Advances in Social Sciences Research Journal – Vol.2, No.2
Publication Date: February 25, 2015
DOI:10.14738/assrj.22.474.
Corbett, A., Holcombe, M., & Wood, S. (2015). It's the People, Stupid! – Formal Models of Social Interaction in Agile Software Development Teams. *Advances in Social Sciences Research Journal, 2*(2*)*, 70-85.

It's the People, Stupid! - Formal Models of Social Interaction in Agile Software Development Teams

Andrea Corbett
Tessella, Abingdon

Mike Holcombe
Department of Computer Science. University of Sheffield

Stephen Wood
School of Management. University of Leicester

ABSTRACT
The success of modern ICT systems is not just about the technology. The human and social dimensions are also critical – especially in terms of understanding the context and environment within which the systems operate. This is true also about the environment in which the system is developed. Although much effort has been expended on building and analysing formal models of software systems, little has been done in terms of how software development teams work and how this might be studied in a formally-based way. This research looks at one of the fundamental aspects involved in collaborative teams working in projects – the transactive memory system (TMS). This, well established, concept in psychology is an approach to how the different people in a team regard the capabilities (knowledge and abilities) of each other as it changes over time. These capabilities are the basis for decision making in software projects about who does what, and when. Using a formal model of the TMS of a team, based on agent-based modelling, simulations were made of how teams might operate under different circumstances. The initial model was validated against published data. The model was then investigated in terms of how different types of project management affected the TMS of a team and on the team's performance. In particular, a comparison was made between a traditional, plan-based approach against an agile method using pair programming. The result demonstrates strong benefits in terms of performance and learning with the agile approach.

INTRODUCTION

Transactive memory theory is growing in interest but there is ambiguity regarding many of the aspects of it including, measurement, antecedents, its impact and how it fits with the larger body of team cognitive theory. Transactive memory is defined as an individual's beliefs relating to "who knows what" in their team (Wegner, 1995) and a Transactive Memory System (TMS) consists of transactive memory as well as the communication and coordination that takes place at team level to manage that knowledge. For a recent overview of TMS research see Peltokorpi (2008). Much of the research uses laboratory experiments (Hollingshead, 2000) or fieldwork (Peltokorpi and Manka, 2008) with teams carrying out repetitive familiar tasks however this paper focuses on teams in the current fast moving knowledge economy, specifically, software development teams who face novel tasks as a matter of course. To take a new perspective on research in this area this paper describes an agent-based modelling approach.

This paper describes how we have used agent based modelling to simulate the knowledge processes and transactive memory systems in software development teams. We have used this method to compare the different processes present in Agile and traditional methodologies and evaluate the outcomes.

Agent-based modelling

Increasingly agent based computational models have been used that can simulate behaviour in a system allowing it to be observed and measured over time. In this way experiments can be designed and the results interpreted by examining the way the model behaves.

Often the 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 and as the number of agents grows the potential for unpredictable or emergent behaviour will increase. There are many examples of emergent behaviour in areas such as nature, physics, biology, economics, and technology including ant colonies, flocks of birds, traffic behaviour, economic systems and weather.

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 that are encoded within their functions. 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 are usually not omnipresent and only act in their immediate environment, in their own interests.

There are many benefits to agent-based modelling (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. Secondly, agent-based models 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. Another benefit to this method is that it 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 agent-based models that should be highlighted. Of course agent-based models 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). Another limitation is related to the highly specialised nature of agent-based models. Nearly everyone can understand narrative but only those familiar with programming languages and environments will understand the intricacies of a model. Finally, 'Bonini's Paradox' (Fum et al., 2007) 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. Agent-based modelling has been used to simulate, amongst others, ants (Georghe, 2001), cells (Walker et al., 2004), and the human immune system (Baldazzi et al., 2006).

FLAME stands for Flexible Large-scale Agent-Based Modelling Environment (FLAME) (Pogson M, 2006) and it is based on a formal X-machines architecture. Agents can reside in states, have memory and operate according to rules. States change by using transition functions. Also, communicating X-machines are used to communicate using messageboards managed as part of the agent functions. The model will run through a predefined number of iterations where the agents interact with each other. Every iteration a states file will be generated containing the memory for each agent. This figure shows the transition functions between states accessing memory, and sending and receiving messages to a messageboard.

