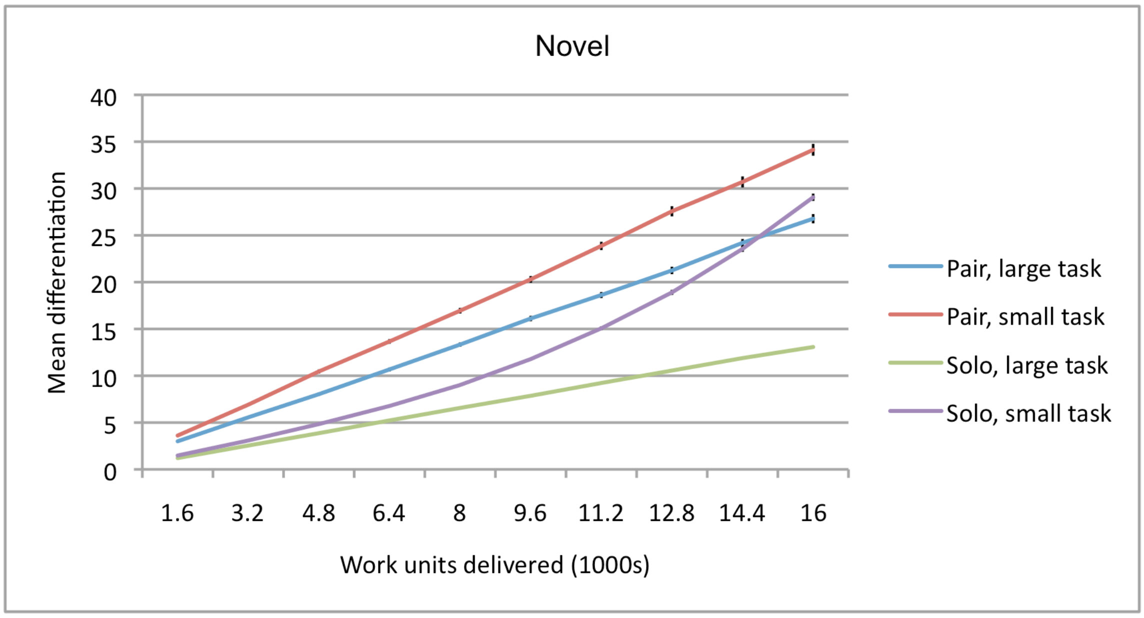


Figure 24: Novelty: Mean performance over time for routine and novel tasks

As can be seen in the graphs at Figure 25 results for differentiation over time are very different for this simulation than for the previous one simulating routine tasks. As suggested by the correlation results, the graph depicting differentiation over time clearly shows higher differentiation for pair programming conditions contradicting hypothesis 5, however, differentiation for solo programming is higher for small tasks increasing faster than for large tasks, with the rate of increase higher than for pair programming. The graph would indicate that differentiation for solo small tasks would become higher than for pair programming if the simulation were run for longer.

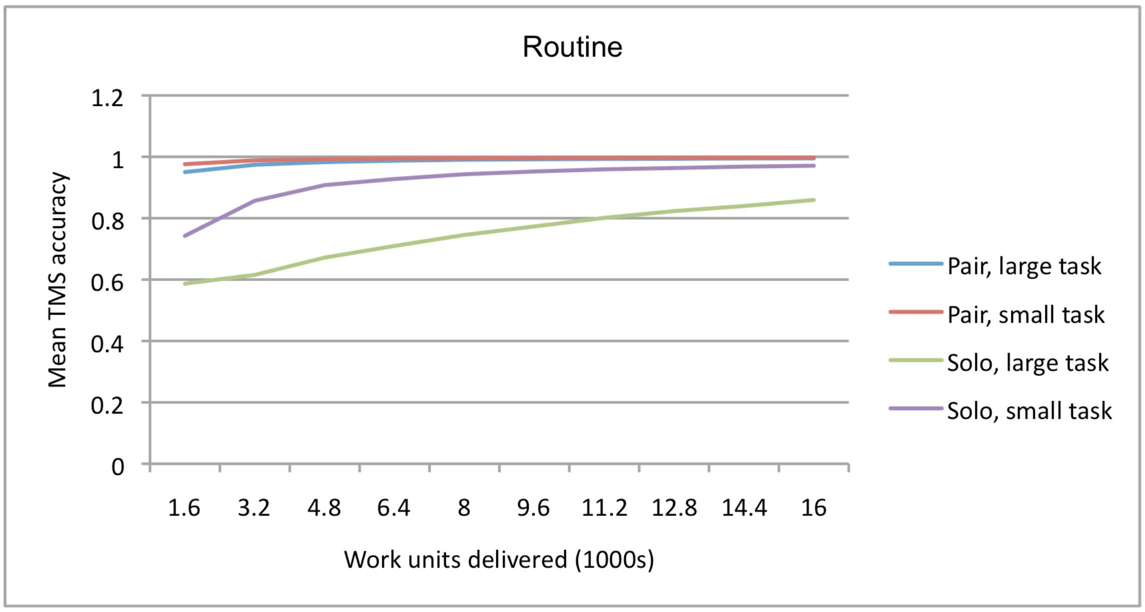




(Please note different axes for differentiation)

Figure 25: Novelty: Mean differentiation over time for routine and novel tasks

There is a marked difference between TMS accuracy over time for pair and solo programming conditions as depicted in the graph at Figure 26. For the former, TMS accuracy increases early and levels off to maintain a level with slightly higher accuracy for small tasks. For the latter, TMS accuracy drops slightly to start with a larger decrease for large tasks than for small tasks. For pair programming conditions increasing the complexity and novelty has had little effect on TMS accuracy and output other than reducing both slightly. For solo conditions, however, increasing novelty and complexity has had a greater effect by reducing TMS accuracy particularly for small tasks. This lends support to hypothesis 4. By reviewing the graphs for both output and TMS accuracy, as with routine tasks it can be seen that for both pair and solo programming there is an inverse relationship between TMS accuracy and output, with higher output for large tasks with lower TMS accuracy.



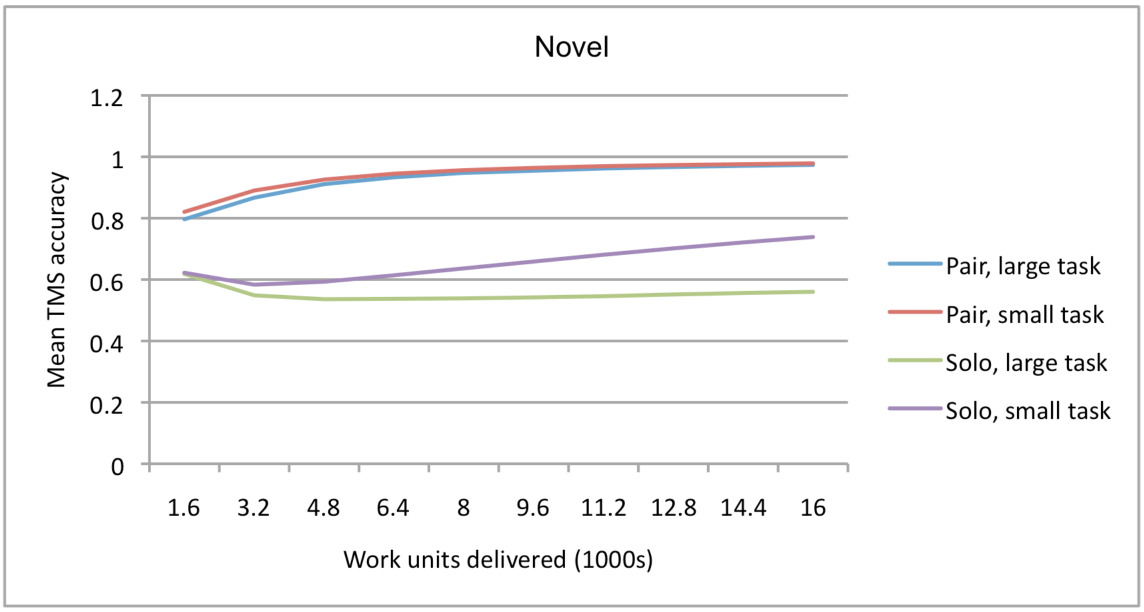
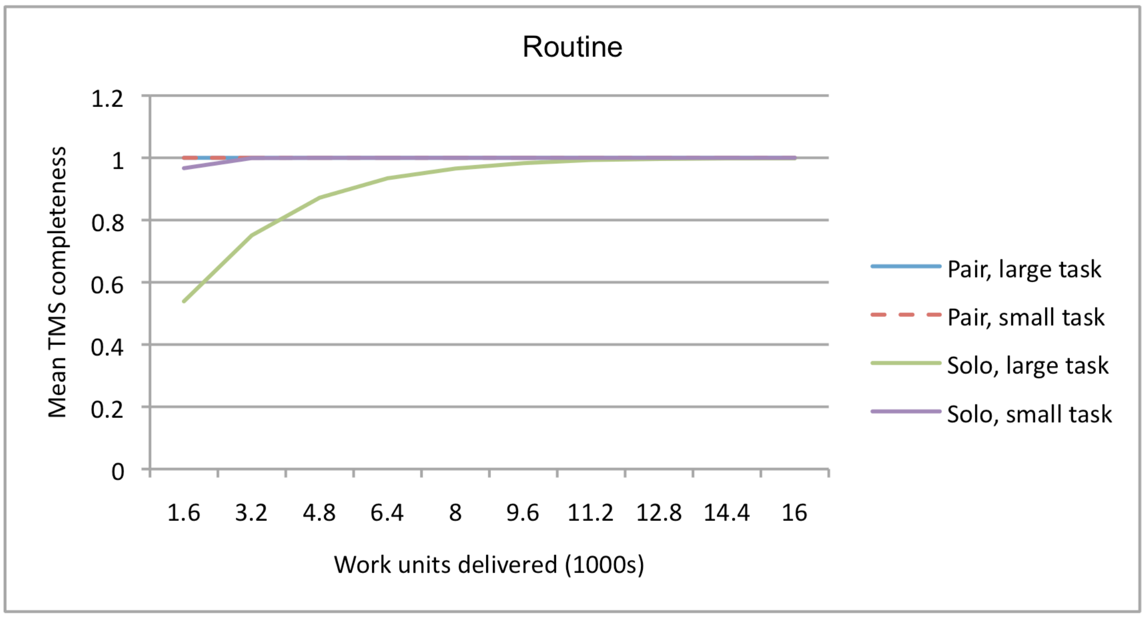


Figure 26: Novelty: Mean TMS accuracy over time for routine and novel tasks

The completeness of the TMS is affected more by the different conditions for when the tasks are more novel and complex, as shown in the graph in Figure 27. For solo conditions the TMS takes much longer to become complete and, for the duration of this simulation, the TMS is does not reach completion.



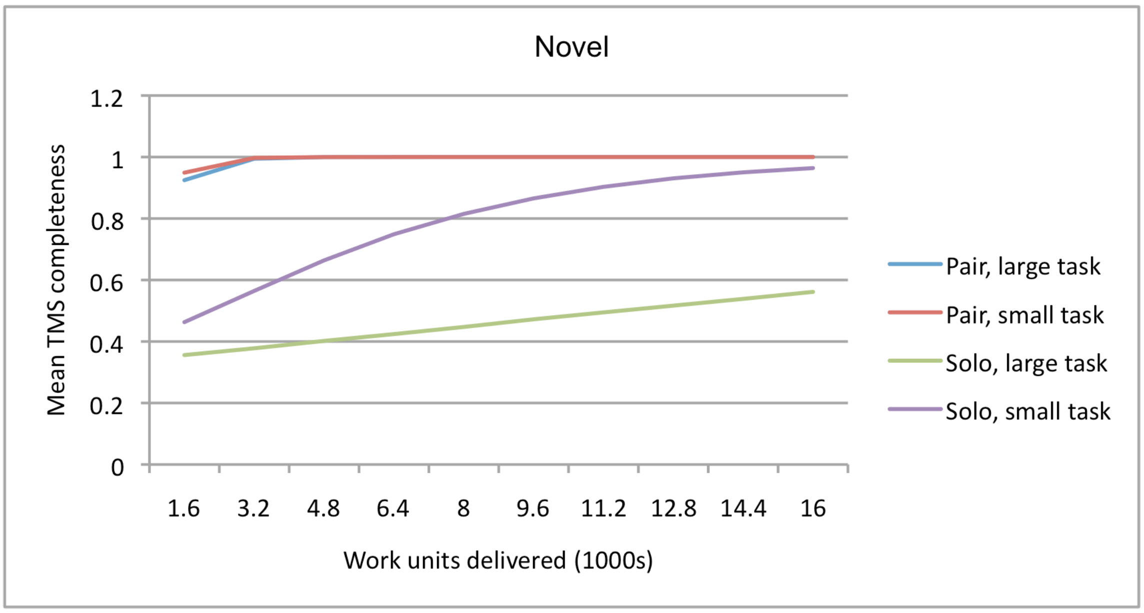


Figure 27: Novelty: Mean TMS completeness over time for routine and novel tasks

To compare the means of the results independent t-tests were carried out to establish whether the data were derived from the same or different populations. Although it is clear from the graphs, to test hypothesis 1 an independent t-test was carried out using the routine data from the previous model to determine if routine and novel tasks had significant overall effects on output. Routine tasks (M = 0.990, SE = 7.2 E-4), resulted in higher output than novel tasks (M = 0.967, SE = 2.3 E-3). In support of hypothesis 1 this difference was significant, t(238.2) = 9.89, p>0.001.

Further independent t-tests were carried out on only the data from this novelty model to determine if work mode and task size had significant effects on output. Pair programming conditions (M = 0.996, SE = 1.3 E-4), resulted in significantly higher output than solo programming conditions (M = 0.938, SE = 2.0 E-3). In support of hypothesis 2 this difference was significant, t(99.81) = -28.97, p>0.001.

Large task conditions (M = 0.977, SE = 2.0 E-3), resulted in significantly higher output than small task conditions (M = 0.956, SE = 3.8 E-3). In support of hypothesis 2 this difference was significant, t(148.34) = 4.86, p>0.001..

T-tests were also carried out for the effects of pair programming and task size on TMS accuracy and differentiation. From the table below it can be seen that pair programming and small tasks have significant positive relationships with TMS accuracy (supporting hypothesis 4) and with differentiation (supporting hypothesis 5). However, for output only pair programming has a significant positive relationship with output with large tasks having an advantage for output.

	Variable	Mean	SE	t	df	
Output	**Pair programming**	0.996	1.3 E-4	-28.97	99.81	***
	Solo programming	0.938	2.0 E-3			
TMS accuracy	**Pair programming**	0.976	2.7 E-4	-36.32	99.18	***
	Solo programming	0.649	9.0 E-3			
Differentiation	**Pair programming**	30.453	0.543	-9.45	171.17	***
	Solo programming	21.067	0.829			

Notes: n=200, ***p<0.001, n = number of runs of the simulation. 50 runs for each of the 4 conditions

Table 12: Novelty: Independent t-tests for work mode

	Variable	Mean	SE	t	df	
Output	Small tasks	0.956	3.8 E-3	4.86	148.34	***
	Large tasks	0.977	2.0 E-3			
TMS accuracy	**Small tasks**	0.859	1.2 E-2	-3.82	158.90	***
	Large tasks	0.767	2.1 E-2			
Differentiation	**Small tasks**	31.60	0.45	-13.56	164.17	***
	Large tasks	19.92	0.73			

Notes: n=200, ***p<0.001, n = number of runs of the simulation. 50 runs for each of the 4 conditions

Table 13: Novelty: Independent t-tests for task size

To further assess the moderation effects hypothesised the graphs below show the relationships between TMS accuracy, differentiation and output at the final iteration. Figure 28 shows the relationship between output and TMS accuracy. The blue lines show the interaction effects of task size and the red lines show the interaction effects of the work mode, showing some support for hypotheses 4. As before, it shows the strong effect of pair programming on output and shows that TMS accuracy is higher for pair programming conditions than for solo programming conditions. In support of hypothesis 4 it also shows a larger difference in output and TMS accuracy relating to task size for solo programming conditions than for pair programming conditions.

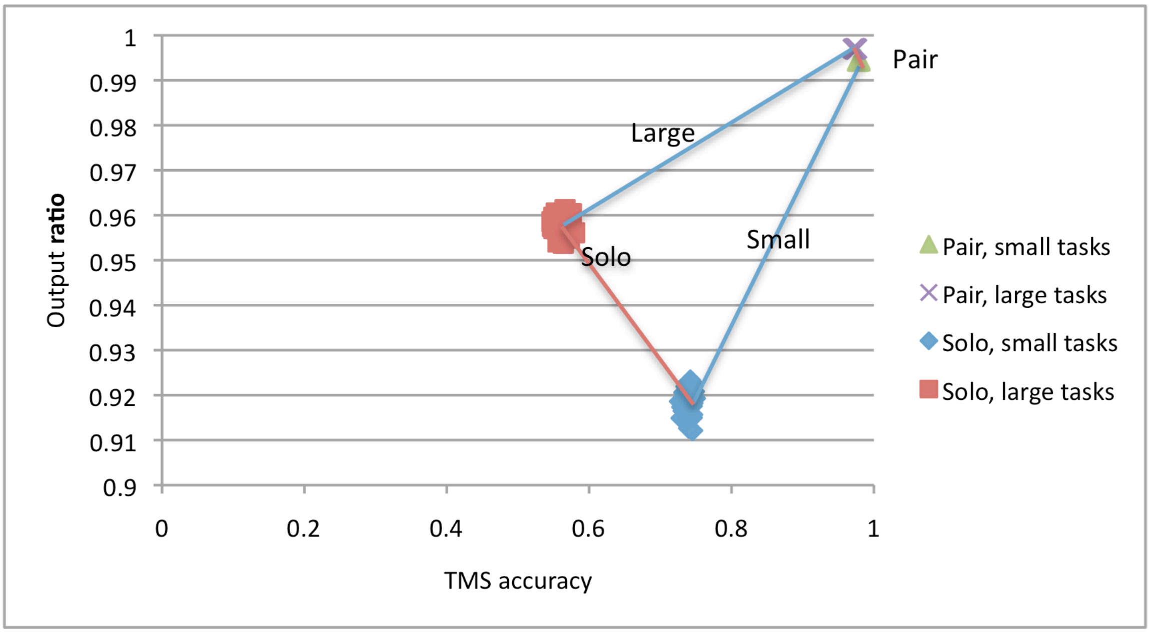


Figure 28: Novelty: Performance by TMS accuracy.

