2007

Embracing knowledge and behaviour management to improve performance of software intensive projects

Ricardo Peculis
University of Wollongong, peculis@uow.edu.au

Derek Rogers
University of South Australia, derek.rogers@drderekrogers.com

Peter Campbell
University of South Australia, pcampbel@uow.edu.au

Embracing knowledge and behaviour management to improve performance of software intensive projects

Abstract

Many reasons have been offered to explain why software projects fail. Still, software intensive projects often present schedule delays, cost overruns and delivering products with reduced functionality. This paper argues that lack of knowledge is yet another factor that causes projects to under perform, which in turn drives undesirable social behaviour that worsens the situation. Software intensive projects develop solutions highly dependent on software that should satisfy a need. The engineering process to develop such complex solutions comprises of a series of transformations that transform products from one domain into products in another domain, requiring knowledge pertinent to both input and output domains. Complex products call for knowledge that may not be available, is often distributed amongst people and organisations and involves learning and cooperation. The engineering of software intensive systems, in particular, frequently adopts techniques that abstract complexity and hide details that reveal, too late, a lack of knowledge in earlier transformations. When knowledge is not applied as required, distortions propagate throughout transformations producing solutions that will not satisfy the need and require rework that increases costs and delays schedules. Software intensive projects may fail because of lack of technical knowledge to engineer the solution, and lack of management knowledge to recognise and plan for this deficiency. When projects are not performing as expected and interests are at risk, decisions are made within constraints that are unlikely to address the lack of knowledge, which often produces undesirable social behaviour that decreases motivation, discourages learning and cooperation and worsens the situation. The paper concludes proposing the adoption of project specific knowledge and behaviour management to improve the performance of software intensive projects.

Keywords

software, intensive, projects, behaviour, embracing, management, knowledge, improve, performance

Disciplines

Engineering | Physical Sciences and Mathematics

Publication Details

Peculis, R., Rogers, D. & Campbell, P. (2007). Embracing knowledge and behaviour management to improve performance of software intensive projects. Systems Engineering/Test and Evaluation Conference: Proceedings (pp. 1-8). Systems Engineering Society of Australia.

Embracing Knowledge And Behaviour Management To Improve Performance Of Software Intensive Projects

Ricardo Peculis
Computer Sciences Corporation;
SEEC University of South Australia,
ricardo.peculis@postgrads.unisa.edu.au

Derek Rogers
SEEC University of South Australia,
derek.rogers@unisa.edu.au

Peter Campbell
SEEC University of South Australia,
peter.campbell@unisa.edu.au

Abstract. Many reasons have been offered to explain why software projects fail. Still, software intensive projects often present schedule delays, cost overruns and delivering products with reduced functionality. This paper argues that lack of knowledge is yet another factor that causes projects to under perform, which in turn drives undesirable social behaviour that worsens the situation. Software intensive projects develop solutions highly dependent on software that should satisfy a need. The engineering process to develop such complex solutions comprises of a series of transformations that transform products from one domain into products in another domain, requiring knowledge pertinent to both input and output domains. Complex products call for knowledge that may not be available, is often distributed amongst people and organisations and involves learning and cooperation. The engineering of software intensive systems, in particular, frequently adopts techniques that abstract complexity and hide details that reveal, too late, a lack of knowledge in earlier transformations. When knowledge is not applied as required, distortions propagate throughout transformations producing solutions that will not satisfy the need and require rework that increases costs and delays schedules. Software intensive projects may fail because of lack of technical knowledge to engineer the solution, and lack of management knowledge to recognise and plan for this deficiency. When projects are not performing as expected and interests are at risk, decisions are made within constraints that are unlikely to address the lack of knowledge, which often produces undesirable social behaviour that decreases motivation, discourages learning and cooperation and worsens the situation. The paper concludes proposing the adoption of project specific knowledge and behaviour management to improve the performance of software intensive projects.

INTRODUCTION

Many reasons have been offered to explain why software projects fail. Still, software intensive projects often present schedule delays, cost overruns and the delivery of products with reduced functionality. This paper argues that lack of knowledge is yet another factor that causes projects to under perform, which in turn drives undesirable social behaviour that worsens the situation.