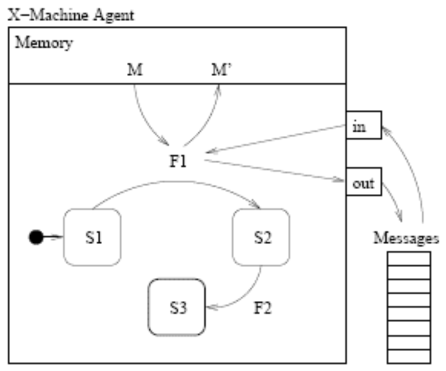


Figure 1. X-Machine agent

Agile and traditional software development techniques

Traditionally, and most commonly, the development of software follows a methodology known as the waterfall method. This entails a strict pre-planned sequence of steps that can take months or even years for very large software development projects. The first stage captures the requirements of the client, secondly, the requirements are analysed in detail and thirdly the software is designed and documented. At this point, no actual software development has taken place. The subsequent stage develops the whole software system as defined and then the software is tested. Finally, the software is delivered and maintained.

Due to the inflexibility of the separate stages this method does not easily allow changes in requirements, which are fixed early in the project and as these stages are often long the client's requirements can change over the course of the project. The result of this could be the delivery of an obsolete system or the high cost in terms of time and money of going back and changing requirements.

Agile methods, in contrast, produce small completely developed and tested features every few weeks. The emphasis is on obtaining the smallest workable piece of functionality to deliver business value early, continually improving it and adding further functionality throughout the life of the project. If a project being delivered under the waterfall method is cancelled at any point up to the end, there is nothing to show for it beyond a huge resources bill. With the Agile method, being cancelled at any point will still leave the customer with some worthwhile code that is likely to have already been put into live operation.

One element of an Agile programming project is Extreme Programming (XP) (Beck, 1999), which is intended to improve software quality and responsiveness to changing customer requirements. XP has some basic practices that include, test first programming, pair programming, small frequent releases, continuous integration and collective code ownership. Pair programming is fundamental to XP and it encourages discussion and continuous review, minimises mistakes, pools intellect, and promotes knowledge sharing.

Knowledge processes and transactive memory

Teams are used for excessive task complexity, when errors lead to severe consequences or in situations where collective insight is required. They are commonly used in most organisations and also in many other walks of life to achieve results beyond individuals' capability. For team cognition to emerge the task requirements of the team must be sufficiently complex for cognitive interdependence to be essential (Akgun et al., 2005, Zhang et al., 2007). To have cognitive interdependence with another team member an individual needs to know about that teammate's knowledge, have the ability to anticipate how the teammate will behave and have trust in the influence that the teammate will exert on the outcome of the shared task (Akgun et al., 2005).

Transactive memory is an individual's beliefs relating to "who knows what" in their team (Wegner, 1995) and a transactive memory system (TMS) is the transactive memory, communication, and coordination at team level (Faraj and Sproull, 2000). A TMS can help team members: compensate for each other, reduce redundancy of effort, share cognitive labour, facilitate sharing of knowledge, and allocate resources. Studies have shown that a good TMS is positively related to team performance (Akgun et al., 2005, Austin, 2003, Lewis, 2004, Littlepage et al., 2008, Zhang et al., 2007) with accuracy of transactive memory being found to be the most significant predictor of team performance (Austin, 2003). The maturation of a TMS will be influenced by the initial conditions of the team and the task processes that the team undertakes.

Transactive memory within a team is constantly changing in response to greater familiarity with teammates and also the changing knowledge and expertise within the team. This highlights issues such as accuracy, consensus and completeness. As teammates work together, their beliefs relating to their teammates are likely to become more complete (Cannon-Bowers and Salas, 2001). As transactive memory in the team becomes more accurate, complete, and consensus is achieved the team is described as having convergent expectations (Hollingshead, 2001). Convergent expectations occur when team members have cognitive interdependence and accurate beliefs relating to each other's knowledge resulting in efficient fulfilment of the task. A team that has a fully convergent, or perfect, TMS will demonstrate the most efficient processing of tasks entering the group. Tasks will be allocated to experts and information will be shared as required for the optimum performance of the team. Recently it has been identified that there is a cyclical relationship between the developing TMS and team knowledge where each affects the other (Brandon and Hollingshead, 2004) and both the knowledge and TM not only grow, but change.