Hypotheses 5 predicts that for XP practices differentiation will be lower and output higher. This is depicted by Figure 29, which shows that differentiation is similar for all conditions other than solo large tasks which show lower differentiation. The blue lines show the interaction effects of task size and the red lines show the interaction effects of the work mode, showing some support for hypotheses 5. However, from the graph showing differentiation over time, figure 25, it can be seen that the relative positions of differentiation differs over the course of the simulation.

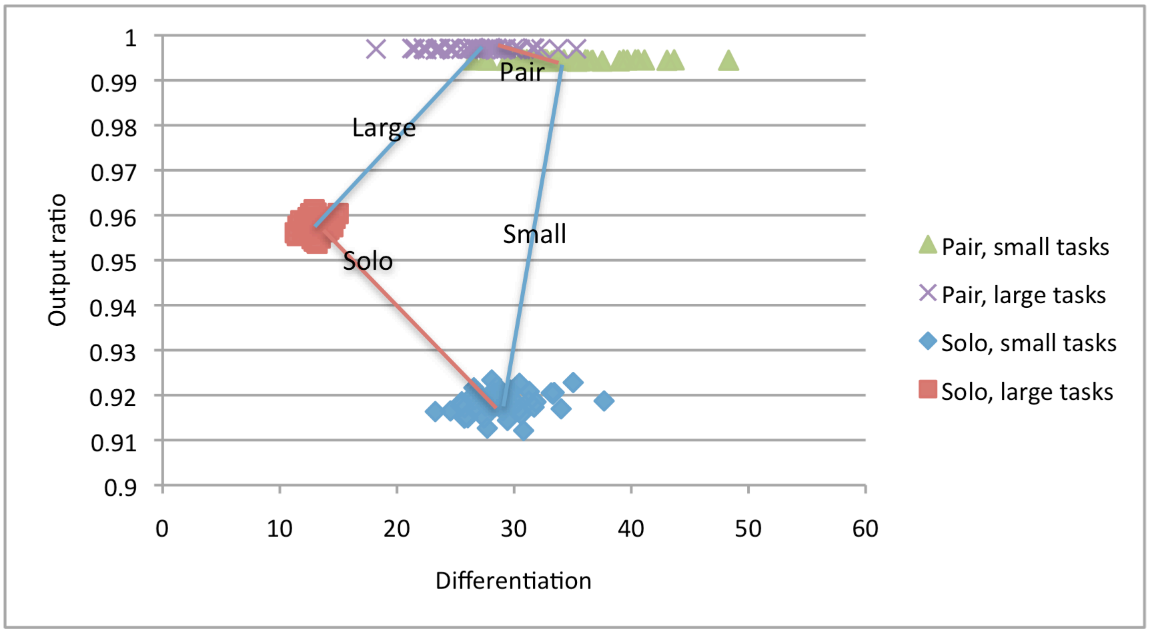


Figure 29: Novelty: Performance by differentiation.

Again contradicting hypothesis 5, Figure 30 depicts TMS accuracy against differentiation. The blue lines show the interaction effects of task size and the red lines show the interaction effects of the work mode. It shows that pair programming results in higher TMS accuracy than solo programming but differentiation is similar for all conditions other than solo, large tasks. Also, it can be seen that TMS accuracy is more affected by task size for solo programming than for pair programming.

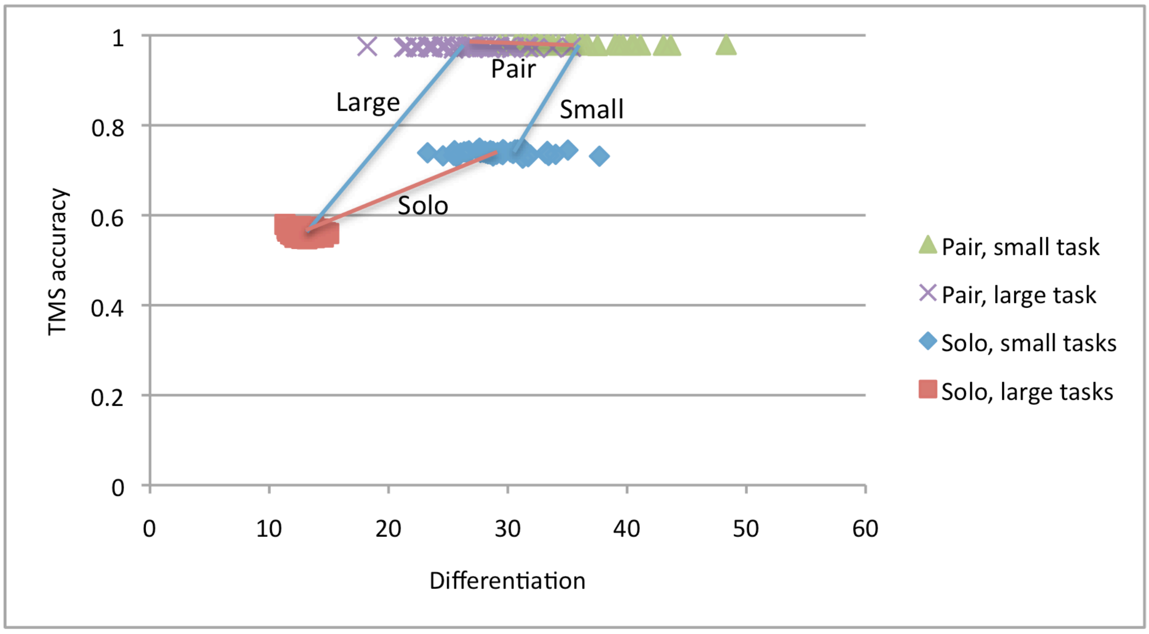


Figure 30: Novelty: TMS accuracy by differentiation.

10.5 Discussion

Software development projects often comprise high levels of innovation presenting tasks with high complexity, novelty and uncertainty. In comparison with routine, repetitive tasks it would be expected that general performance would drop and more collaboration, distribution of knowledge and team cognition would be demanded of the team. The model described in this chapter increased the range of knowledge in the model from 10 to 100 to simulate more innovative tasks so all tasks could contain numbers from 1 to 100 and that the knowledge that an agent could develop will range from 1 to 100.

The questions for this model related to changes in behaviour of TMS accuracy, differentiation and output as a result of introducing greater complexity and novelty to the tasks. The literature suggests that although output will be reduced due to greater task novelty and complexity the reduction will be smaller for pair programming and small task conditions. It is also hypothesised that differentiation will be reduced with a much larger knowledge range. So contrary to TMS literature, it was hypothesised that the TMS will remain important resulting in a negative relationship between TMS accuracy and differentiation resulting in higher output under pair programming and small task conditions.

Hypothesis 1 predicts that output will be generally lower with higher novelty and complexity, although hypothesis 3 suggests that this reduction will be reduced for pair conditions. Hypothesis 2 predicts that, pair programming and small task conditions will result in significantly higher output and hypotheses 4 and 5 predict that with greater novelty and complexity pair programming and task size conditions will affect the relationship between TMS accuracy and differentiation respectively with output. Specifically that differentiation will be low and TMS accuracy high for high output under pair programming and small task conditions.

As expected, supporting hypothesis 1, the overall performance for all conditions was lower than in the previous model, Tabula Rasa (Argote et al., 1995, Espinosa et al., 2007a). From the model point of view this is due to the tasks containing more unfamiliar elements, therefore the agent has no related knowledge to contribute to the success of the task. From a theoretical perspective this makes sense as greater novelty and complexity of tasks will demand greater resources, demand knowledge that the team member does not have, and increase team interdependency to complete the unfamiliar task.

Pair programming is more successful for complex tasks (Arisholm et al., 2007) than for simple routine tasks. For this reason it is expected that the reduction in output is attenuated by pair programming and small task practices and it can be clearly seen in Figure 17 that the output drop for pair programming is indeed smaller than the output drop for solo programming. It is posited that pair programming reduces the drop in performance because by working together agents are pooling resources to address unfamiliar tasks. By working together there is more chance of having useful knowledge for the fulfilment of the task and they are both increasing their knowledge which is distributed more evenly around the team for use in future tasks.

There is contradictory evidence in the literature relating to the role of TMS for more novel or complex tasks. One view states that if the work of the team is novel and requires less established knowledge then damage to the TMS will have less effect. So higher complexity and novelty will require lower dependence on TMS (Argote et al., 1995, Espinosa et al., 2007a). The contradictory view posits that complex tasks require greater interdependency and thus the impact of the TMS on performance is increased with increased task complexity (Akgun et al., 2005). Related to this question is the role of the relationship between TMS accuracy and differentiation and the subsequent impact on performance. Traditional TMS research states that there is a positive relationship between TMS accuracy and differentiation as with very low or non-existent levels of differentiation there will be no requirement for interaction if all members have the same knowledge (Klimoski and Mohammed, 1994). Less differentiated knowledge will inhibit the need for team members to work together thus rendering the TMS less useful (Brandon and Hollingshead, 2004). This model found that with a slightly reduced performance for more novel and complex tasks, the TMS was correspondingly slightly lower for all conditions but the differentiation was substantially lower for all conditions. The shape of the graph would indicate that the differentiation curves would develop in a similar way to the previous routine graph but would take a lot longer to do so. This is plausible as knowledge within the team and consequently the differentiation is developing over a much larger range of knowledge than for routine tasks.

Research into XP practices has found that using collective ownership weakens the relationship between TMS accuracy and performance (Maruping et al., 2009). It could be argued that pair programming has the same effect on the TMS/performance relationship but this simulation demonstrated that the importance of the TMS is maintained with high output.

The previous simulation, Tabula Rasa, using a small range of knowledge representing routine tasks found low differentiation for pair programming conditions the results of this model simulating more novel complex tasks found that for pair programming conditions differentiation was higher than for solo conditions. However, the graph would indicate that the differentiation for solo programming and small tasks would increase exponentially over time, overtaking pair programming conditions. By increasing the novelty and complexity, the differentiation is delayed. By working together on tasks not only are team members providing a larger pool of resources to fulfil the task, they are gaining greater knowledge increasing their levels of specialism.

Regarding the TMS this model also found that high levels of TMS accuracy are maintained for pair programming conditions so in support of Akgun and in contradiction of Maruping et al. it would appear that complex tasks do require greater interdependency and so there is a positive relationship between TMS accuracy and output for increased task complexity (Akgun et al., 2005). By working closely together in pairs, it would be expected that team members can update their TMS more readily than if working solo.

However, the picture is different for solo, task size conditions, where for higher output, differentiation and TMS accuracy are lower in large task conditions. This contradicts the view of Littlepage et al (2008) who proposed that small tasks and higher differentiation would result in higher performance, however this study was carried out on staff doing relatively routine work and the findings may not apply in an innovative environment such as software development where tasks are often more novel, complex and non-routine. The results of this simulation would seem to agree with this, supporting other literature (Brandon and Hollingshead, 2004, Espinosa et al., 2007b). In novel environments, for solo programmers large tasks are lowering differentiation by generating increased expertise across a much larger range for each task, than with small tasks that are very focussed. For the same reason TMS accuracy is reduced with large tasks as the effect is much more general and less focussed.

10.6 Conclusion

In summary, increasing the novelty and complexity in the model has reduced output in general but that reduction is smaller for pair programming conditions.

This model has found a discrepancy between work mode and task size. Whereas, pair programming increases output with higher differentiation and higher TMS accuracy, large tasks are also found to increase output but with lower differentiation and lower TMS accuracy. It was also found that task size has less effect in pair programming conditions and a larger effect in solo conditions.

This model also found that the development of differentiation is delayed for more novel and complex tasks resulting in the differentiation although being significantly lower than for routine tasks, pair programming conditions produces higher differentiation than solo conditions.

Both this model and the previous model have operated with no initial knowledge or TMS. This was useful to establish some baseline behaviours and to look at some parameters without the additional complication of existing knowledge and TMS. It is, however, somewhat unrealistic. So in order to include some additional realism and to investigate the behaviour of teams under different knowledge levels the next model will introduce knowledge and TMS.

Chapter 11: Model Differentiation

11.1 Introduction

Previous chapters have described simulations using agents with no initial knowledge or TMS, which although generating some relevant results could be said to be unrealistic. This simulation introduces agents with existing knowledge and TMS profiles that will give some insight into how initial conditions affect the knowledge, TMS and output of the team over time for different software development methodology techniques.

11.2 Theoretical framework

Inevitably there is some disagreement regarding the impact that differentiation of knowledge has on performance within a team. Diversity in a team directly impedes performance (Ancona and Caldwell, 1992, Smith-Jentsch et al., 2009) and while it does have some advantages in terms of internal processes and communication, overall the effect of diversity on performance is negative. It is suggested that diversity lowers the capability for coordination and cooperation within the team and this improves for more homogenous teams.

It may be the type of work that the team is undertaking that determines which type of knowledge structure is optimum for performance. Gupta and Hollingshead (2010) found evidence to suggest that knowledge structures with low differentiation resulted in higher performance for intellectual tasks.

In contrast an alternative opinion states that group performance is higher when members differ in ability. Highly differentiated groups are more successful at identifying and utilising expertise in the group therefore the advantages that TMS bestow are greater when group members' knowledge is more diverse (Baumann and Bonner, 2004, Littlepage et al., 2008). This supports the general literature on TMS - that the reliance on TMS is increased with greater differentiation because cognitive interdependence is elevated. However, this is not necessarily the case as it has been shown that in traditional software development projects interaction declined over the course of the project as team members' roles became more specialised (Levesque et al., 2001). It is suggested that the division of labour due to specialisation exacerbates this differentiation resulting in a reduced reliance on TMS. In the software development industry this is often the case with new teams being formed for each project, however, this may not be the case for teams that have longevity as they will continue to slowly develop a TMS even with reduced cognitive interdependence (Levesque et al., 2001).

Often teams are constructed from members with a diverse range of skills to leverage the unique expertise of different members. Performance will depend on each team member's contribution of knowledge to the team and the formation of a successful TMS. Expectations within the team relating to individual expertise will have an impact on TMS development and team members tend to learn more in their own specialisation if they believe that others hold different knowledge. They will often take responsibility for a particular area of expertise in the team. Likewise, other team members will defer to an individual who is perceived as the expert in a particular area (Hollingshead, 2001, Wegner, 1995). So the extent to which a team's knowledge is initially distributed is likely to define the initial structure of the TMS, and this should encourage information sharing leading to its further development (Lewis, 2004). Initially differentiated expertise is positively related to TMS emergence suggesting that more differentiated expertise helps define an initial framework of member-expertise associations. Teams with initially differentiated expertise were better at developing a TMS than those with overlapping expertise and this was even stronger when teams had initial TMS. For teams with overlapping expertise, prior knowledge of each other hindered TMS production suggesting that initial knowledge not initial TMS is responsible for the development of TMS (Lewis, 2004).

It is widely believed that a group with initial knowledge of each other, or a partially or fully developed TMS will have higher performance than a newly formed group where the members have no prior knowledge of each other, or no TMS (Akgun et al., 2005, Espinosa et al., 2007a, Lewis, 2004, Liang et al., 1995). For example studies have shown that teams that are trained together, and thus develop a TMS, perform better than those that are trained individually (Liang et al., 1995). This makes sense intuitively as the coordination and cooperation within a team would be improved if the team members have some knowledge of each other.

The development of differentiation of knowledge in a team is directly motivated by the outcome (Hollingshead, 2001). It has been found that group members tend to learn relatively more information in their own area of expertise when they believe their teammates have different expertise (Hollingshead, 2000). However, in circumstances where the outcome favours lower differentiation team members are more likely to learn in areas in which their colleagues are expert resulting in more integrated knowledge distribution. This strategy relies on an accurate TMS but as this develops the strategy is likely to become more successful. Much of the literature on TMS asserts that more differentiation is more beneficial to teams (Austin, 2003, Wegner, 1995) but Hollingshead suggests that there may be circumstances where differentiation may hinder group performance. From the results of the previous chapters it would seem that differentiation affects teams under different conditions.

It was stated in the previous chapter that pair programming works well when the pair has to work on a challenging problem and a study found that novice-novice pairs are much more productive against novice solos in terms of elapsed time and software quality than expert-expert pairs against expert solos (Lui and Chan, 2006). This agrees with a meta-analysis of papers on the effects of pair versus solo programming which considered the relationships between juniors, intermediate and senior pairs (Hannay et al., 2009). It found that the improvement in quality was most significant for juniors, with an overall 73% increase in correctness, which rose to 149% for complex tasks only. For intermediates and seniors the increase in correctness was 28% and 4% respectively furthermore these groups experienced decreases in correctness of 29% and 13% respectively for simple tasks. This study also found that pairing up juniors resulted in elevated performance, to near senior performance, suggesting that pair programming is most beneficial for novice programmers. This may provide an explanation for the success of pair programming in previous models as all the agents were modelled as novices with no initial knowledge.