Software intensive projects develop solutions highly dependent on software that should satisfy a need. The engineering process to develop such complex solutions comprises of a series of transformations that transform products from one domain into products in another domain, requiring knowledge pertinent to both input and output domains. Complex products call for knowledge that may not be available, is often distributed amongst people and organisations and involves learning and cooperation. The engineering of software intensive systems, in particular, frequently adopts techniques that abstract complexity and hide details that reveal, too late, a lack of knowledge in earlier transformations. When knowledge is not applied as required, distortions propagate throughout transformations producing solutions that will not satisfy the need and require rework that increases costs and delays schedules.

Executive support, customer involvement and experienced project managers, together with clear business objectives, top the list of factors that have been reported as key contributors to success of software intensive projects (Standish, 1998, 2001). Lower in the list, but still in the top ten, are also stable requirements and competent staff. When these factors are put into context, their level of influence and importance may be questioned. Software intensive projects are business enterprises and should meet objectives for customers and providers. The involvement of competent staff in both customer and provider organisations is fundamental to establish requirements and contractual conditions that should satisfy both parties. Executive support creates enabling conditions, while experienced project managers plan the tasks, monitor them and intervene when deviations occur. Staff and managers must possess knowledge and skills to execute their tasks.

Software intensive projects may fail because of lack of technical knowledge to engineer the solution, and lack of management knowledge to recognise and plan for this deficiency. When projects are not performing as expected and interests are at risk, decisions are made within constraints that are unlikely to address the lack of knowledge, which often produces undesirable social behaviour that decreases motivation, discourages learning and cooperation and worsens the situation.

The purpose of specific knowledge management is to identify the knowledge and skills required to engineer the solution, to ascertain what of this knowledge and skills is available and what is not; and to develop a plan to harvest distributed knowledge and to acquire the knowledge that is missing. Specific behaviour management aims to observe the social behaviour of teams and organisations, and to develop strategies to promote constructive behaviour that fosters learning and cooperation. The paper concludes by proposing the adoption of project specific knowledge and behaviour management to improve the performance of software intensive projects.

SOFTWARE INTENSIVE PROJECTS

Software intensive projects apply the process of engineering to develop products highly depend on software. As the scale[1], life cycle[2] and interdependency of internal components of engineering projects increase, their complexity also increases (Aslaksen, 1996). Complexity refers to the limitation of the human brain[3] in the visualisation and manipulation of systems that involve a large number of non linear interactions between components and parameters. To be able to handle complex objects, the brain removes what it judges to be less important or relevant but even with such a "fuzzy picture" the brain is capable of manipulating complex situations with complex entities.

To handle large projects, partitioning is essential. From the design perspective, functional requirements are partitioned in the most logical way, while management will break up the work in a most efficient work breakdown structure. While project management deals with physical entities such as the coordination and management of the project activities, engineering management deals with functional entities (Aslaksen, 1996). Physical entities are more tangible and are less difficult to understand and communicate, while functional entities are more difficult to understand and manipulate, due to the manner in which the brain deals with complex tasks.

Engineers and managers are subject to limitations in dealing with complex entities and situations. Consequently, mental pictures created by the cognitive process are a simplified version of reality. As the brain removes what it finds less important or unnecessary, the mental picture is influenced by knowledge, emotions, beliefs and preferences. As the result, the mental picture is likely to be incomplete, distorted and biased. The accuracy of mental pictures and the way they are shared by the team is of fundamental importance for the success in achieving the common goal (Newell et al, 1990).

[1] Scale refers to size, cost and number of components (Aslaksen, 1996).

[2] Life cycle refers to environmental and temporal dependencies (Aslaksen, 1996).

[3] Studies show that the brain has difficulty in manipulating entities which comprise more than seven elements (Aslaksen, 1996).

Experience shows that the engineering of products with a large software component is not an easy task. The challenge can be associated to the fact that for the development of software, functional products extend longer throughout the project, and are more difficult to communicate about and estimate than the hardware components. The engineering of software intensive systems, in particular, frequently adopts techniques that abstract complexity and hide details that then reveal problems at late stages of the project.

In addition to the complexity of the multitude of interacting technical components, large software intensive projects are likely to be executed by multiple teams and organisations, increasing the complexity of the project.