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

Eijk (t) represents the level of expertise of team member j at time time t in area k according to team member i, eijk (t) ≥ 0.

The TMS of the team at time t, is:

T (t) = (T1 (t), TN (t)) where Ti (t) = ((ei11 (t), ei1M (t)), (eiNM (t)), eiNM (t)))

As the knowledge 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. 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, in other words maturation of the TMS, 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).

As team members gain experience working together and become more familiar with each other their ability to recognise knowledge and expertise increases (Lewis, 2004). Lewis found that initial differentiation was positively related to TMS emergence and this effect was stronger when the team members had some familiarity with each other before entering the team.

With heterogeneity of knowledge team members need to know who knows what, implying that TMS is more effective when differentiation is larger. Baumann and Bonner argued that the size of the performance benefits deriving from transactive memory will be greater when group members have large differentiation because it facilitates the recognition, utilisation and impact of team member expertise (Baumann and Bonner, 2004). However, extreme differentiation may be detrimental to team performance as some overlap in expertise allows effective communication and better understanding of each other's knowledge and expertise (Peltokorpi, 2008).

TMS is likely to be less useful when the knowledge is homogeneous (Brandon and Hollingshead, 2004). With very low differentiation, each member of a team would have no incentive to use other members of the team for knowledge therefore there would be low cognitive interdependence (Palazzolo et al., 2006) and no motivation to develop a TMS.

Inevitably there is some disagreement regarding the impact that differentiation of knowledge has on performance within a team. It is suggested that diversity lowers the capability for coordination and cooperation within the team and this improves for more homogenous teams (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.

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 (Littlepage et al., 2008, Baumann and Bonner, 2004). 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.

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 found evidence to suggest that knowledge structures with low differentiation resulted in higher performance for intellectual tasks (Gupta and Hollingshead, 2010).

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 (TMS) 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., 2007, 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. Much of the literature on TMS makes the assumption that more differentiation is more beneficial to teams (Wegner, 1995, Austin, 2003) but Hollingshead suggests that there may be circumstances where differentiation may hinder group performance.

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. 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. 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. This model will examine the behaviour of performance, 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 discrete knowledge. For both of those scenarios there will be a condition each for no initial TMS and complete TMS. The model will have pair programming versus solo programming conditions and small task versus large tasks conditions.

Hypothesis 1. Initial differentiation will have different effects on TMS accuracy and differentiation for different working conditions.

Hypothesis 2. Initial TMS will have different effects on TMS accuracy and differentiation for different working conditions.

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

Hypothesis 4. XP factors will moderate the relationship between initial differentiation and performance. Lower initial differentiation will result in higher performance for pair programming conditions and lower performance for solo conditions.

METHOD

This model is written in FLAME. Each simulation runs over a number of iterations as specified at the start of the simulation. For each simulation there are a number of predefined variables such as number of agents (team size), range of knowledge allowed, task size, working mode and other variables that relate to communication and administrative processes.

Each agent has pairs of numbers in memory. In each pair one number represents a 'chunk' of knowledge and the other represents the level of expertise in that knowledge. This knowledge can grow as the agent learns and the expertise levels can increase as the agent gains experience. Each agent's memory also holds information relating to the knowledge and expertise of the other agents in the simulation. This memory represents TM and this can also develop over the course of a simulation as the agent learns about the other agents. Each iteration, each agent is given a task consisting of numbers representing the knowledge required to complete that task. Each agent values their task in relation to their own knowledge and expertise, and also their beliefs about the knowledge and expertise of other agents, and may decide to offer their task to another agent. They do this if they believe that the agent will gain more team value doing the task than them. Each agent then chooses from the tasks, if any, that have been offered and decides which to keep and reject based on the value of them doing each task. This results in tasks being redistributed based on not only knowledge and expertise, but also beliefs relating to other agents' knowledge and expertise. After task redistribution each agent increments the expertise level for each of the numbers in their task that they hold in memory by a random number between 0 and 1. This represents them carrying out the task and becoming more expert in those areas. They also share some information relating to their task with other random agents to allow them to update their TM.