In a study looking at pairs with different educational backgrounds and hence skill sets, forming pairs with similar skills enhanced the pair programming benefits (Bellini et al., 2005). However, pairs formed with very different backgrounds and skill sets reduces the more skilled member to a lower level. This contradicts Hannay et al. above and much existing work into the benefits of pair programming (Hannay et al., 2009, Muller, 2005, Muller, 2007, Nosek, 1998, Williams et al., 2000, Williams, 2000).

The TMS literature has conflicting views relating to differentiation in a team, and the resulting TMS and performance. As found in previous chapters the relationship between differentiation, TMS accuracy and output do not necessarily agree with the traditional TMS literature with differing levels of differentiation and TMS producing higher output under different conditions.

This model will examine the behaviour of output, TMS accuracy and differentiation for initial conditions relating to a variety of levels of initial knowledge and TMS. It will model teams containing all experts with the same knowledge and all experts with diverse knowledge. For both of those scenarios there will be a condition each for no initial TMS and complete TMS. As before, the model will have pair programming versus solo programming conditions and small task versus large tasks conditions.

Of this model the general results will be analysed for unanticipated behaviour, and in addition the following hypotheses are put forward.

As Lewis (2004) proposed it is the differentiation not the presence of an initial TMS that influences the development of TMS. This in turn will have an influence on output, which, it is expected, will be moderated by pair programming. It is also expected that pair programming conditions will be more resilient to the effects of initial TMS and initial differentiation conditions.

Hypothesis 1. Initial differentiation will have different effects on TMS accuracy and differentiation for different working conditions. It is expected that initial differentiation will have a positive relationship with TMS accuracy and with differentiation. Meaning that high initial differentiation produces high TMS accuracy and high differentiation. This effect will be weaker for pair programming and small task conditions.

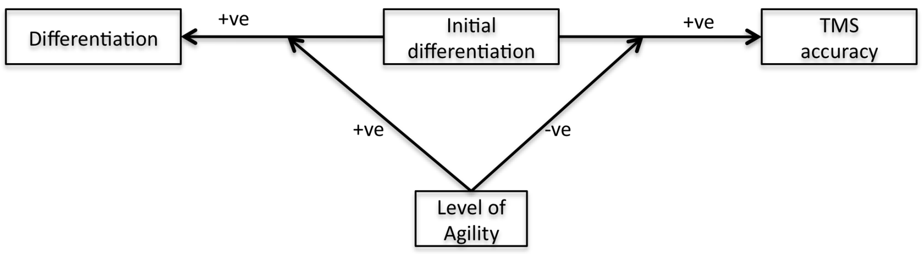


Figure 31: Differentiation: Diagram of hypothesis 1 relationship

Hypothesis 2. Initial TMS will have different effects on TMS accuracy and differentiation for different working conditions. It is expected that initial TMS will have a negative relationship with TMS accuracy and differentiation, with the presence of an initial TMS producing lower TMS accuracy and differentiation. This effect will be weaker for pair programming and small task conditions.

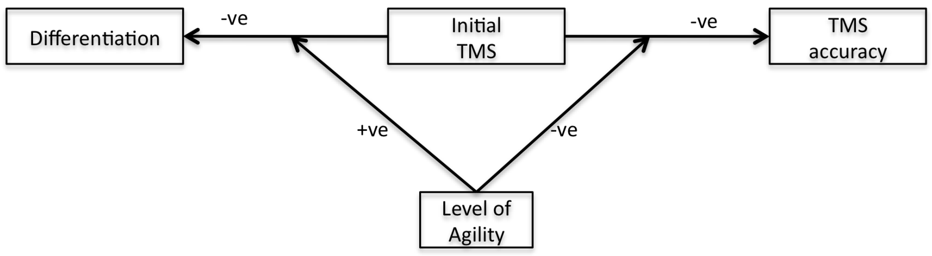


Figure 32: Differentiation: Diagram of hypothesis 2 relationship

Hypothesis 3. Work mode and task size will moderate the relationship between initial TMS and output. It is expected that for pair programming conditions, the presence or not of an initial TMS will have no effect on output but for solo conditions the existence of an initial TMS will have a positive effect on output.

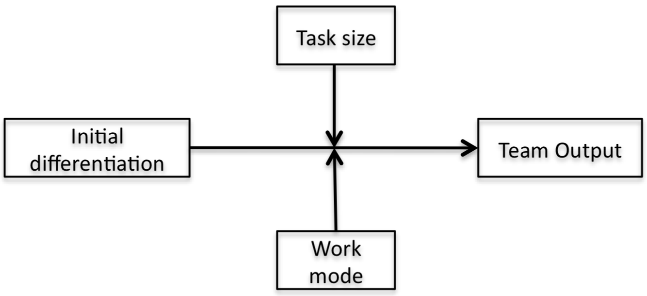


Figure 33: Differentiation: Diagram of hypothesis 3 relationship

Hypothesis 4. Work mode and task size will moderate the relationship between initial differentiation and output. As seen in previous chapters, lower initial differentiation will result in higher output for pair programming conditions and lower output for solo conditions.

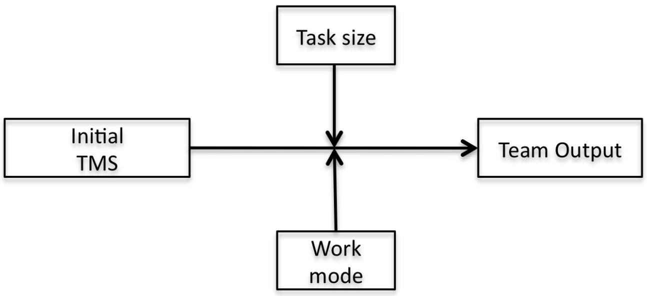


Figure 34: Differentiation: Diagram of hypothesis 4 relationship

11.3 Method

In addition to the existing variables of work mode and task size new variables have been added to this model to look at the effects of high initial differentiation or low initial differentiation for the agents.

For the high differentiation condition each of the four agents was furnished with knowledge value of 10 for discrete 25% elements of the knowledge range such that there was no overlapping knowledge. For the low differentiation condition, each of the agents was furnished with a knowledge value of 10 in the same 25% range of knowledge such that there was 100% overlapping knowledge.

Each of these conditions is also tested with no initial TMS and a complete and accurate TMS to establish the effects of the presence of a TMS or not.

The set of conditions for this model is detailed in the following table.

Number of agents	4
Task size	4 or 20
Work mode	Solo or in pairs
Knowledge range	1 to 100
Initial knowledge	4 agents with same 25% of knowledge or 4 agents with discrete 25% each of knowledge
Initial transactive memory	None or complete

Table 14: Differentiation: Parameters for model

Condition no.	Task size	Work mode	Initial TMS	Initial differentiation	No. of iterations
1	4	Pair	TMS	High	2000
2	4	Pair	TMS	Low	2000
3	4	Pair	No TMS	High	2000
4	4	Pair	No TMS	Low	2000
5	4	Solo	TMS	High	1000
6	4	Solo	TMS	Low	1000
7	4	Solo	No TMS	High	1000
8	4	Solo	No TMS	Low	1000
9	20	Pair	TMS	High	400
10	20	Pair	TMS	Low	400
11	20	Pair	No TMS	High	400
12	20	Pair	No TMS	Low	400
13	20	Solo	TMS	High	200
14	20	Solo	TMS	Low	200
15	20	Solo	No TMS	High	200
16	20	Solo	No TMS	Low	200

Table 15: Differentiation: Description of conditions

Each condition was simulated 50 times.

The volume of work delivered for each simulation defined the number of iterations for each condition. For each condition 16000 task elements were delivered to the team.

11.4 Results

This model introduces greater cognitive realism by giving the agents some initial knowledge and in some cases a complete TMS. Due to the results from previous chapters it is expected that performance for pair programming will remain high but the behaviour of the model with more realistic profiles will aid understanding of how initial conditions affect team performance.

Table 16 provides the descriptive statistics and the correlations among the constructs in the model giving some preliminary overall indications of the behaviour of the model. Of course these results give an overall view and do not indicate how the model behaves for individual conditions. It shows that working mode (coded 0 for solo programming; 1 for pair programming) is again positively, significantly correlated to output indicating that pair programming is highly positively correlated with output as expected. As previously, task size (coded 0 for large tasks; 1 for small tasks) is significantly positively correlated with differentiation and TMS accuracy and significantly negatively correlated with output, indicating that for large tasks produce better performance with lower TMS accuracy and differentiation.

Differing from the previous model the relationships between differentiation, TMS accuracy and output show that overall performance has a significant negative relationship with differentiation and a positive one with TMS accuracy resulting in a negative relationship between TMS accuracy and differentiation.

Regarding the new factors of initial TMS and initial differentiation; Initial TMS has no significant relationships with TMS accuracy, differentiation or output, lending some support to hypotheses 2 and 3. Initial differentiation has no significant relationship with TMS accuracy but it does have a positive significant relationship with differentiation and a significant negative one with output. This indicates that low initial differentiation encourages higher output and lower differentiation, supporting hypothesis 1 and giving some support to hypothesis 4.

	Variable	Mean	SD	1		2		3		4		5		6	
1	Working mode														
2	Task size														
3	TMS or not														
4	Initial differentiation														
5	Differentiation	52.44	38.24	-.325	**	.459	**	.057		.237	**				
6	TMS accuracy	.82	.17	.922	**	.274	**	-.041		-.005		-.090	*		
7	Performance	.97	.03	.923	**	-.126	**	.044		-.170	**	-.385	**	.793	**

Notes: n =800, *p<0.05, **p<0.01, n = number of runs of the simulation. 50 runs for each of the 16 conditions

Table 16: Differentiation: Descriptive statistics and correlations

To review the model results in more detail the following graphs show the behaviour of the model over time. There are four graphs for each of output, differentiation and TMS accuracy, showing results for conditions with high and low initial differentiation and the presence of an initial TMS or not. Each graph shows results for the four working conditions for task size and work mode.

The following four graphs show the model behaviour over time for output. Error bars show standard error of the mean (SEM) where:

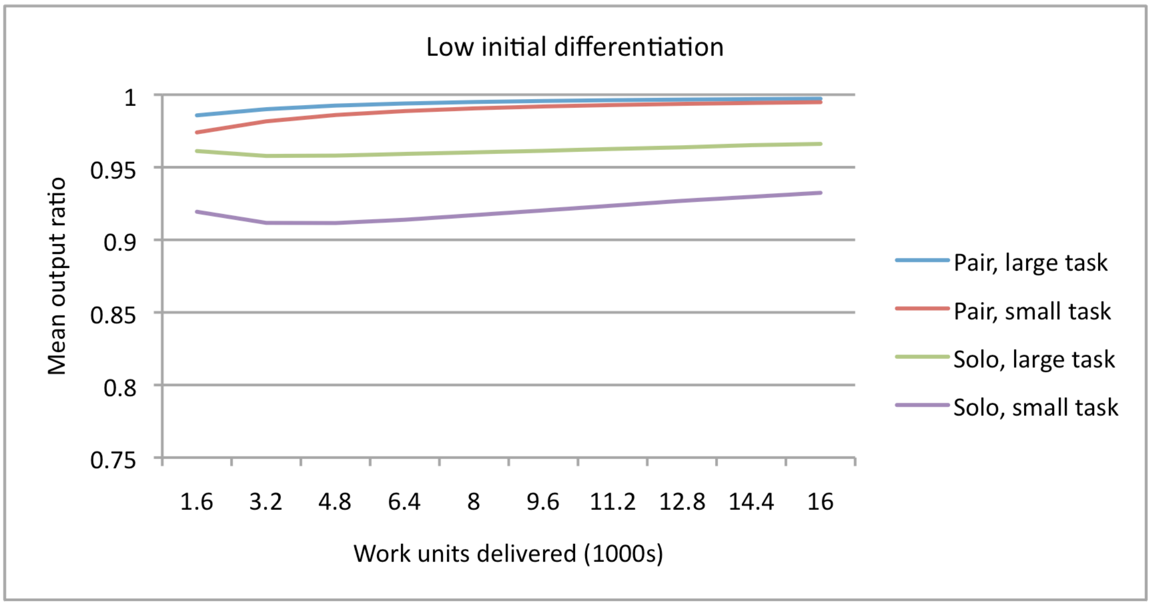
$$SEM = \sigma / \sqrt{n}$$

where σ is standard deviation and n is sample size. Where error bars are not apparent on the graphs, they are too small to be visible.

For pair programming the existence of an initial TMS seems to have little effect on output supporting hypothesis 3. For solo conditions the presence of an initial TMS has little effect for conditions with low initial differentiation, but for conditions with high initial differentiation the presence of an initial TMS produced substantially higher output, also lending support to hypothesis 3.

For pair programming conditions low initial differentiation was expected to have a positive effect on output but this was not seen; the level of initial differentiation had little effect on output. For solo conditions, however, for conditions of no initial TMS low initial differentiation generated higher output than high initial differentiation. When an initial TMS was present the difference in output was very similar but contrary to all other simulations, small tasks had the advantage for high initial differentiation. These results show no support for hypothesis 4.

Graphs for output over time



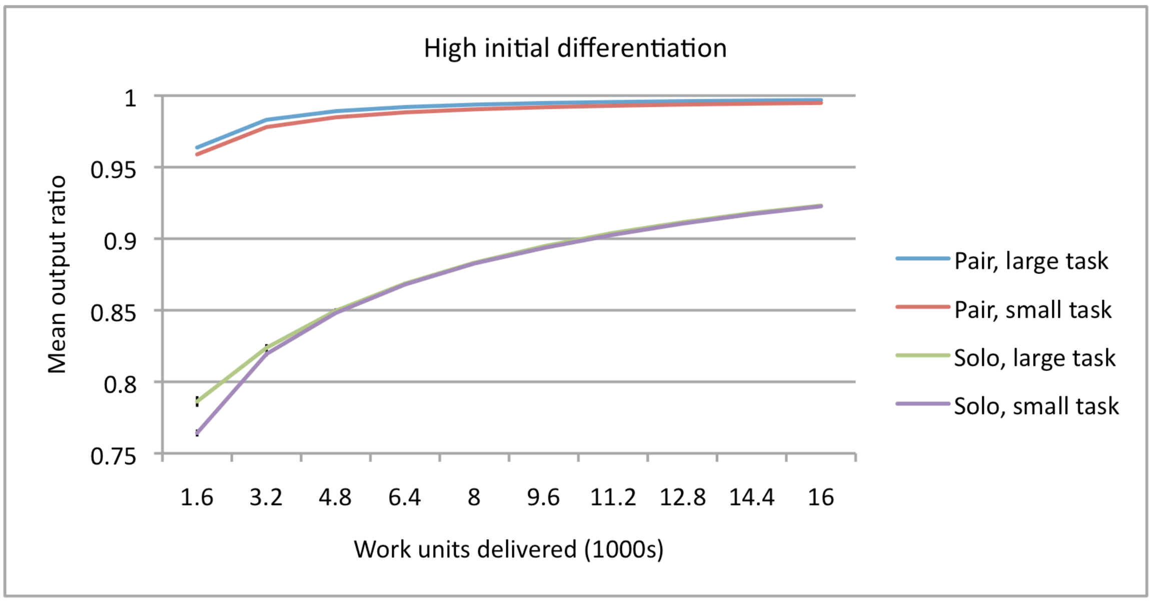
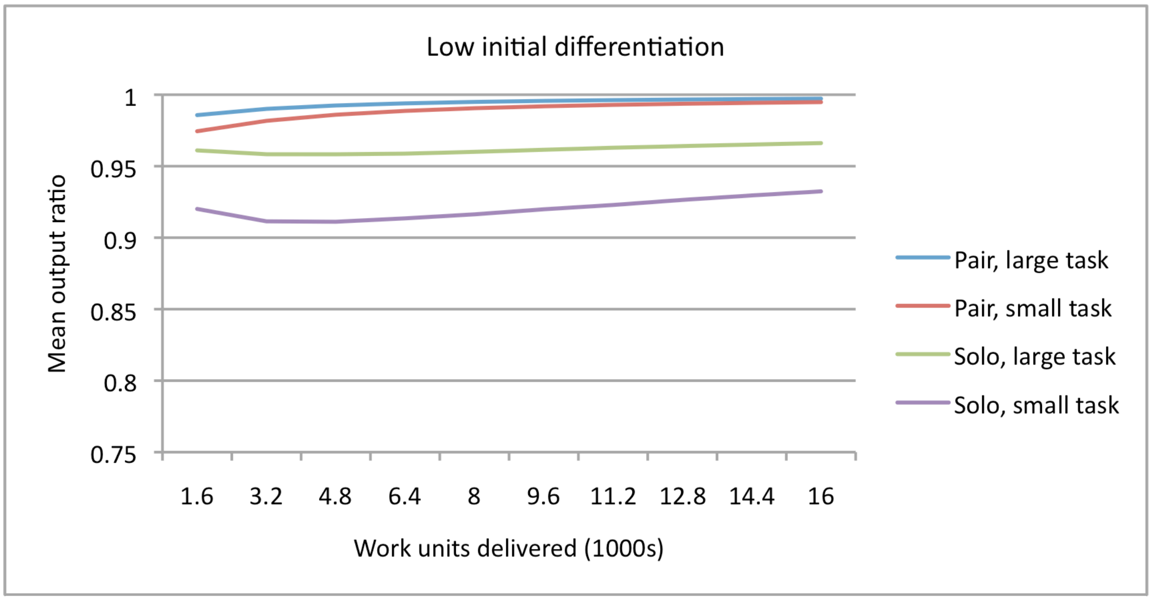