Each iteration produces measures of performance, knowledge and transactive memory. A controller agent aggregates the output from each agent and calculates team level results. The task values for team members are aggregated to give a score per iteration. The optimum performance score is also calculated as if the tasks were perfectly distributed in the team for maximum performance. The ratio of aggregate score and optimum aggregate score is used as the measure of performance. 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. TMS accuracy is the level of agreement of the team TMS with the actual knowledge of each team member. The parameters used for this model were, work mode, task size, initial TMS and initial differentiation. They are shown in the following table.

Table 1: Parameters for model

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

The set of conditions for this model is detailed in the following table. For each condition 16000 task elements were delivered to the team, which defined the number of iterations for each condition. Each condition was simulated 50 times.

Table 2: Description of conditions

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

RESULTS

Table 3 provides the descriptive statistics and the correlations among the constructs in the model giving some preliminary overall indications of the behaviour of the model. It shows that working mode (coded 0 for solo programming; 1 for pair programming) is positively, significantly correlated to performance indicating that pair programming is highly positively correlated with performance as expected. Task size (coded 0 for large tasks; 1 for small tasks) is significantly positively correlated with differentiation and TMS accuracy but not performance. The relationships between differentiation, TMS accuracy and performance 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 factors of initial TMS and initial differentiation, both are significantly negatively correlated with performance indicating that no initial TMS and low initial differentiation have positive relationships with performance. Neither of them have significant relationships with TMS accuracy or differentiation.

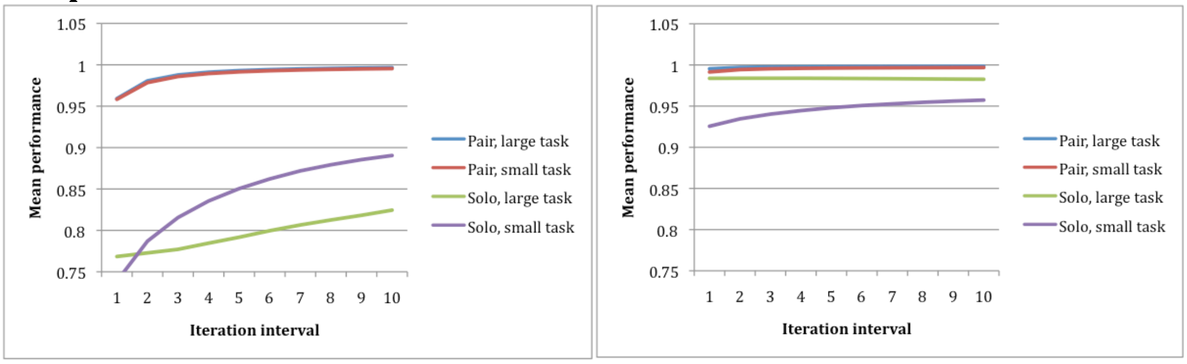
Table 3: Descriptive statistics and correlations - Notes: n=200, *p<0.01

	Variable	Mean	SD	1		2		3		4		5		6	
1	Working mode														
2	Task size														
3	TMS or not														
4	Initial diff'n														
5	Differentiation	40.89	39.42	-.882	*	.123	*	-.009		-.039					
6	TMS accuracy	.96	.03	.888	*	.330	*	-.066		-.009		-.709	*		
7	Performance	.97	.05	.628	*	.001		-.260	*	-.390	*	-.576	*	.523	*

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 performance, 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.

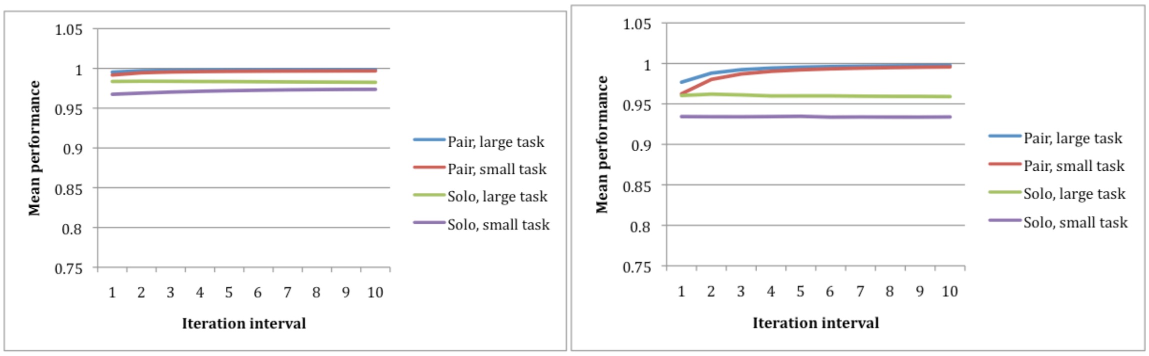
The following four graphs show the model behaviour over time for performance. For pair programming the existence of an initial TMS seems to have little effect on performance, and high initial differentiation slightly lowers performance. For solo conditions the presence of an initial TMS and low initial differentiation both have a substantial positive effect on performance. This means that conditions with low initial differentiation and an initial TMS result in highest performance for solo conditions and those with high initial differentiation and no initial TMS generate the lowest. For pair programming conditions large tasks generate slightly greater performance than small tasks however the significance of this will be tested by the t-tests. For solo conditions, large tasks generate significantly greater performance than small tasks other than conditions when initial differentiation was high and there was no initial TMS where performance is substantially higher for small tasks. The results shown in these graphs support hypotheses 3 and 4.

Graphs for performance over time



Low initial differentiation **High initial differentiation**

Figure 2: Performance, no initial TMS

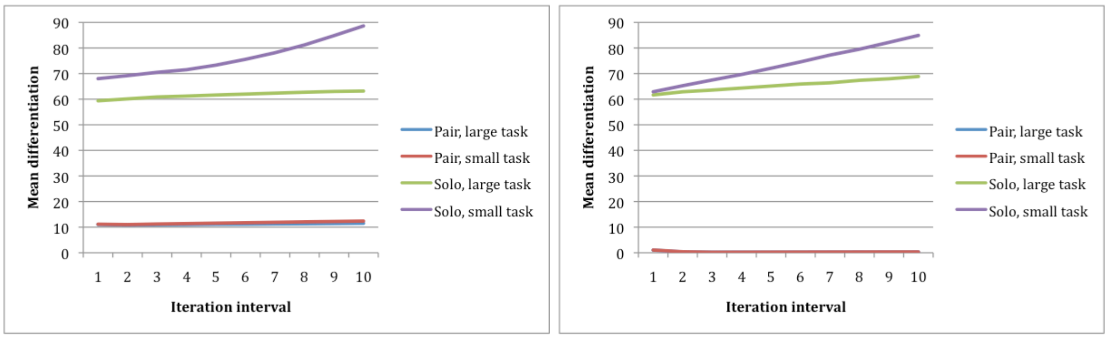


Low initial differentiation **High initial differentiation**

Figure 3: Performance, complete initial TMS

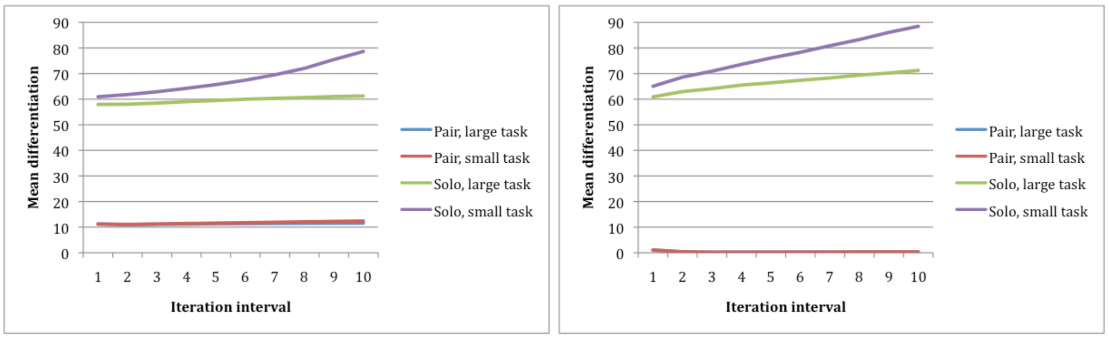
The following four graphs show the model behaviour over time for differentiation. It can be seen that differentiation over time for pair programming conditions remains level for the whole simulation with lower differentiation over time for conditions with high initial differentiation. Task size appears to have little effect for pair programming conditions. For solo conditions differentiation is generally higher than for pair programming and rises over time, rising faster for large task conditions. There does not seem to be much difference between high and low initial differentiation, and initial TMS conditions. This will be tested by the t-tests following. The results in these graphs support hypotheses 1 and 2.