Figure 35: Differentiation: Output, no initial TMS



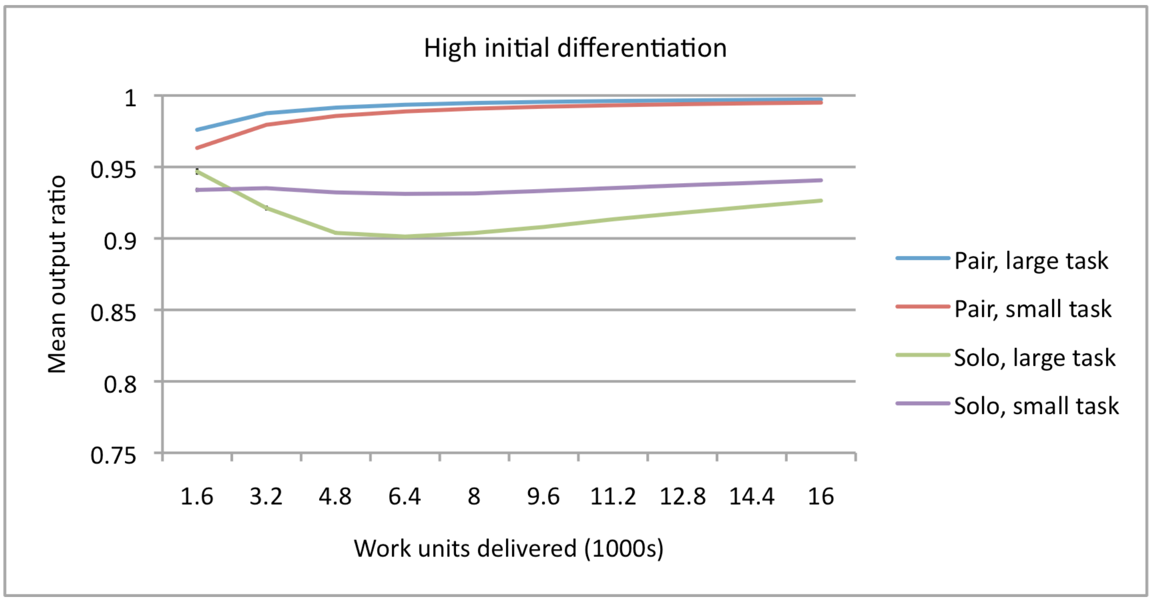
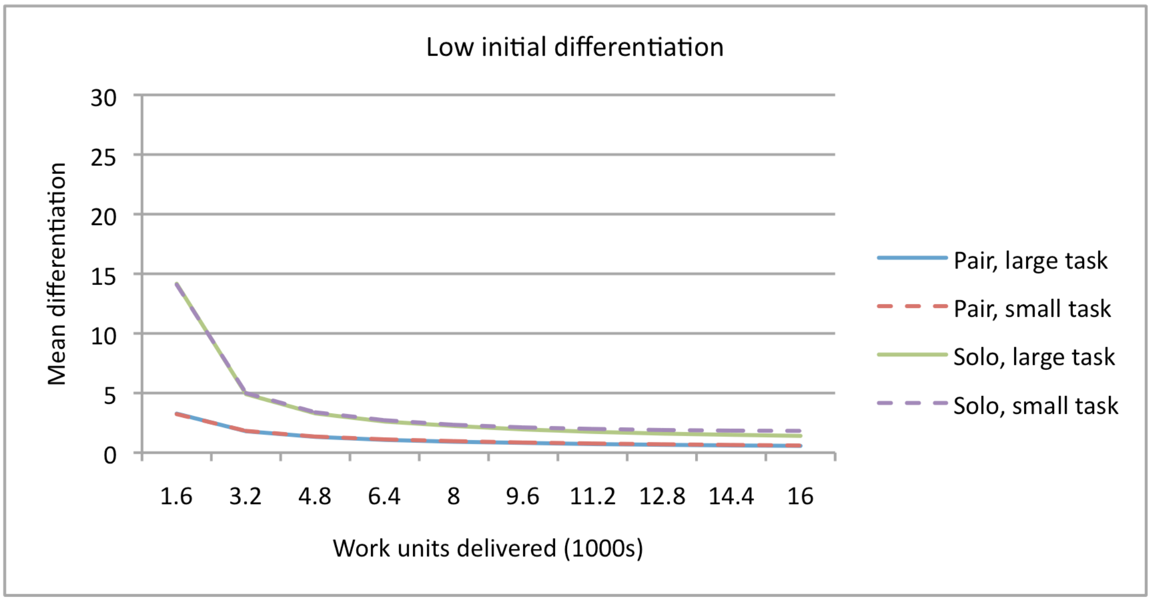


Figure 36: Differentiation: Output, a complete initial TMS.

The following graphs show the model behaviour over time for differentiation. For pair programming the existence of an initial TMS seems to have little effect on differentiation supporting hypothesis 2. For solo conditions, contradicting hypothesis 2, the presence of an initial TMS produces slightly higher differentiation for conditions of both high and low initial differentiation. For conditions of high initial differentiation there was a marked difference in differentiation by task size with small tasks producing higher differentiation than large tasks.

For pair programming conditions low initial differentiation was expected to have little effect on differentiation and this was seen; the level of initial differentiation had little effect on differentiation supporting hypothesis 1. For solo conditions, however, hypothesis 1 was supported. In the presence of an initial TMS or not, high initial differentiation resulted in higher differentiation over time. In addition in high initial differentiation conditions the effect of task size was marked with small tasks generating higher differentiation.

Graphs for differentiation over time



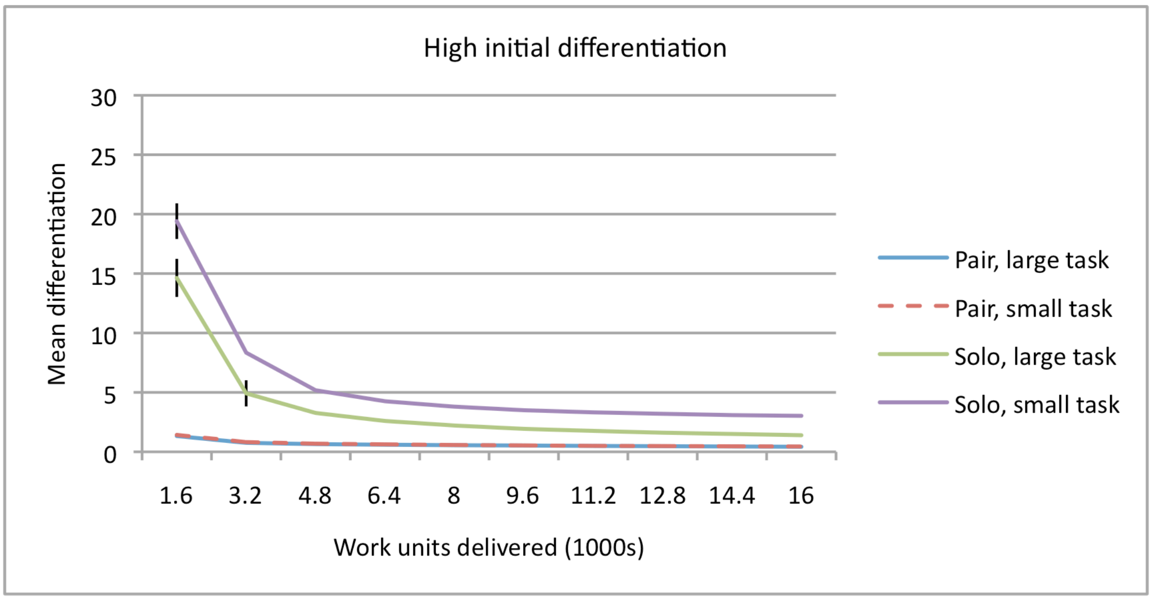
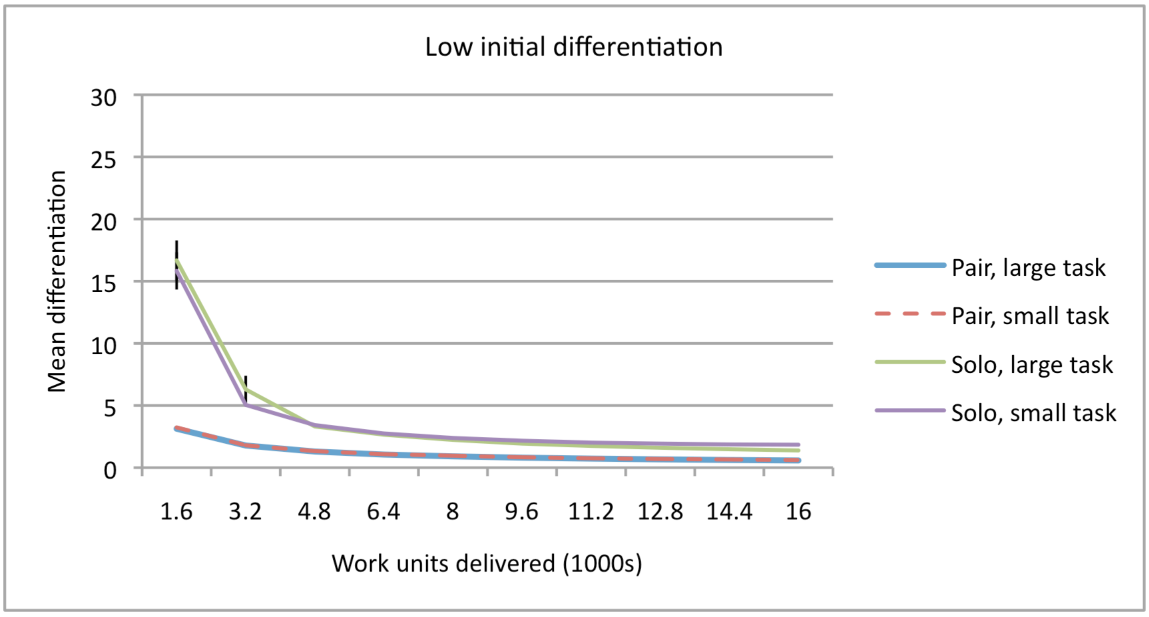


Figure 37: Differentiation: Differentiation, no initial TMS



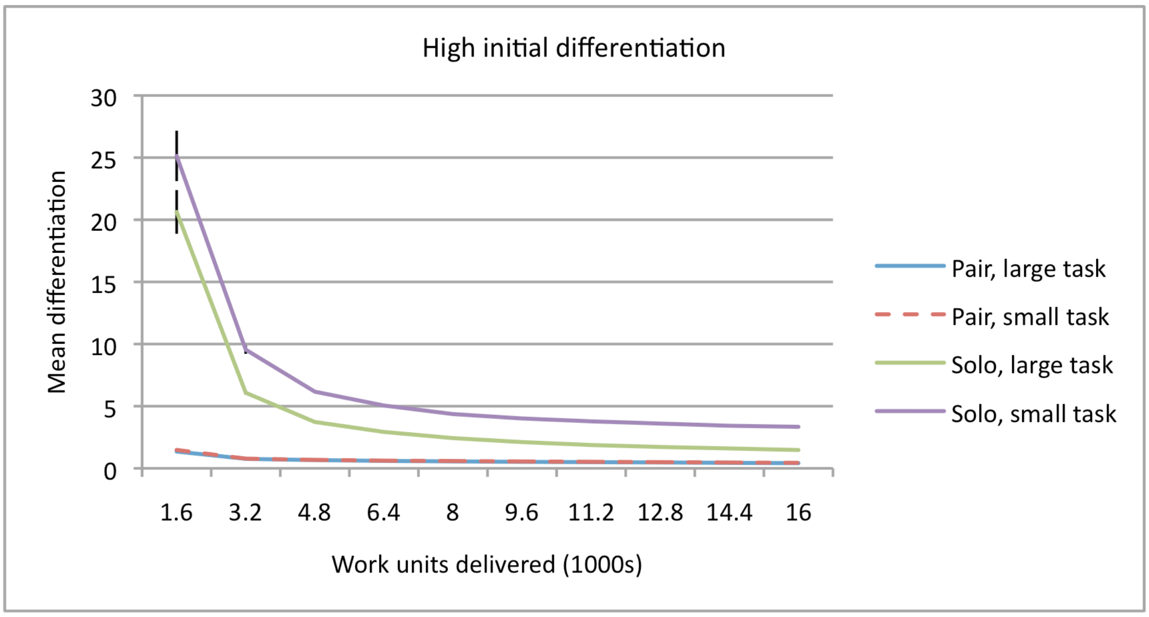


Figure 38: Differentiation: Differentiation, a complete initial TMS.

The following graphs show the model behaviour over time for TMS accuracy.

For pair programming the existence of an initial TMS seems to have little effect on TMS accuracy supporting hypothesis 2. For solo conditions, in support of hypothesis 2, the presence of an initial TMS produces lower TMS accuracy for conditions of both high and low initial differentiation, but only for large tasks.

For pair programming conditions low initial differentiation was expected to have little effect on differentiation and this was seen; the level of initial differentiation had little effect on differentiation supporting hypothesis 1. For solo conditions, however, hypothesis 1 was contradicted. The level of initial differentiation seemed to have no effect on levels of TMS accuracy.

Graphs for TMS accuracy over time

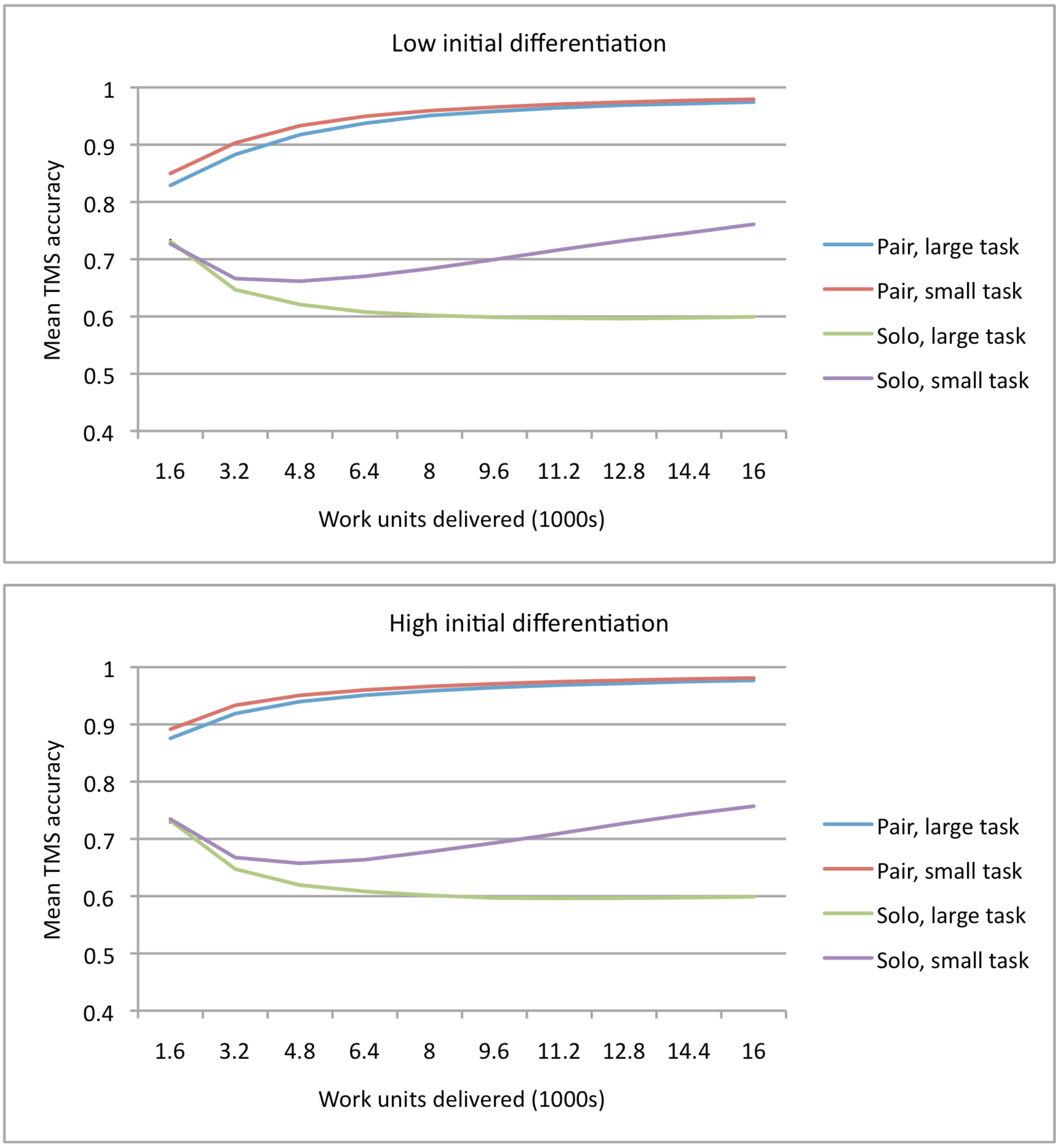
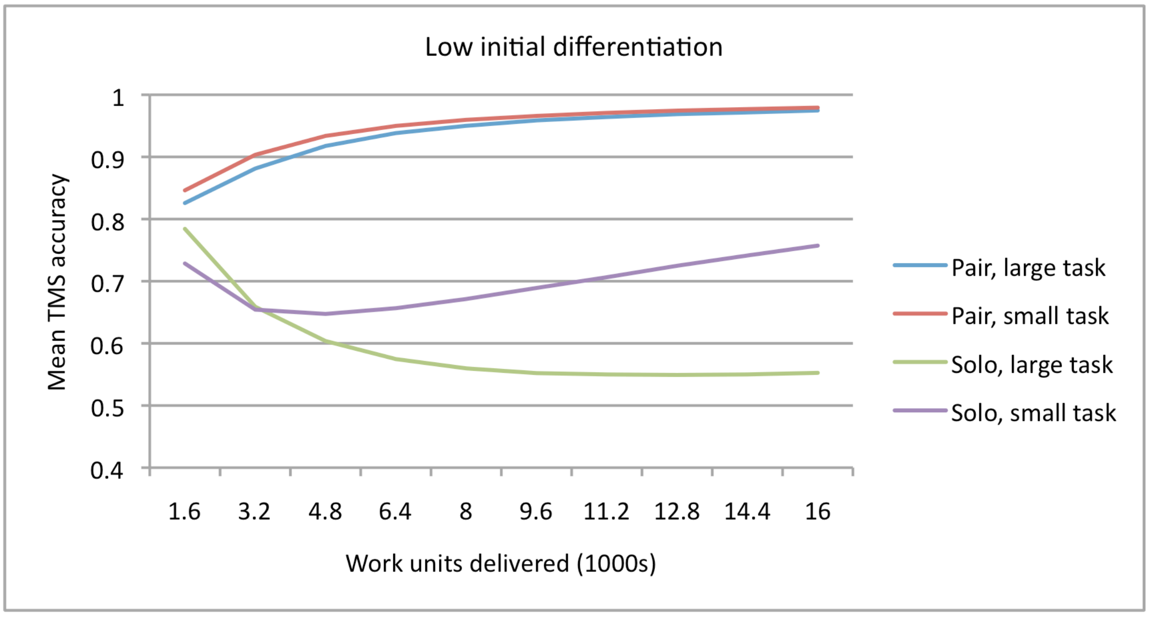


Figure 39: Differentiation: TMS accuracy, no initial TMS.