Graphs for differentiation over time



Low initial differentiation **High initial differentiation**

Figure 4: Differentiation, no initial TMS



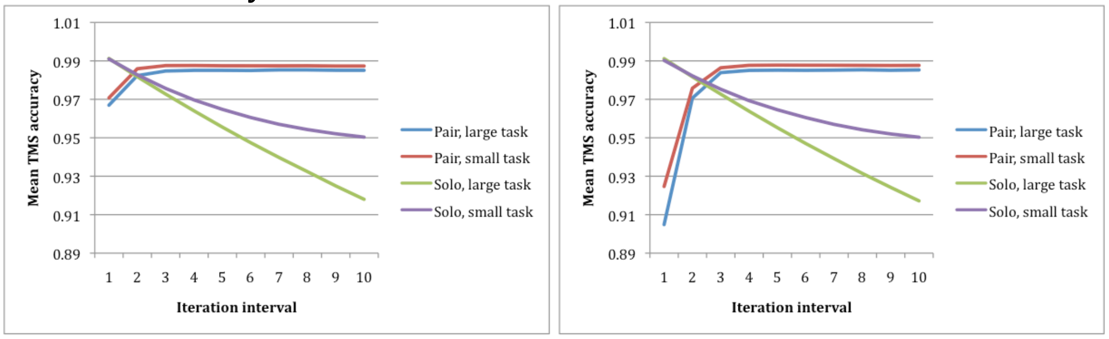
Low initial differentiation **High initial differentiation**

Figure 5: Differentiation, complete initial TMS

The following graphs show the model behaviour over time for TMS accuracy. TMS accuracy, for pair programming conditions, rises quickly and remains level, whereas for solo conditions it continues to drop with the reduction being greater for large tasks. There appears to be little difference between the high and low initial differentiation conditions, although the TMS accuracy takes longer to develop for high initial differentiation for pair conditions. For solo conditions TMS accuracy appears to drop lower for conditions with an initial TMS.

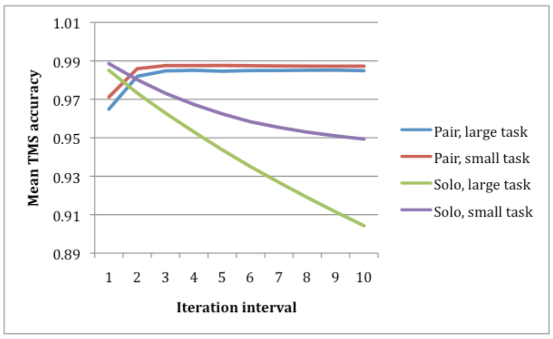
Within solo and pair programming conditions small tasks result in higher TMS accuracy than large tasks for every condition of initial differentiation and TMS. Again, t-tests will determine if this is significant. The results in these graphs support hypotheses 1 and 2.

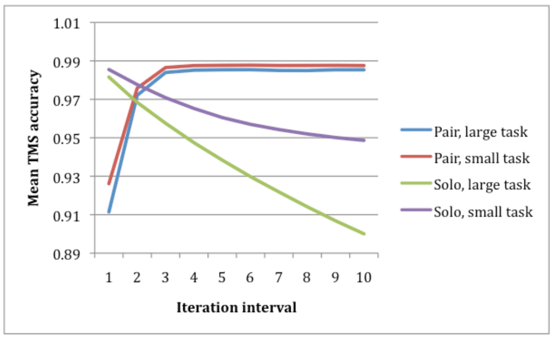
Graphs for TMS accuracy over time



Low initial differentiation **High initial differentiation**

Figure 6: TMS accuracy, no initial TMS





Low initial differentiation **High initial differentiation**

Figure 7: TMS accuracy, 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 4) it can be seen that low initial differentiation generates higher performance for every condition of working mode and task size however by looking at the graphs above, and the differences in means in the table, this effect is greater for solo working conditions. Task size also has a greater effect on solo conditions than for pair conditions with the difference in performance for solo conditions being greater for high initial differentiation.

Regarding differentiation over time, low initial differentiation generates higher differentiation over time for pair programming conditions. For solo conditions there is less significance but in contrast high initial differentiation has a positive effect on differentiation for solo large tasks. Although it is not clear from the graphs, the t-tests determine that TMS accuracy is higher for pair programming when initial differentiation is high, and 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 and although overall performance is lower this effect is larger for solo programming. Differentiation over time is unaffected by initial differentiation for solo programming, but high initial differentiation reduces differentiation over time for pair programming conditions.

Finally there is an inverse effect of initial differentiation on TMS accuracy where low initial differentiation has a positive effect for pair programming and a negative one for solo programming. These results support hypotheses 1 and 4.

The table on the next page (Table 5) shows that the existence of an initial TMS results in higher performance for all working conditions apart from pair programming with small 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. 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.

Table 4: Independent t-tests for initial differentiation, separated by working conditions.

	Work mode	Task size	Initial differentiation	Mean	SE	t	df	
Performance	Solo	4	High	0.912	2.2 E-3	22.53	126.29	***
			Low	0.965	8.3 E-4			
		20	High	0.892	6.8 E-3	13.35	99.03	***
			Low	0.983	8.8 E-4			
	Pair	4	High	0.996	2.7 E-5	45.99	107.24	***
			Low	0.997	5.5 E-6			
		20	High	0.997	5.8 E-5	21.62	102.77	***
			Low	0.998	8.0 E-6			
Differentiatio n	Solo	4	High	86.68	2.75	-.814	198	
			Low	83.63	2.56			
		20	High	70.06	2.22	-2.72	191.23	**
			Low	62.22	1.83			
	Pair	4	High	0.313	3.8 E-3	151.09	99.45	***
			Low	12.38	8.0 E-2			
		20	High	0.29	3.5 E-3	159.68	99.49	***
			Low	11.54	7.0 E-2			
TMS accuracy	Solo	4	High	0.949	1.3 E-4	2.11	198	*
			Low	0.950	1.2 E-4			
		20	High	0.909	8.7 E-4	2.24	189.96	*
			Low	0.911	7.1 E-4			
	Pair	4	High	0.988	5.4 E-5	-4.09	198	***
			Low	0.987	5.3 E-5			
		20	High	0.985	8.8 E-5	-2.38	198	*
			Low	0.985	1.0 E-4			

Notes: n=200, *p<0.05, **p<0.005, ***p<0.001

Hypothesis 3 is partially supported as both pair programming and small tasks have to be in use before the existence of an initial TMS is not significant. Also, hypothesis 2 is partially supported as differentiation over time is unaffected by the presence of an initial TMS but there is an inverse relationship between initial TMS and TMS accuracy for work mode.

Table 5: Independent t-tests for initial TMS separated by working conditions

	Work mode	Task size	Initial TMS	Mean	SE	t	df	
Performance	Solo	4	No TMS	0.924	3.4 E-3	-7.60	161.74	***
			TMS	0.954	2.0 E-4			
		20	No TMS	0.903	8.0 E-3	-8.33	103.75	***
			TMS	0.971	1.2 E-3			
	Pair	4	No TMS	0.996	7.2 E-5	-1.33	191.29	
			TMS	0.996	5.9 E-5			
		20	No TMS	0.997	9.2 E-5	-4.92	138.21	***
			TMS	0.998	4.2 E-5			
Differentiation	Solo	4	No TMS	86.76	2.54	0.854	198	
			TMS	83.55	2.76			
		20	No TMS	66.02	2.07	-.082	198	
			TMS	66.26	2.08			
	Pair	4	No TMS	6.34	0.61	-.01	198	
			TMS	6.35	0.61			
		20	No TMS	5.91	0.56	-.022	198	
			TMS	5.92	0.57			