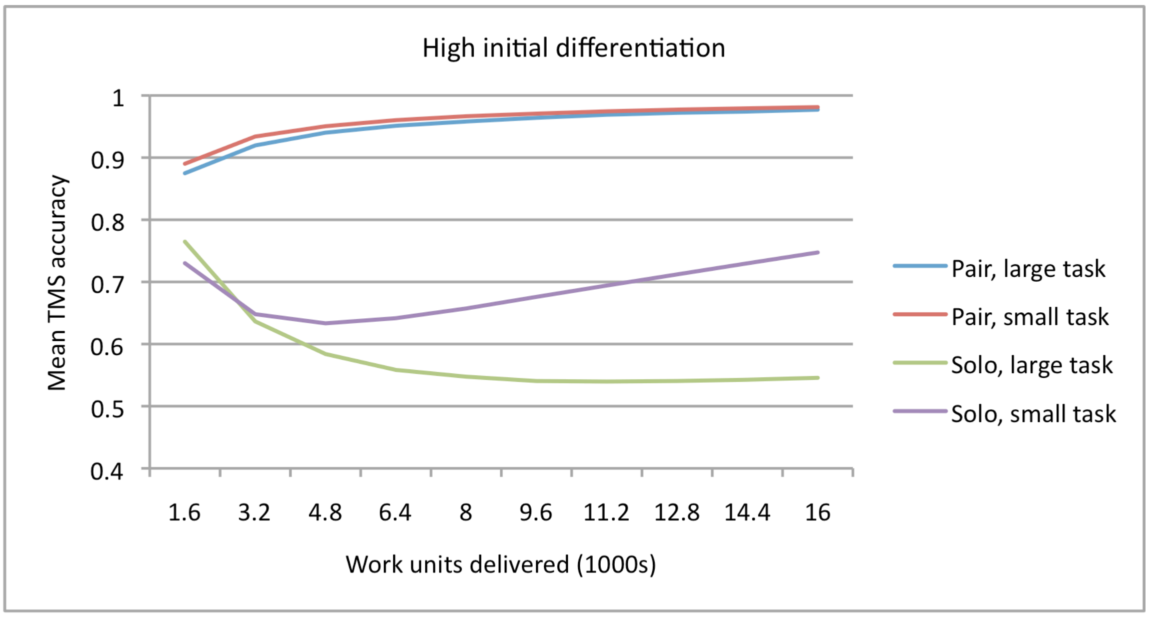


Figure 40: Differentiation: TMS accuracy, a complete initial TMS.

T-tests were carried out on the data for the final iteration to determine any significant differences in means. They were carried out separately on each set of conditions of working mode and task size to give an indication of the impact of initial differentiation for each scenario.

From the table below (Table 17) it can be seen that low initial differentiation generates higher output for every condition of working mode and task size apart from solo programming with small tasks.

Regarding differentiation over time, low initial differentiation generates higher differentiation over time for pair programming conditions and the opposite for solo conditions; high initial differentiation generates lower differentiation for solo conditions.

The t-tests determine that TMS accuracy is higher for pair programming when initial differentiation is high, but there is only significance for solo conditions when tasks are small when conversely TMS accuracy is higher for solo programming when initial differentiation is low.

In summary, low initial differentiation has a significant positive effect on performance for all conditions other than solo with small tasks. There is an inverse effect of initial differentiation on differentiation over time where high initial differentiation has a negative effect for pair programming conditions and a positive one for solo conditions. Finally high initial differentiation has a positive effect on TMS accuracy in pair programming conditions but has a negative effect on solo conditions with small tasks. There is no significant effect on solo conditions with large tasks.

These results lend partial support to hypotheses 1 and 4.

	Work mode	Task size	Initial differentiation	Mean	SE	*t*	df	
Output	Solo	4	High	0.931	9.5 E-4	0.77	104.21	
			Low	0.932	1.5 E-4			
		20	High	0.925	3.1 E-4	121.18	136.75	***
			Low	**0.966**	1.4 E-4			
	Pair	4	High	0.995	1.3 E-5	-7.61	164.36	***
			Low	**0.995**	8.2 E-6			
		20	High	0.997	2.2 E-5	7.79	139.31	***
			Low	**0.997**	1.0 E-5			
Differentiation	Solo	4	**High**	**149.81**	1.50	-70.30	105.13	***
			Low	42.50	0.26			
		20	**High**	**34.89**	0.32	-7.30	152.43	***
			Low	32.23	0.17			
	Pair	4	High	33.87	0.44	24.31	168.93	***
			Low	**53.73**	0.69			
		20	High	27.43	0.42	23.97	176.93	***
			Low	**45.06**	0.60			
TMS accuracy	Solo	4	High	0.752	7.0 E-4	8.30	166.90	***
			Low	**0.759**	4.4 E-4			
		20	High	0.572	2.8 E-3	.97	194.82	
			Low	0.576	2.5 E-3			
	Pair	4	**High**	**0.981**	8.2 E-5	-15.31	198	***
			Low	0.979	8.9 E-5			
		20	**High**	**0.977**	1.5 E-4	-9.84	198	***
			Low	0.975	1.8 E-4			

Notes: n=800, ***p<0.001, n = number of runs of the simulation. 50 runs for each of the 16 conditions

Table 17: Differentiation: Independent t-tests for initial differentiation, separated by working conditions.

The following table (Table 18) shows that the existence of an initial TMS results in higher performance for all working conditions apart from solo programming with large tasks when there is no significant difference. Regarding differentiation over time there is no effect attributed to the existence of an initial TMS for any condition apart from solo conditions with large tasks. For TMS accuracy, the presence of an initial TMS has a negative effect on TMS accuracy over time for solo programming conditions and no significant effect for pair programming conditions. These effects can be easily seen on the graphs above showing model behaviour over time.

Hypothesis 3 is supported for pair programming and for solo conditions small tasks have to be in use for the existence of an initial TMS to be significant. Also, hypothesis 2 unsupported as the existence of a TMS has a positive effect on output for pair programming and also for solo conditions with small tasks.

	Work mode	Task size	Initial TMS	Mean	SE	*t*	df	
Output	Solo	4	No TMS	0.927	5.4 E-4	-12.57	194.46	***
			TMS	**0.936**	4.7 E-4			
		20	No TMS	0.945	2.2 E-3	-0.57	196.77	
			TMS	0.976	2.0 E-3			
	Pair	4	No TMS	0.994	1.1 E-5	-3.19	187.90	**
			TMS	**0.995**	1.4 E-5			
		20	No TMS	0.997	2.2 E-5	-6.77	152.64	***
			TMS	**0.997**	1.2 E-5			
Differentiation	Solo	4	No TMS	89.00	4.71	-1.86	185.93	
			TMS	103.31	6.12			
		20	No TMS	32.28	0.18	-6.94	158.46	***
			TMS	**34.83**	0.32			
	Pair	4	No TMS	43.37	1.14	-.521	198	
			TMS	44.22	1.16			
		20	No TMS	36.41	1.08	-.226	198	
			TMS	36.08	0.97			
TMS accuracy	Solo	4	**No TMS**	**0.759**	4.6 E-4	8.11	172.42	***
			TMS	0.752	6.9 E-4			
		20	**No TMS**	**0.599**	8.7 E-4	41.53	198	***
			TMS	0.549	8.4 E-4			
	Pair	4	No TMS	0.980	1.3 E-4	-.366	198	
			TMS	0.980	1.2 E-4			
		20	No TMS	0.976	2.2 E-4	-1.01	198	
			TMS	0.976	1.8 E-4			

Notes: n=800, $^{**}p<0.01$, $^{***}p<0.001$, n = number of runs of the simulation. 50 runs for each of the 16 conditions

Table 18: Differentiation: Independent t-tests for initial TMS separated by working conditions.

11.5 Discussion

To include initial knowledge and TMS of agents improves the cognitive realism of the model. With the baseline models complete the impact of adding knowledge and TMS can be assessed. As described in the theoretical framework the behaviour of teams is likely to vary with different knowledge structures and the extent to which the initial knowledge in a team overlaps is one area that has been addressed in previous studies (Ancona and Caldwell, 1992, Austin, 2003, Baumann and Bonner, 2004, Gupta and Hollingshead, 2010, Hollingshead, 2000, Hollingshead, 2001, Lewis, 2004, Levesque et al., 2001, Littlepage et al., 2008, Wegner, 1995). Also, the initial development of the TMS is likely to have an influence on the performance of the team (Akgun et al., 2005, Espinosa et al., 2007a, Lewis, 2004, Liang et al., 1995).

This model introduced the dichotomous parameters of low and high initial differentiation and the presence of an initial TMS or not. Low initial differentiation was modelled by each agent having the same knowledge in the same 25% of the range of knowledge available. The high differentiation condition furnished the agents with the same level of knowledge but each agent's knowledge occupied a discrete 25% of the available knowledge, resulting in no overlapping initial knowledge at all. For those conditions where a TMS was present the TMS was complete and accurate.

The hypotheses presented for this model predicted that the impact of initial differentiation and initial TMS on performance, differentiation and TMS accuracy will be moderated by the XP factors of work mode and task size. The results showed that the hypotheses were partially supported.

As described in the theoretical framework there is continuing discussion relating to the role that differentiation plays in the performance of teams with one school of thought proposing that diversity in a team impedes performance (Ancona and Caldwell, 1992, Smith-Jentsch et al., 2009) and another suggesting that the advantages that TMS bestow are greater when group members' knowledge is more diverse (Baumann and Bonner, 2004, Littlepage et al., 2008).

Initially differentiated expertise is positively related to TMS emergence suggesting that more differentiated expertise helps define an initial framework of member-expertise associations. According to Lewis, teams with initially differentiated expertise were better at developing a TMS than those with overlapping expertise (Lewis, 2004). The initial differentiation in a team is likely to have an impact on the development of the TMS and the resulting team output, as team members will defer to an individual who is perceived as the expert in a particular area (Hollingshead, 2001, Wegner, 1995). However, as previous models have found, utilising pair programming can change the relationships that TMS accuracy and differentiation have with team output. In support of a number of studies (Ancona and Caldwell, 1992, Gupta and Hollingshead, 2010, Smith-Jentsch et al., 2009) the results of this model showed that low initial differentiation had a more positive effect on output than high initial differentiation apart from solo conditions with small tasks. Disagreeing with Lewis, the model found that, in support of hypothesis 4 for pair programming, low initial differentiation generated higher team output, with no support for solo conditions. Low initial differentiation also generated for pair programming, lower TMS accuracy and higher differentiation, partially supporting hypothesis 1.

The positive effect of low initial differentiation can be explained by the functioning of the model. Output is measured as the ratio between actual output and optimum output. As each of the agents has identical knowledge the potential detriment from allocating tasks to the wrong agent is reduced.

In addition, as TMS theory predicts (Akgun et al., 2005, Espinosa et al., 2007a, Lewis, 2004, Liang et al., 1995), results show that the existence of an initial TMS gives higher output for all working conditions apart from solo programming with large tasks when there is no significant difference. Hypothesis 3 is unsupported as the existence of an initial TMS significantly affects the output for all conditions apart from solo programming with large tasks. The advantage from the presence of an initial TMS is clear as tasks are directed to the most qualified agent for the task from the outset.

In terms of the theoretical framework, the model supports the theory that low initial differentiation predicts higher performance because diversity lowers the ability of a team to cooperate and coordinate (Ancona and Caldwell, 1992, Smith-Jentsch et al., 2009). Also, pairing programmers with similar knowledge was found to improve performance (Bellini et al., 2005), which may explain the additional advantage that pair programming had over solo programming for low initial differentiation in the model.

Regarding differentiation over time, the existence of an initial TMS had no effect under any condition other than for the case of solo programming with large tasks. Differentiation over time was found to have a significant negative relationship with initial differentiation for pair programming conditions and a significant positive relationship with initial differentiation for solo conditions.

Agents that work together will exchange knowledge that differs. In the case of low differentiation there is no difference in agents' knowledge, therefore, there will be little or no exchange of knowledge and each will retain their own knowledge. In the case of high differentiation as agents work in pairs there will be high levels of exchange of knowledge due to the high diversity, resulting in a flatter knowledge structure. In reality, if all team members have the same knowledge, so have little reason to exchange knowledge, low initial differentiation reduces the interdependence in the team. For high initial differentiation there is high interdependence and with pair programming this results in large amounts of knowledge sharing.

For pair programming TMS accuracy was not affected by initial TMS but for solo conditions the presence of an initial TMS actually reduced TMS accuracy over time. The former is not a surprise as the elevated sharing of knowledge also includes greater TMS accuracy meaning that the TMS 'gets up to speed' very quickly for pair programming conditions. The initial presence of a complete and accurate TMS reducing the TMS accuracy over time for solo programming is more difficult to explain as it seems counterintuitive. It is possible that as changes to knowledge happen in the team it is more difficult for agents using solo programming to maintain the complete accurate TMS than if they were building the TMS from scratch. This is contrary to the literature (Akgun et al., 2005, Espinosa et al., 2007a, Lewis, 2004, Liang et al., 1995) which states that groups with an initial TMS will have higher quality TMSs and perform better.

However, by reviewing the graphs of output it can be seen that regardless of initial differentiation and TMS conditions, for pair programming conditions, output is consistently higher, differentiation over time consistently lower and TMS accuracy consistently higher than for solo programming conditions. This is consistent with previous models and demonstrates that the use of pair programming mitigates the effects of initial conditions of a team on output and generates lower differentiation than the TMS literature predicts.

Also, regarding solo conditions, in conditions with low initial differentiation large tasks produce higher output and for high initial differentiation. For high initial differentiation, small tasks are only advantageous when there is an initial TMS. For all solo conditions small tasks produce higher TMS accuracy even though this does not necessarily translate to better output.

In summary, as with previous models pair programming improved output and TMS accuracy, and reduced differentiation over time for all conditions. Generally, lower initial differentiation and an initial TMS increased output. For pair programming initial TMS had no effect on TMS accuracy or differentiation over time; and low initial differentiation increased both TMS accuracy and differentiation over time. For solo programming the picture is a little different, initial TMS reduced TMS accuracy and only improved differentiation over time for large tasks; and initial differentiation increased differentiation over time, and reduced TMS accuracy but only for small tasks.

11.6 Conclusion

This final model introduced yet more cognitive realism by using agents that had existing knowledge and TMS with varying characteristics. Both heterogeneous and homogenous initial knowledge in the team was simulated, and the presence or absence of a complete TMS simulating team members that have prior knowledge, or not, of each other. The same comparisons of pair versus solo programming, and small versus large tasks were carried out to test how various initial conditions relating to knowledge and TMS affect the team in terms of TMS, differentiation and output over time.

The results showed that team members with similar knowledge, who had worked together before, would result in higher output. The use of pair programming compensates for different initial team conditions producing consistently higher output and higher TMS accuracy. Also, in contradiction of the TMS literature, the use of pair programming produces lower differentiation while maintaining the importance of an accurate TMS for better performance.

If pair programming, team members having worked together before, and so having initial knowledge of each other, is unlikely to have an effect on the accuracy of their TMS or knowledge diversity over time. In contrast homogenous initial knowledge is likely to have an effect on the accuracy of TMS and the diversity of knowledge in the team, increasing them both to a greater degree over time.

For solo programmers, previously working together reduces the accuracy of the team members' TMS over time and initial diversity of knowledge will increase the diversity over time. Other effects of initial conditions are dependent on other factors such as task size.

Software development teams are often newly formed for each project and as such team members may not have worked together before meaning a lack of TMS relating to each other. A lack of initial TMS has been shown to reduce output, however, this simulation has demonstrated that the use of pair programming techniques can mitigate this disadvantage.