TMS accuracy	Solo	4	No TMS	0.950	9.3 E-5	9.11	191.290	***
			TMS	0.949	1.1 E-4			
		20	No TMS	0.918	1.7 E-4	50.90	71.89	***
			TMS	0.902	2.5 E-4			
	Pair	4	No TMS	0.987	5.6 E-5	1.11	198	
			TMS	0.987	5.4 E-5			
		20	No TMS	0.985	9.7 E-5	.356	198	
			TMS	0.985	9.8 E-5			

*Notes: n=200, ***p<0.001*

DISCUSSION

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, Lewis, 2004, Hollingshead, 2000, Hollingshead, 2001, 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., 2007, 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. 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.

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 performance than high initial differentiation. This advantage was stronger for solo programming than for pair programming demonstrating a moderation effect. This is in contradiction to Lewis (Lewis, 2004) who found that teams with initially diverse knowledge produced higher team performance. In addition, as theory predicts, results show that the existence of an initial TMS gives higher performance for all working conditions apart from pair programming with small tasks when there is no significant difference. So, according to the model, XP factors compensate for the advantage that the initial TMS bestows on the team, but only if pair programming is used in conjunction with small tasks. Hypothesis 4 is supported and hypothesis 3 is partially supported as both pair programming and small tasks have to be in use before the existence of an initial TMS is not significant.

The positive effect of low initial differentiation can be explained by the functioning of the model. Performance is measured as the ratio between actual performance and optimum performance. As each of the agents has identical knowledge the potential detriment from allocating tasks to the wrong agent is reduced. 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, however, pair programming together with small tasks compensates for this as both practices bring the task value closer to the optimum. Pair programming does this by pooling the resources of two programmers, and small tasks by focussing the allocation of tasks to the most qualified agent.

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.

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., 2007, Lewis, 2004, Liang et al., 1995) which states that groups with an initial TMS will have higher quality TMSs and perform better. Hypothesis 1 is supported and hypothesis 2 is partially supported as differentiation over time is unaffected by the presence of an initial TMS, but there is an inverse relationship between initial TMS and TMS accuracy for work mode.

In 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, for new pair programmers, 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.

In summary, performance was increased with lower initial differentiation and an initial TMS, however, this reliance on the initial TMS was reduced by XP factors to a degree. 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 and initial differentiation had no effect on differentiation over time and they both had the effect of reducing TMS accuracy over time.

CONCLUSIONS

This model introduced 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. 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 showed that team members with similar knowledge, who had worked together before, would result in higher performance. This is less important when using XP techniques, which seem to compensate for this.

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, for pair programmers, 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 and diversity of initial knowledge will have no effect on diversity of knowledge over time but they both are likely to reduce the accuracy of the team members' TMS over time to a greater extent.

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 performance, however, this simulation has demonstrated that the use of XP techniques can mitigate this disadvantage.

Acknowledgements: This work is jointly funded by the Economic and Social Research Council (ESRC), and the Engineering and Physical Sciences Research Council (EPSRC).

References

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.

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

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.

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

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.

BECK, K. 1999. Exteme Programming Explained, Addison-Wesley.

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.

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

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

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

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

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.

GEORGHE, M. E. A. 2001. Computational Models of Collective Behavior. Trends in Cognitive Sciences, 9, 10-15.

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.

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.

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. 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.

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.

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.

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.

PELTOKORPI, V. 2008. Transactive Memory Systems. Review of General Psychology, 12, 378-394.

PELTOKORPI, V. & MANKA, M. L. 2008. Antecedents and the performance outcome of transactive memory in daycare work groups. European Psychologist, 13, 103-113.

POGSON M, S. R., QWARNSTROM E, HOLCOMBE M 2006. Formal agent-based modelling of intracellular chemical interactions. Biosystems, 85, 37-45

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.

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.

WEGNER, D. M. 1995. A computer network model of human transactive memory. Social Cognition, 13, 319-339.

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-+.

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.