Chapter 12: Discussion

12.1 Review

The main objective of this research was to develop an agent-based model of transactive memory systems and knowledge processes simulating software development teams operating within two different software development methodologies. EXtreme programming, as a more recent Agile methodology, has been compared with the more established Waterfall methodology in the literature but very little of that research has focussed on the cognitive aspects of team working. The aim was to contribute to the understanding of how knowledge distribution and transactive memory systems affect performance of software development teams under different conditions and to identify some optimum parameters such that software development teams can operate under different conditions for the best performance. In order to address these objectives, agent-based modelling was utilised and the modelling environment FLAME was used to build a model to address these questions.

There is a body of research relating to transactive memory systems and there is also research comparing different software development methodologies; there is research into the cognitive processes of software development teams and also agent-based modelling of social and cognitive systems. This research aimed to consolidate these discrete streams of research and take an agent-based modelling approach to investigate the behaviour of knowledge and transactive memory systems for both traditional and eXtreme programming conditions in software development teams. The model compared knowledge, TMS and performance for both methodologies in situations relating to routine versus novel tasks and combinations of knowledge distribution.

After initial validation and alignment exercises the first experiment simulated simple routine tasks and compared two aspects of XP. The two parameters compared were pair/solo programming and small/large task size, with measures of TMS accuracy and completeness, differentiation and output being taken. The literature relating to software development often emphasises that the nature of the work of software development teams is complex and novel so the second model developed the simulation further to increase the knowledge range. Introducing greater realism into the model had the effect of simulating greater complexity and novelty to the work the agents were asked to do. The first two models simulated agents with no initial knowledge or TMS thus modelling people with no knowledge and no knowledge of each other. To introduce further realism the third model portrayed agents with initial knowledge and for some conditions an existing TMS. The questions asked of this model were related to the distribution of the agents' initial knowledge and the presence of an existing TMS and their impact on output.

12.1.1 Model Tabula Rasa

The aims of this initial model were to compare work mode, pair or solo programming, and task size, small or large, and look at how they affected the measures of TMS, differentiation and team output in a simple simulation modelling routine tasks, and agents with no knowledge or TMS. It was hypothesised that work mode and task size would moderate the relationships of TMS accuracy and differentiation with performance. It was also predicted that TMS accuracy and differentiation would be lower for both pair programming and small task conditions. A further hypothesis stated that performance would be elevated for both small task and pair programming conditions.

The literature suggests that for optimum performance in traditional team environments the TMS develops hand in hand with differentiation with the emergence of specialists (Austin, 2003, Brandon and Hollingshead, 2004, Hollingshead, 2000, Wegner, 1995). The introduction of pair programming and small task practices, it was hypothesised, changes the way that TMS is utilised in a team and the mechanisms for the emergence of specialists, or in other words, differentiation. It was argued that for optimum performance in XP teams, differentiation and the reliance on TMS is reduced, as there is elevated communication and sharing of task work. Studies into other aspects of XP have found that knowing the location of expertise in a team becomes less important when there is collective ownership of code, which can substitute for expertise coordination mechanisms such as TMS (Maruping et al., 2009). There are conflicting views in the literature relating to task size (Espinosa et al., 2007a, Littlepage et al., 2008) so this model was of interest to give some indication of the role task size plays under different methodologies. There is a body of literature on the impact of pair programming on team performance and the results are mixed, however, it is clear that there are many other factors that influence the outcome of a team and pair programming can be of benefit under certain circumstances.

The results from this model suggested that in pair programming conditions, with high differentiation, TMS accuracy is important for performance. According to this model, pair programming generates high performance, high TMS accuracy and low differentiation, with large tasks having a positive effect on output and differentiation. Solo programming produces lower output with an advantage for large tasks, however solo programming generates lower accuracy and higher differentiation with an advantage for small tasks. The hypotheses were partly supported.

This model supported the hypothesis that pair programming generates low differentiation. With negative relationships being found between TMS accuracy and differentiation the results do not support the literature (Brandon and Hollingshead, 2004, Espinosa et al., 2007a, Klimoski and Mohammed, 1994) asserting that less differentiated knowledge inhibits team interdependence and reduces the reliance on the TMS.

12.1.2 Model Novelty

The previous experiment focussed on a simple initial model with a small range of knowledge simulating repetitive routine tasks. This model increased the knowledge range to between 1 and 100 to simulate greater complexity and novelty. The aim of this model was to simulate greater levels of complexity and novelty for the agents' tasks modelling the more innovative environment of a software development team (Hoegl et al., 2003). It was hypothesised that general output would be reduced, output would be higher for pair programming and small task conditions, but the reduction in output would be attenuated by the use of pair programming and small task size. Further hypotheses predicted that pair programming and small task size practices would affect the relationship between both TMS and differentiation, and output. It was predicted that this model would demonstrate that for pair programming differentiation would be lower.

The literature on TMS in highly innovative environments is contradictory (Akgun et al., 2005, Argote et al., 1995, Espinosa et al., 2007a, Hoegl et al., 2003, Ren et al., 2006) therefore the results of this model were difficult to predict. The model also demonstrated the influence of task size on TMS, differentiation and output.

The results of this model found that increasing the novelty and complexity in the model reduced output in general but as hypothesised, that reduction was smaller for pair programming conditions with pair programming conditions again producing higher output. It is suggested that pair programming reduces the drop in output because by working together agents are pooling resources and there is more chance of having useful knowledge for the fulfilment of the unfamiliar task.

It was also shown, as hypothesised, that for tasks with greater novelty and complexity, differentiation is lower for pair programming conditions than for solo conditions. By working together on tasks team members are sharing knowledge, which increases the distribution of knowledge around the team. As the pairs change regularly, this knowledge distribution is team wide.

Regarding the TMS this model also found that high levels of TMS accuracy are maintained for pair programming conditions so in support of Akgun it would appear that complex tasks do require greater interdependency and so there is a positive relationship between TMS accuracy and output for increased task complexity (Akgun et al., 2005).

This model has found a discrepancy between work mode and task size. Whereas, pair programming increases output with higher differentiation and higher TMS accuracy, large tasks are also found to increase output but with lower differentiation and lower TMS accuracy. It was also found that task size has less effect in pair programming conditions and a larger effect in solo conditions.

This model also found that the development of differentiation is delayed for more novel and complex tasks resulting in the differentiation although being significantly lower than for routine tasks, pair programming conditions produces higher differentiation than solo conditions.

In summary, this model demonstrates that the drop in performance due to complexity and novelty of tasks can be reduced by the utilisation of pair programming and that the development of differentiation is delayed due to a much larger knowledge base, potentially having an effect on the results of the team.

12.1.3 Model Differentiation

From the results of the previous chapters low differentiation is beneficial for teams using pair programming practices. While working in pairs not only is knowledge being distributed more evenly about the team resulting in lower differentiation but also, it may be more conducive to efficient pair working if the pairs have closer knowledge structures.

This model examined the behaviour of output, TMS accuracy and differentiation for initial conditions relating to a variety of levels of initial knowledge and TMS. It modelled teams containing all experts with the same knowledge and all experts with diverse knowledge. For both of those scenarios there was a condition each for no initial TMS and complete TMS. As before the model had pair programming versus solo programming conditions and small task versus large tasks conditions.

As found in previous chapters the relationship between differentiation, TMS accuracy and output do not necessarily agree with the traditional TMS literature with TMS retaining an importance for team output even when differentiation is low. For this reason it was expected that initial conditions of high differentiation would not produce high output in pair programming conditions although they might for solo conditions.

The hypotheses for this model predicted that initial TMS and differentiation would have different effects on TMS accuracy and differentiation for different working conditions. In addition it was predicted that pair programming and task size factors would moderate the relationship between initial TMS, initial differentiation and output. It was expected that for pair programming conditions, the presence or not of an initial TMS would have no effect on output but for solo conditions the existence of an initial TMS would have a positive effect on output. Also, lower initial differentiation would result in higher output for pair programming conditions and lower performance for solo conditions.

If pair programming, team members having worked together before, and so having initial knowledge of each other, is unlikely to have an effect on the accuracy of their TMS or knowledge diversity over time. In contrast homogenous initial knowledge is likely to have an effect on the accuracy of TMS and the diversity of knowledge in the team, increasing them both to a greater degree over time.

For solo programmers, previously working together reduces the accuracy of the team members' TMS over time and initial diversity of knowledge will increase the diversity over time. Other effects of initial conditions are dependent on other factors such as task size.

This might imply that team members with similar knowledge, who had worked together before would result in higher output. This is less important for pair programming factors, which seem to compensate for this.

12.2 Conclusion

The results of the three experiments carried out in this work can give some preliminary indications of the dynamics of knowledge and transactive memory in teams working under different development methodologies. With increasing realism the models have looked at the complex relationships between TMS accuracy, differentiation and team output.

Pair programming was found to be conducive to high output for all simulations and there was generally a positive relationship between TMS accuracy and output, and, as expected pair programming generated lower differentiation. So, contradicting the literature, which asserts that less differentiated knowledge inhibits team interdependence and reduces the reliance on the TMS (Brandon and Hollingshead, 2004, Espinosa et al., 2007a, Klimoski and Mohammed, 1994), negative relationships were found between TMS accuracy and differentiation for pair programming. For solo programming the first model found that higher output was attained for large tasks but in contradiction of the literature, with lower differentiation and lower TMS accuracy.

The main findings from the second model indicated that the use of pair programming reduces the drop in output resulting from the introduction of complexity or increases in unfamiliarity of the task. It was also found that task size has less effect in pair programming conditions and a larger effect in solo conditions. This model also found that the development of differentiation is delayed for more novel and complex tasks resulting in the differentiation although being significantly lower than for routine tasks, pair programming conditions produces higher differentiation than solo conditions.

The main findings from the third model showed that that team members with similar knowledge, who had worked together before, would result in higher output. The use of pair programming compensates for different initial team conditions producing consistently higher output and higher TMS accuracy. Also, in contradiction of the TMS literature (Brandon and Hollingshead, 2004, Espinosa et al., 2007a, Klimoski and Mohammed, 1994) the use of pair programming produces lower differentiation while maintaining the importance of an accurate TMS for better performance.

This research has challenged the established TMS literature by demonstrating that transactive memory systems processes may differ from those expected when utilising eXtreme programming techniques. These simulations have given a more detailed understanding of the behaviour of TMS, differentiation and ultimately output under different conditions and although it is important to be critical of the results of these simulations, some initial findings can be gleaned that may be applicable in the context of teams of software developers using XP practices. This research can provide a basis upon which to develop further investigation into TMS processes that take place in software development teams.

As interdisciplinary research spanning both psychology and computer science this thesis has contributed in a variety of ways. It has pioneered the use of FLAME in a novel way by modelling small numbers of agents where the information of interest is in their memory and not in their position in space. Secondly, this research has contributed to the TMS literature by highlighting potential circumstances that challenge existing research; and finally, this research contributes to the XP literature and to the understanding of factors that optimise the use of XP techniques and to identify optimum conditions for best team output.

All results and model code are attached in digital form; an index can be found at Appendix 2.

12.3 Limitations

Limitations to this work have to be recognised. As with any agent-based model the extent to which it represents reality must be questioned. Development of the model is an iterative process where the model becomes more plausible as it develops. The relationship between model and empirical or academic data is not one-way. Research will validate the model but data from the model may open up new lines of research, which will in turn feed back into the model design. The relationship is a circular one resulting in ongoing exploration of the domain of research.

The operation of this model was informed by theoretical research and the corpus relating to the subjects addressed, however, it is often the practice to carry out empirical research for the cycle of model development to take place. There were some attempts to carry out empirical research using the software engineering students at the University of Sheffield, working on projects using the XP software development methodology, as subjects. They were given questionnaires regularly throughout their project and were asked to report on their beliefs relating to their teammates' skills. It was felt that the results were unreliable, as the students did not take the questionnaire seriously. It was suspected that they did not fill them in truthfully, either through disinterest, becoming bored with the repetition of the questionnaire, or wanting to enhance the skill sets of their friends in the eyes of the staff. In addition there were no subjects relating to traditional software development methodologies available. So, in consequence it was decided to carry out a theoretical exercise using existing literature to inform the research.

12.4 Further work

This initial foray into the effects of different software development methodologies on transactive memory, differentiation and performance for software development teams has given some preliminary results which will be useful to inform subsequent research into this area. There are many avenues which this research could take potentially revealing some very interesting results.

The literature around transactive memory systems and XP is diverse and can provide more material for the development of the model:

- Existing parameters - Experiment with existing parameters such as knowledge range, task size, team size, communication settings and number of iterations.

- TMS - The introduction of greater cognitive realism by including other aspects of TMS such as consensus, knowledge stock and credibility

- Staff turnover - Investigate the impact on the team by changes in team members.

- Altruism - Other cognitive aspects could be added to the agents in the model such as including personal or team goals for agents. This would model selfish or altruistic behaviour in the team and the impact on performance, at the agent and team level.

- Forgetting - Currently agents in the model can only increase knowledge; deterioration of knowledge if it is not used is another potential development of the model.

- Knowledge granularity - This model has one measure of knowledge; this could be increased to include hierarchical levels of knowledge giving greater insight into how knowledge is distributed in a team.

- Time - This model has a fixed iteration period for agents to carry out tasks. The model could be enhanced to model changes in the time taken to fulfil tasks with some agents taking longer to fulfil tasks than others based on rules and conditions.

- Communication - changes in the volume and nature of communication in the team could be investigated.

- Multiple team interaction - the model could be developed to model multiple teams, agents moving between teams, inter-team communication and external knowledge sources.

- Trust - TMS is based on team members' beliefs. The model could be developed to introduce the concept of trust to establish the effect on knowledge, memory and team dynamics.

- XP - introduce more practices of the XP methodology in order to understand in greater detail the cognitive processes that take place.

The potential for the model to become ever more cognitively realistic and to include many more aspects of TMS are great. This thesis has introduced a preliminary model and established some interesting findings. It also contributes to the literature on understanding the benefits of XP practices and how they can be used for the best outcomes.

References

ACT-R. Available: http://act-r.psy.cmu.edu/ [Accessed 25.06.2011.

AKGUN, A. E., BYRNE, J., KESKIN, H., LYNN, G. S. & IMAMOGLU, S. Z. 2005. Knowledge networks in new product development projects: A transactive memory perspective. *Information & Management,* 42**,** 1105-1120.

AKGUN, A. E., BYRNE, J. C., KESKIN, H. & LYNN, G. S. 2006. Transactive memory system in new product development teams. *Ieee Transactions on Engineering Management,* 53**,** 95-111.

ALGE, B. J., WIETHOFF, C. & KLEIN, H. J. 2003. When does the medium matter? Knowledge-building experiences and opportunities in decision-making teams. *Organizational Behavior and Human Decision Processes,* 91**,** 26-37.

ANCONA, D. G. & CALDWELL, D. F. 1992. DEMOGRAPHY AND DESIGN - PREDICTORS OF NEW PRODUCT TEAM PERFORMANCE. *Organization Science,* 3**,** 321-341.

ARGOTE, L., INSKO, C. A., YOVETICH, N. & ROMERO, A. A. 1995. GROUP LEARNING-CURVES - THE EFFECTS OF TURNOVER AND TASK COMPLEXITY ON GROUP-PERFORMANCE. *Journal of Applied Social Psychology,* 25**,** 512-529.

ARISHOLM, E., GALLIS, H., DYBA, T. & SJOBERG, D. I. K. 2007. Evaluating pair programming with respect to system complexity and programmer expertise. *Ieee Transactions on Software Engineering,* 33**,** 65-86.

ASHWORTH, M. & CARLEY, K. 2007. Can tools help unify organization theory? Perspectives on the state of computational modeling. *Computational & Mathematical Organization Theory,* 13**,** 89-111.

AUSTIN, J. R. 2003. Transactive memory in organizational groups: The effects of content, consensus, specialization, and accuracy on group performance. *Journal of Applied Psychology,* 88**,** 866-878.

AXTELL, AXELROD, EPSTEIN & COHEN 1996a. Aligning Simulation Models: A Case Study and Results. *Computational and Mathematical Organization Theory,* 1**,** 123-141.

AXTELL, R. L., AXELROD, R., EPSTEIN, J. M. & COHEN, M. D. 1996b. Aligning Simulation Models: A Case Study and Results. *Computational and Mathematical Organization Theory,* 1**,** 123-141.

AXTELL, R. L., EPSTEIN, J. M., DEAN, J. S., GUMERMAN, G. J., SWEDLUND, A. C., HARBURGER, J., CHAKRAVARTY, S., HAMMOND, R., PARKER, J. & PARKER, M. Population growth and collapse in a multiagent model of the Kayenta Anasazi in Long House

Valley. Sackler Colloquium on Adaptive Agents, Intelligence, and Emergent Human Organization - Capturing Complexity through Agent-Based Modeling, Oct 04-06 2001 Irvine, California. Natl Acad Sciences, 7275-7279.

BALANESCU, T., COWLING, A. J., GEORGESCU, M., HOLCOMBE, M. & VERTAN, C. 1999. Communicating stream X-machines are no more than X-machines. *Journal of Universal Computer Science,* 5**,** 494-507.

BALCI, O., GILLEY, W. S., ADAMS, R. J., TUNAR, E. & BARNETTE, N. D. 2010. *The Waterfall Model* [Online]. Virginia Tech,Blacksburg, VA 24061, U.S.A. Available: http://courses.cs.vt.edu/csonline/SE/Lessons/Waterfall/index.html [Accessed 15.11.2010.

BALDAZZI, V., CASTIGLIONE, F. & BERNASCHI, M. 2006. An enhanced agent based model of the immune system response. *Cellular Immunology,* 244**,** 77-79.

BARON, R. M. & KENNY, D. A. 1986. THE MODERATOR MEDIATOR VARIABLE DISTINCTION IN SOCIAL PSYCHOLOGICAL-RESEARCH - CONCEPTUAL, STRATEGIC, AND STATISTICAL CONSIDERATIONS. *Journal of Personality and Social Psychology,* 51**,** 1173-1182.

BAUMANN, M. R. & BONNER, B. L. 2004. The effects of variability and expectations on utilization of member expertise and group performance. *Organizational Behavior and Human Decision Processes,* 93**,** 89-101.

BAUMANN, M. R. & BONNER, B. L. 2011. The Effects of Expected Group Longevity and Expected Task Difficulty on Learning and Recall: Implications for the Development of Transactive Memory. [Article]. *Group Dynamics: Theory, Research, & Practice,* Advance online.

BECK, K. 1999. *Extreme Programming Explained*, Addison-Wesley.

BECK, K., BEEDLE, M., BENNEKUM, A. V., COCKBURN, A., CUNNINGHAM, W., FOWLER, M., GRENNING, J., HIGHSMITH, J., HUNT, A., JEFFRIES, R., KERN, J., MARICK, B., MARTIN, R. C., MELLOR, S., SCHWABER, K., SUTHERLAND, J. & THOMAS, D. 2001. *The Agile Manifesto* [Online]. Available: http://www.agilemanifesto.org/principles.html [Accessed 11/01/10.

BEGEL, A., NAGAPPAN, N. & ACM 2008. Pair Programming: What's in it for Me? *Esem'08: Proceedings of the 2008 Acm-Ieee International Symposium on Empirical Software Engineering and Measurement***,** 120-128.

BELLINI, E., CANFORA, G., CIMITILE, A., GARCIA, F., PIATTINI, M. & VISAGGIO, C. A. 2005. Impact of educational background on design knowledge sharing during pair programming: An empirical study. *In:* ALTHOFF, K. D., DENGEL, A., BERGMANN, R., NICK, M. & ROTHBERGHOFER, T. (eds.) *Professional Knowledge Management.* Berlin: Springer-Verlag Berlin.

BICAK, M. 2010. *Agent-Based Modelling of Decentralised Ant Behaviour using HIgh Performance Computing.* PhD, University of Sheffield.

BLANCHE. Available: http://blanche.northwestern.edu/ [Accessed 25.06.2011.

BOERO, R., CASTELLANI, M. & SQUAZZONI, F. 2008. Individual behavior and macro social properties. An agent-based model. *Computational and Mathematical Organization Theory,* 14**,** 156-174.

BOLLINGER, T. 2001. XP: Two Concerns. *eXtreme Programming Pros and Cons: What Questions Remain?* : IEEE Computer Society Dynabook.

BONABEAU, E. 2002. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences of the United States of America,* 99**,** 7280-7287.

BOSWELL, J. 1986. *The Life of Samuel Johnson,* New York, Penguin Classics.

BRANDON, D. P. & HOLLINGSHEAD, A. B. 2004. Transactive Memory Systems in Organizations: Matching Tasks, Expertise, and People. *Organization Science,* 15**,** 633-644.

BRAUNER, E. & BECKER, A. 2006. Beyond knowledge sharing: the management of transactive knowledge systems. *Knowledge and Process Management,* 13**,** 62-71.

BRYSON, J. J., ANDO, Y. & LEHMANN, H. 2007. Agent-based modelling as scientific method: a case study analysing primate social behaviour. *Philosophical Transactions of the Royal Society B-Biological Sciences,* 362**,** 1685-1698.

BURTON, R. M. & OBEL, B. 1995. The validity of computational models in organization science: From model realism to purpose of the model. *Computational & Mathematical Organization Theory,* 1**,** 57-71.

CANFORA, G., CIMITILE, A. & VISAGGIO, C. A. 2005. Empirical study on the productivity of the pair programming. *Extreme Programming and Agile Processes in Software Engineering, Proceedings,* 3556**,** 92-99.

CANNON-BOWERS, J. A. & SALAS, E. 2001. Reflections on shared cognition. *Journal of Organizational Behavior,* 22**,** 195-202.

CHI, M. T. H., FELTOVICH, P. J. & GLASER, R. 1981. CATEGORIZATION AND REPRESENTATION OF PHYSICS PROBLEMS BY EXPERTS AND NOVICES. *Cognitive Science,* 5**,** 121-152.

CHI, M. T. H., GLASER, R. & REES, E. 1982. Expertise in problem solving. *In:* STERNBERG, R. (ed.) *Advances in the Psychology of Human Intelligence.* Lawrence Erlbaum Associates Inc.

CHOMSKY, N. 1957. *Syntactic structures,* The Hague, Mouton.

CLARION. Available: http://www.ccc.utexas.edu/cogsci08/tut05-helie.pdf [Accessed 25.06.2011.

COAKLEY, S. 2011. *FLAME* [Online]. Available: http://www.flame.ac.uk/ [Accessed 18/07/2011 2011].

COAKLEY, S., SMALLWOOD, R. & HOLCOMBE, M. 2006a. FROM MOLECULES TO INSECT COMMUNITIES - HOW FORMAL AGENT BASED COMPUTATIONAL MODELLING IS UNCOVERING NEW BIOLOGICAL FACTS. *Scientiae Mathematicae Japonicae Online,* e-2006**,** 765–778.

COAKLEY, S., SMALLWOOD, R. & HOLCOMBE, M. Using X-Machines as a formal basis for describing agents in agent-based modelling. European Conference on Complex Systems, 2006b.

COCKBURN, A. & HIGHSMITH, J. 2001. Agile software development, the people factor. *Computer,* 34**,** 131-133.

COOKE, N. J., KIEKEL, P. A., SALAS, E., STOUT, R., BOWERS, C. & CANNON-BOWERS, J. 2003. Measuring team knowledge: A window to the cognitive underpinnings of team performance. *Group Dynamics-Theory Research and Practice,* 7**,** 179-199.

COOKE, N. J., SALAS, E., CANNON-BOWERS, J. A. & STOUT, R. J. 2000. Measuring Team Knowledge. *Human Factors: The Journal of the Human Factors and Ergonomics Society,* 42**,** 151-173.

DECHURCH, L. A. & MESMER-MAGNUS, J. R. 2010. Measuring Shared Team Mental Models: A Meta-Analysis. *Group Dynamics-Theory Research and Practice,* 14**,** 1-14.

DEISSENBERG, C., VAN DER HOOG, S. & DAWID, H. 2008. EURACE: A massively parallel agent-based model of the European economy. *Applied Mathematics and Computation,* 204**,** 541-552.

DENG, P. S. & TSACLE, E. G. 2006. Emergent learning behaviour in a simulated organization faced with tasks requiring team effort. *Journal of the Operational Research Society,* 57**,** 603-611.

DIONNE, S. D., SAYAMA, H., HAO, C. Y. & BUSH, B. J. 2010. The role of leadership in shared mental model convergence and team performance improvement An agent-based computational model. *Leadership Quarterly,* 21**,** 1035-1049.

DYBA, T. & DINGSOYR, T. 2008. Empirical studies of agile software development: A systematic review. *Information and Software Technology,* 50**,** 833-859.

EILENBERG, S. 1974. *Automata, Languages and Machines*, Academic Press.

ELEFTHERAKIS, G., KEFALAS, P., SOTIRIADOU, A. & KEHRIS, E. 2005. Modeling Biology Inspired Reactive Agents Using X-machines. *World Academy of Science, Engineering and Technology,* 1**,** 63-66.

ERICSSON, K. A. & CHARNESS, N. 1994. EXPERT PERFORMANCE - ITS STRUCTURE AND ACQUISITION. *American Psychologist,* 49**,** 725-747.

ESPINOSA, J. A., SLAUGHTER, S. A., KRAUT, R. E. & HERBSLEB, J. D. 2007a. Familiarity, complexity, and team performance in geographically distributed software development. *Organization Science,* 18**,** 613-630.

ESPINOSA, J. A., SLAUGHTER, S. A., KRAUT, R. E. & HERBSLEB, J. D. 2007b. Team knowledge and coordination in geographically distributed software development. *Journal of Management Information Systems,* 24**,** 135-169.

EYSENCK, M. W. & KEANE, M. T. 2000. *Cognitive Psychology,* Hove, East Sussex, Psychology Press Ltd.

FAN, X. & YEN, J. 2004. Modeling and simulating human teamwork behaviors using intelligent agents. *Physics of Life Reviews,* 1**,** 173-201.

FARAJ, S. & SPROULL, L. 2000. Coordinating expertise in software development teams. *Management Science,* 46**,** 1554-1568.

FARMER, J. D., PATELLI, P. & ZOVKO, I. 2005. The predictive power of zero intelligence in financial markets. *Proceedings of the National Academy of Sciences of the United States of America,* 102**,** 2254-2259.

FIELD, A. 2005. *Discovering Statistics Using SPSS,* London, Sage.

FIORE, S. M., ROSEN, M. A., SMITH-JENTSCH, K. A., SALAS, E., LETSKY, M. & WARNER, N. 2010. Toward an Understanding of Macrocognition in Teams: Predicting Processes in Complex Collaborative Contexts. *Human Factors,* 52**,** 203-224.

FISHER, J. & HENZINGER, T. A. 2007. Executable cell biology. *Nature Biotechnology,* 25**,** 1239-1249.

FUM, D., DEL MISSIER, F. & STOCCO, A. 2007. The cognitive modeling of human behavior: Why a model is (sometimes) better than 10,000 words. *Cognitive Systems Research,* 8**,** 135-142.

GHEORGHE, M., HOLCOMBE, M. & KEFALAS, P. 2001. Computational models of collective foraging. *Biosystems,* 61**,** 133-141.

GIBSON, C. B. 1999. Do They Do What They Believe They Can? Group Efficacy and Group Effectiveness across Tasks and Cultures. *The Academy of Management Journal,* 42**,** 138-152.

GILBERT, N. 2008. *Agent-based models,* London, Sage.

GOLDSTONE, R. L. & JANSSEN, M. A. 2005. Computational models of collective behavior. *Trends in Cognitive Sciences,* 9**,** 424-430.

GOLDSTONE, R. L., ROBERTS, M. E. & GURECKIS, T. M. 2008. Emergent processes in group behavior. *Current Directions in Psychological Science,* 17**,** 10-15.

GORE, R. & REYNOLDS, P. F. 2007. An exploration-based taxonomy for emergent behavior analysis in simulations. *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come.* Washington D.C.: IEEE Press.

GRIFFITH, T. L. & NEALE, M. A. 2001. 8. Information processing in traditional, hybrid, and virtual teams: From nascent knowledge to transactive memory. *Research in Organizational Behavior,* 23**,** 379-421.

GRIMM, V. 1999. Ten years of individual-based modelling in ecology: what have we learned and what could we learn in the future? *Ecological Modelling,* 115**,** 129-148.

GUPTA, N. & HOLLINGSHEAD, A. B. 2010. Differentiated Versus Integrated Transactive Memory Effectiveness: It Depends on the Task. *Group Dynamics-Theory Research and Practice,* 14**,** 384-398.

HANNAY, J. E., DYBA, T., ARISHOLM, E. & SJOBERG, D. I. K. 2009. The effectiveness of pair programming: A meta-analysis. *Information and Software Technology,* 51**,** 1110-1122.

HAZY, J. K. 2007. Computer models of leadership: Foundations for a new discipline or meaningless diversion? *The Leadership Quarterly,* 18**,** 391-410.

HE, J., BUTLER, B. S. & KING, W. R. 2007. Team cognition: Development and evolution in software project teams. *Journal of Management Information Systems,* 24**,** 261-292.

HOEGL, M. & GEMUENDEN, H. G. 2001. Teamwork quality and the success of innovative projects: A theoretical concept and empirical evidence. *Organization Science,* 12**,** 435-449.

HOEGL, M. & PARBOTEEAH, K. P. 2007. Creativity in innovative projects: How teamwork matters. *Journal of Engineering and Technology Management,* 24**,** 148-166.

HOEGL, M., PARBOTEEAH, K. P. & GEMUENDEN, H. G. 2003. When teamwork really matters: task innovativeness as a moderator of the teamwork-performance relationship in software development projects. *Journal of Engineering and Technology Management,* 20**,** 281-302.

HOLCOMBE, M. 1988. X-machines as a basis for dynamic system specification. *Software Engineering Journal,* 3**,** 69-76.

HOLCOMBE, M. 2008. *Running an Agile Software Development Project,* Hoboken, New Jersey, Wiley.

HOLCOMBE, M., COAKLEY, S. & SMALLWOOD, R. A General Framework for agent-based modelling of complex systems European Conference on Complex Systems, 2006.

HOLLINGSHEAD, A. B. 1998. Communication, Learning, and Retrieval in Transactive Memory Systems. *Journal of Experimental Social Psychology,* 34**,** 423-442.

HOLLINGSHEAD, A. B. 2000. Perceptions of Expertise and Transactive Memory in Work Relationships. *Group Processes Intergroup Relations,* 3**,** 257-267.

HOLLINGSHEAD, A. B. 2001. Cognitive interdependence and convergent expectations in transactive memory. *Journal of Personality and Social Psychology,* 81**,** 1080-1089.

HUTCHINS, E. 1992. Distributed Cognition. *In:* RESNICK, L. B. (ed.) *Perspectives on Socially Shared Cognition.* Washington DC: American Psychological Association.

IOERGER, T. R. & JOHNSON, J. C. A formal model of responsibilities in agent-based teamwork. *In:* ARABNIA, H. R., ed. International Conference on Artificial Intelligence, Jun 25-28 2001 Las Vegas, Nv. C S R E a Press, 58-64.

JACKSON, D., HOLCOMBE, M. & RATNIEKS, F. 2004. Coupled computational simulation and empirical research into the foraging system of Pharaoh's ant (Monomorium pharaonis). *Biosystems,* 76**,** 101-112.

JACKSON, M. & MORELAND, R. L. 2009. Transactive Memory in the Classroom. *Small Group Research,* 40**,** 508-534.

JEFFRIES, R. E. 2011. *What is Extreme Programming?* [Online]. Available: http://xprogramming.com/what-is-extreme-programming [Accessed 12.3.2011.

JI, F. & SEDANO, T. 2011. Comparing Extreme Programming and Waterfall Project Results. *Conference on Software Engineering Education and Training.* Carnegie Mellon University.

JURAN, D. C. & SCHRUBEN, L. W. 2004. Using worker personality and demographic information to improve system performance prediction. *Journal of Operations Management,* 22**,** 355-367.

KANG, H. R., YANG, H. D. & ROWLEY, C. 2006. Factors in team effectiveness: Cognitive and demographic similarities of software development team members. *Human Relations,* 59**,** 1681-1710.

KANG, M. 2007. The effects of agent activeness and cooperativeness on team decision efficiency: A computational simulation study using Team-Soar. *International Journal of Human-Computer Studies,* 65**,** 497-510.

KIRAN, M., COAKLEY, S., WALKINSHAW, N., MCMINN, P. & HOLCOMBE, M. 2008. Validation and discovery from computational biology models. *Biosystems,* 93**,** 141-150.

KLIMOSKI, R. & MOHAMMED, S. 1994. Team mental model: construct or metaphor? *Journal of Management,* 20**,** 403-437.

LARSON, J. R. 2007. Deep diversity and strong synergy - Modeling the impact of variability in members' problem-solving strategies on group problem-solving performance. *Small Group Research,* 38**,** 413-436.

LEVESQUE, L. L., WILSON, J. M. & WHOLEY, D. R. 2001. Cognitive divergence and shared mental models in software development project teams. *Journal of Organizational Behavior,* 22**,** 135-144.

LEWIS, K. 2003. Measuring transactive memory systems in the field: Scale development and validation. *Journal of Applied Psychology,* 88**,** 587-604.

LEWIS, K. 2004. Knowledge and performance in knowledge-worker teams: A longitudinal study of transactive memory systems. *Management Science,* 50**,** 1519-1533.

LIANG, D. W., MORELAND, R. & ARGOTE, L. 1995. GROUP VERSUS INDIVIDUAL TRAINING AND GROUP-PERFORMANCE - THE MEDIATING ROLE OF TRANSACTIVE MEMORY. *Personality and Social Psychology Bulletin,* 21**,** 384-393.

LITTLEPAGE, G. E., HOLLINGSHEAD, A. B., DRAKE, L. R. & LITTLEPAGE, A. M. 2008. Transactive memory and performance in work groups: Specificity, communication, ability differences, and work allocation. *Group Dynamics-Theory Research and Practice,* 12**,** 223-241.

LOUIE, M. A. & CARLEY, K. M. 2008. Balancing the criticisms: Validating multi-agent models of social systems. *Simulation Modelling Practice and Theory,* 16**,** 242-256.

LUI, K. M. & CHAN, K. C. C. 2006. Pair programming productivity: Novice-novice vs. expert-expert. *International Journal of Human-Computer Studies,* 64**,** 915-925.

MACAL, C. M. & NORTH, M. J. 2006. Tutorial on agent-based modeling and simulation part 2: how to model with agents. *Proceedings of the 38th conference on Winter simulation.* Monterey, California: Winter Simulation Conference.

MACIAS, F. J. 2004. *Empirical Assessment of Extreme Programming.* Doctor of Philosophy, University of Sheffield

MACIAS, F. J., HOLCOMBE, M. & GHEORGHE, M. 2003. A formal experiment comparing extreme programming with traditional software construction. *Proceedings of the Fourth Mexican International Conference on Computer Science (Enc 2003)***,** 73-80.

MARKS, M. A. 2000. A Critical Analysis of Computer Simulations for Conducting Team Research. *Small Group Research,* 31**,** 653-675.

MARUPING, L. M., ZHANG, X. J. & VENKATESH, V. 2009. Role of collective ownership and coding standards in coordinating expertise in software project teams. *European Journal of Information Systems,* 18**,** 355-371.

MCCONNELL, S. 1996. *Rapid Development: Taming Wild Software Schedules.*, Microsoft Press.

MORELAND, R. L. 1999. Transactive Memory: Learning who knows what in work groups and organizations. *In:* THOMPSON, L. L., LEVINE, J. M. & MESSICK, D. M. (eds.) *Shared Cognition in Organizations: The management of knowledge.* Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

MULLER, M. M. 2005. Two controlled experiments concerning the comparison of pair programming to peer review. *Journal of Systems and Software,* 78**,** 166-179.

MULLER, M. M. 2007. Do programmer pairs make different mistakes than solo programmers? *Journal of Systems and Software,* 80**,** 1460-1471.

NOSEK, J. T. 1998. The case for collaborative programming. *Communications of the Acm,* 41**,** 105-108.

ORGMEM. Available: http://www.casos.cs.cmu.edu/projects/orgmem/index.html [Accessed 12.07.2011.

PALAZZOLO, E. T., SERB, D. A., SHE, Y. C., SU, C. K. & CONTRACTOR, N. S. 2006. Coevolution of communication and knowledge networks in transactive memory systems: Using computational models for theoretical development. *Communication Theory,* 16**,** 223-250.

PANZARASA, P. & JENNINGS, N. R. 2002. Social influence, negotiation and cognition. *Simulation Modelling Practice and Theory,* 10**,** 417-453.

PELTOKORPI, V. 2008. Transactive Memory Systems. *Review of General Psychology,* 12**,** 378-394.

PFLEEGER, S. L. & ATLEE, J. M. 2010. *Software Engineering,* New Jersey, US, Pearson.

PHELAN, S. E. 2002. Cognitive capacity as competitive advantage: a simulation test. *Simulation Modelling Practice and Theory,* 10**,** 455-471.

POGSON, M., SMALLWOOD, R., QWARNSTROM, E. & HOLCOMBE, M. 2006. Formal agent-based modelling of intracellular chemical interactions. *Biosystems,* 85**,** 37-45.

PRICHARD, J. S. & ASHLEIGH, M. J. 2007. The Effects of Team-Skills Training on Transactive Memory and Performance. *Small Group Research,* 38**,** 696-726.

PURNOMO, H., MENDOZA, G. A., PRABHU, R. & YASMI, Y. 2005. Developing multi-stakeholder forest management scenarios: a multi-agent system simulation approach applied in Indonesia. *Forest Policy and Economics,* 7**,** 475-491.

REAGANS, R., ARGOTE, L. & BROOKS, D. 2005. Individual experience and experience working together: Predicting learning rates from knowing who knows what and knowing how to work together. *Management Science,* 51**,** 869-881.

REN, Y. Q., CARLEY, K. M. & ARGOTE, L. 2006. The contingent effects of transactive memory: When is it more beneficial to know what others know? *Management Science,* 52**,** 671-682.

RENTSCH, J. R. & KLIMOSKI, R. J. 2001. Why do 'great minds' think alike?: antecedents of team member schema agreement. *Journal of Organizational Behavior,* 22**,** 107-120.

RESNICK, M. 1996. *Exploring Emergence* www.playfulinvention.com/emergence/contents.html.

RICO, R., SANCHEZ-MANZANARES, M., GIL, F. & GIBSON, C. 2008. Team implicit coordination processes: A team knowledge-based approach. *Academy of Management Review,* 33**,** 163-184.

ROTH, C. 2007. Empiricism for descriptive social network models. *Physica A: Statistical Mechanics and its Applications,* 378**,** 53-58.

ROUCHIER, J., BOUSQUET, F., REQUIER-DESJARDINS, M. & ANTONA, M. 2001. A multi-agent model for describing transhumance in North Cameroon: Comparison of different rationality to develop a routine. *Journal of Economic Dynamics & Control,* 25**,** 527-559.

ROYCE, W. Managing the Development of Large Software Systems. IEEE WESTCON, 1970 Los Angeles. 1-9.

SALAS, E., COOKE, N. J. & ROSEN, M. A. 2008. On teams, teamwork, and team performance: Discoveries and developments. *Human Factors,* 50**,** 540-547.

SALAS, E. & FIORE, S. M. (eds.) 2004. *Team Cognition: Understanding the factors that drive process and performance,* Washington, DC: American Psychological Association.

SMITH, E. R. & CONREY, F. R. 2007. Agent-based modeling: A new approach for theory building in social psychology. *Personality and Social Psychology Review,* 11**,** 87-104.

SMITH-JENTSCH, K. A., KRAIGER, K., CANNON-BOWERS, J. A. & SALAS, E. 2009. Do Familiar Teammates Request and Accept More Backup? Transactive Memory in Air Traffic Control. *Human Factors,* 51**,** 181-192.

SOAR. Available: http://sitemaker.umich.edu/soar/home [Accessed 25.06.2011.

SOMMERVILLE, I. 2004. *Software Engineering,* Harlow, Essex, Pearson Education Ltd.

STANNETT, M. 2005. *X-machines.com* [Online]. Available: http://x-machines.com/physicaldescription.php.

SUN, R. (ed.) 2006. *Cogntion and Multi-Agent Interaction: From cognitive modelling to social simulation,* Cambridge, UK.: Cambridge University Press.

SUN, R. 2009. Theoretical status of computational cognitive modeling. *Cognitive Systems Research,* 10**,** 124-140.

SUN, R. & NAVEH, I. 2004. Simulating organizational decision-making using a cognitively realistic agent model. *Jasss-the Journal of Artificial Societies and Social Simulation,* 7**,** 27.

SYED-ABDULLAH, S. L., HOLCOMBE, M. & GHEORGE, M. 2006. The impact of an agile methodology on the well being of development teams. *Empirical Software Engineering,* 11**,** 143-167.

SYED-ABDULLAH, S. L., KARN, J., HOLCOMBE, M., COWLING, T. & GHEORGE, M. 2005. The positive affect of the XP methodology. *Extreme Programming and Agile Processes in Software Engineering, Proceedings,* 3556**,** 218-221.

THEINER, G., ALLEN, C. & GOLDSTONE, R. L. 2010. Recognizing group cognition. *Cognitive Systems Research,* 11**,** 378-395.

WALKER, D. C., SOUTHGATE, J., HILL, G., HOLCOMBE, A., HOSE, D. R., WOOD, S. M., MAC NEIL, S. & SMALLWOOD, R. H. 2004. The epitheliome: agent-based modelling of the social behaviour of cells. *Biosystems,* 76**,** 89-100.

WALKER, D. C., SUN, T., MACNEIL, S. & SMALLWOOD, R. 2006. Modeling the effect of exogenous calcium on keratinocyte and HaCat cell proliferation and differentiation using an agent-based computational paradigm. *Tissue Engineering,* 12**,** 2301-2309.

WATTS, H. 2001. Comments on eXtreme Programming. *eXtreme Programming Pros and Cons: What questions remain?* : IEEE Computer Society Dynabook.

WEGNER, D. M. 1995. A computer network model of human transactive memory. *Social Cognition,* 13**,** 319-339.

WELLS, D. 2009. *Extreme Programming* [Online]. Available: http://www.extremeprogramming.org/ [Accessed 28/07/2011 2011].

WILLIAMS, L. 2000. *The Collaborative Software Process.* PhD, University of Utah.

WILLIAMS, L., KESSLER, R. R., CUNNINGHAM, W. & JEFFRIES, R. 2000. Strengthening the case for pair programming. *Ieee Software,* 17**,** 19-+.

YEN, J., FAN, X. C., SUN, S., WANG, R., CHEN, C., KAMALI, K., MILLER, M. & VOLZ, R. A. 2003. On modeling and simulating agent teamwork in CAST. *Active Media Technology***,** 18-33.

YUAN, Y. C., FULK, J. & MONGE, P. R. 2007. Access to information in connective and communal transactive memory systems. *Communication Research,* 34**,** 131-155.

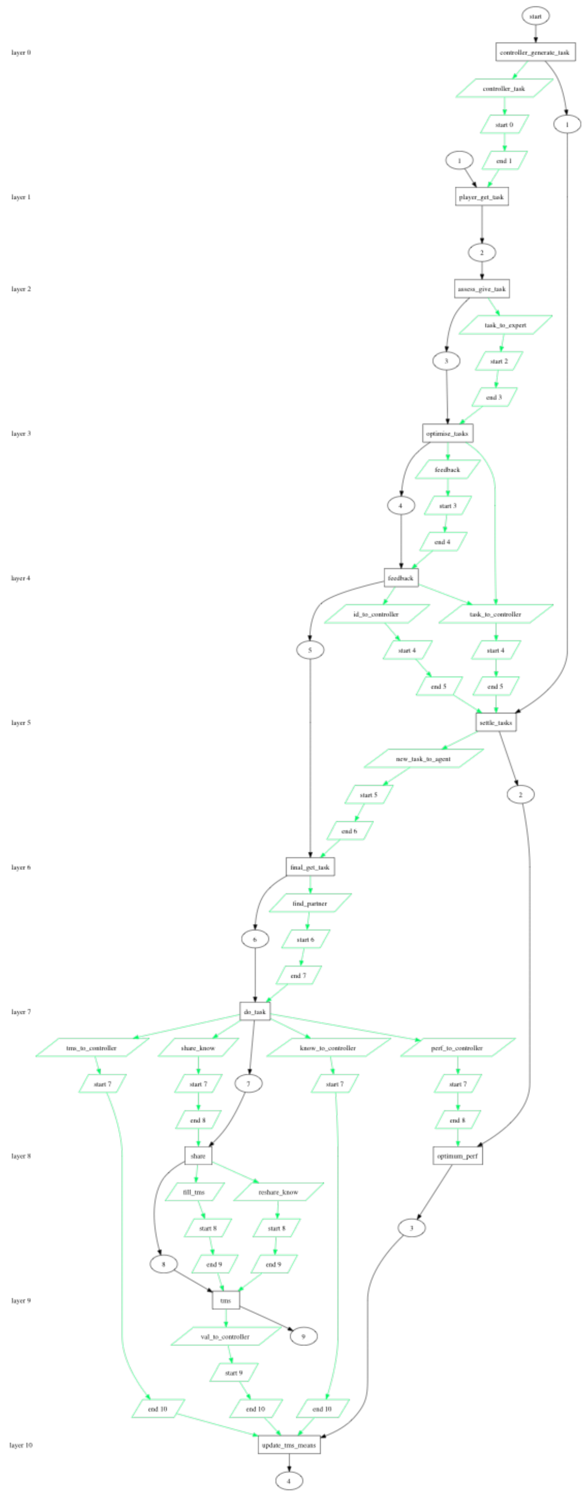
ZHANG, Z. X., HEMPEL, P. S., HAN, Y. L. & TJOSVOLD, D. 2007. Transactive memory system links work team characteristics and performance. *Journal of Applied Psychology,* 92**,** 1722-1730.

ZHUGE, H. 2003. Workflow- and agent-based cognitive flow management for distributed team Cooperation. *Information & Management,* 40**,** 419-429.

Appendix 1. Stategraph of the model

Appendix 1: Stategraph of the model

Appendix 1: Stategraph of the model



Appendix 1: Stategraph of the model

Appendix 2. Digital resources

This appendix provides an index of the digital resources that are supplied with this thesis.

1. PDF of this thesis
2. Stategraph of the model
3. The model
    a. controller_functions.c – functions for controller that provides team level services such as issuing tasks and calculating team level measures.
    b. mod5.xml – file providing the xml definition of the agents and the environment.
    c. my_library_functions.c – library of functions for use throughout the model
    d. player_functions.c – functions for the operation of the individual agents
4. Results
    a. Validation model
        i. 0.xml
        ii. valid.xlsx
    b. Alignment model
        i. notms.xlsx – model output and calculation for trained individually condition
        ii. tms.xlsx – model output and calculation for trained as a team condition
        iii. repmeasacc.spv, repmeasdiff.spv and repmeasoutput.spv – SPSS output files with results of repeated measures ANOVA

c. Model Tabula Rasa

    i. agile4 - 0.xml and spreadsheet with model output for pair programming and small tasks condition

    ii. agile20 - 0.xml and spreadsheet with model output for pair programming and large tasks condition

    iii. trad4 - 0.xml and spreadsheet with model output for solo programming and small tasks condition

    iv. trad20 - 0.xml and spreadsheet with model output for solo programming and large tasks condition

    v. xp_stats.spv – SPSS output file containing output of statistical tests.

d. Model Novelty

    i. novelty_stats.spv – SPSS file containing output of statistical tests

    ii. pp4k100 - 0.xml and spreadsheet with model output for high novelty, pair programming and small tasks

    iii. pp20k100- 0.xml and spreadsheet with model output for high novelty, pair programming and large tasks

    iv. wa4k100 - 0.xml and spreadsheet with model output for high novelty, solo programming, small tasks

    v. wa20k100 - 0.xml and spreadsheet with model output for high novelty, solo programming and large tasks

e. Model Differentiation

    i. pp4x4dhi notms – 0.xml and spreadsheet with model output for pair programming, small tasks, high initial differentiation and no initial TMS

    ii. pp4x4dhi tms - 0.xml and spreadsheet with model output for pair programming, small tasks, high initial differentiation and an initial TMS

    iii. pp4x4dlo notms - 0.xml and spreadsheet with model output for pair programming, small tasks, low initial differentiation and no initial TMS

iv. pp20x4dlo tms - 0.xml and spreadsheet with model output for pair programming, large tasks, low initial differentiation and an initial TMS

v. pp20x4dhi notms – 0.xml and spreadsheet with model output for pair programming, large tasks, high initial differentiation and no initial TMS

vi. pp20x4dhi tms - 0.xml and spreadsheet with model output for pair programming, large tasks, high initial differentiation and an initial TMS

vii. pp20x4dlo notms - 0.xml and spreadsheet with model output for pair programming, large tasks, low initial differentiation and no initial TMS

viii. pp20x4dlo tms - 0.xml and spreadsheet with model output for pair programming, large tasks, low initial differentiation and an initial TMS

ix. stats.spv – SPSS output file containing output of statistical tests

x. wa4x4dhi notms – 0.xml and spreadsheet with model output for solo programming, small tasks, high initial differentiation and no initial TMS

xi. wa4x4dhi tms - 0.xml and spreadsheet with model output for solo programming, small tasks, high initial differentiation and an initial TMS

xii. wa4x4dlo notms - 0.xml and spreadsheet with model output for solo programming, small tasks, low initial differentiation and no initial TMS

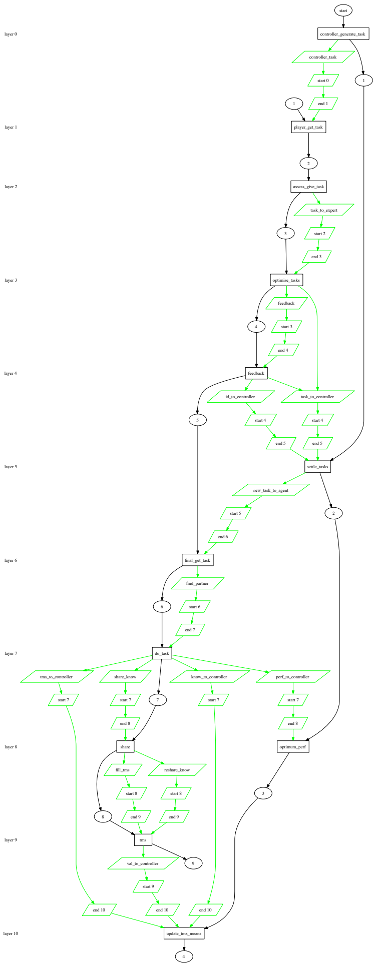
xiii. wa20x4dlo tms - 0.xml and spreadsheet with model output for solo programming, large tasks, low initial differentiation and an initial TMS

xiv. wa20x4dhi notms – 0.xml and spreadsheet with model output for solo programming, large tasks, high initial differentiation and no initial TMS

xv. wa20x4dhi tms - 0.xml and spreadsheet with model output for solo programming, large tasks, high initial differentiation and an initial TMS

xvi. wa20x4dlo notms - 0.xml and spreadsheet with model output for solo programming, large tasks, low initial differentiation and no initial TMS

xvii. wa20x4dlo tms - 0.xml and spreadsheet with model output for solo programming, large tasks, low initial differentiation and an initial TMS



start
layer 0
controller_generate_task
controller_task
1
start 0
end 1
1
layer 1
player_get_task
2
layer 2
assess_give_task
task_to_expert
3
start 2
end 3
layer 3
optimise_tasks
feedback
4
start 3
end 4
layer 4
feedback
id_to_controller
task_to_controller
5
start 4
start 4
end 5
end 5
layer 5
settle_tasks
new_task_to_agent
2
start 5
end 6
layer 6
final_get_task
find_partner
6
start 6
end 7
layer 7
do_task
tms_to_controller
share_know
know_to_controller
perf_to_controller
start 7
start 7
7
start 7
start 7
end 8
end 8
layer 8
share
optimum_perf
fill_tms
reshare_know
3
start 8
start 8
8
end 9
end 9
layer 9
tms
val_to_controller
9
start 9
end 10
end 10
end 10
layer 10
update_tms_means
4