The Acquisition Model

The aim of the acquisition process is to acquire a solution that will satisfy a need, and software intensive acquisitions, in particular, acquire solutions that depend on engineering projects to develop products with software components. The acquisition process can be represented by a set of products and transformations within the acquisition space, comprising of three subspaces: Situation, Problem and Solution (Peculis et al, 2007). The acquisition originates in the Situation domain, where a need exists and can be expressed by a set of products pertinent to that space. The need leads to the definition of the problem as another set of products in the Problem domain. The solution that resolves the problem comprises a set of products in the Solution domain. When the solution is applied to the Situation it should satisfy the need. Transformations consist of tasks that apply engineering processes transforming products from one domain into products in another domain. The execution of these tasks requires knowledge associated with the domain space of the input and output products. The quality of the products depends on people's knowledge and skills and their motivation to execute the task.

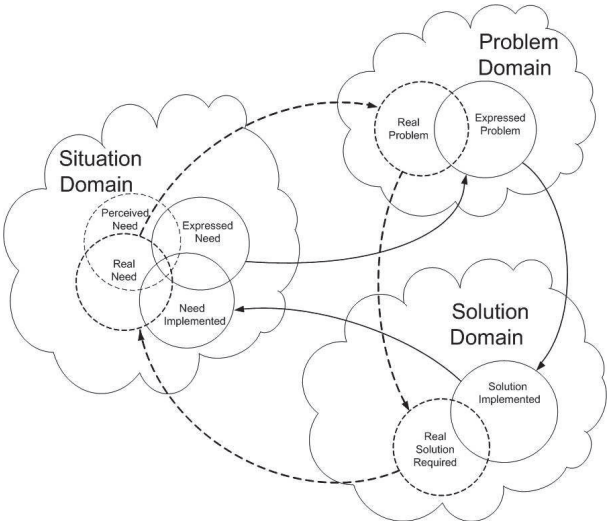

Figure 1: Transformations between Domain Spaces.

Ideally, transformations should occur without distortions. The need should be accurately expressed, the problem well defined, the solution correctly implemented to address satisfactorily the need. In reality, distortions occur because people within social organisations, limited by their own knowledge, skills, cognition, emotions, moved by personal and corporate goals and bounded by their roles in the social organisation, are responsible for the execution of transformations between and within domains. Distortions are carried from one domain to another and propagate with a "Chinese Whispers" effect (Peculis et al, 2007).

The effectiveness of the acquisition process indicates how well the acquisition achieves its goals and efficiency is measured by how much waste is produced in the process. From Figure 1, effectiveness indicates how close the implemented solution came to fulfilling the real need, i.e. the unbroken line circles coinciding with the dotted circles, and efficiency shows how much effort, in the form of enabling tasks and rework, is required to bring the implemented solution to meet the real need. Lack of knowledge and skills, or lack of applying knowledge and skills available, are some of the causes of low effectiveness and efficiency in software-intensive acquisitions.

THE INFLUENCE OF KNOWLEDGE

The need for knowledgeable, talented and motivated people to develop software products is undeniable (DAF, 2000) (SEI, 2001). Knowledge is not only of fundamental importance to providers that formulate the problem and engineer the solution, but also to customers in their ability to communicate the need (DAF, 2000). Lack of management knowledge also has been reported as one of the causes of failure of software projects (Perkins, 2006). Lack of knowledge is thus another factor that causes projects to under perform.

Distributed and Incomplete Knowledge

No single individual or organisation possesses all knowledge, nor has access to all information required to specify, design and implement complex software intensive systems. Individuals and organisations are likely to be more knowledgeable on subjects that are pertinent to the domain where they operate. Distributed knowledge often leads to incomplete information and creates different perceptions and mental models. Distributed and unshared knowledge turn teams and organisations away from common understanding, common goals, and affects the system effectiveness and efficiency.

The effects of incomplete information and distributed knowledge combined with less than ideal skills create distortions that propagate through specification and design documents, resulting into incomplete or even wrong information when they reach the software domain. Problems that are not addressed and resolved in their domain of origin will propagate through the acquisition space to other subspaces where the required knowledge to address the problems is inadequate or non-existent. These distortions become defects that individuals, teams and organisations will try to back loop and correct, and are drivers for management intervention.

The larger the number of organisations in the system, the greater these effects of distributed knowledge become. The acquisition process, and consequently the software intensive project, ought to impel the acquisition organisation, as a collection of several organisations, to harvest and apply collective knowledge. Collective knowledge is an emergent property, whose effectiveness should be maximised by the organisational design. The distributed nature of knowledge in software intensive acquisitions is a fact that must be inserted into the organisational design.

THE INFLUENCE OF BEHAVIOUR

Although knowledgeable people are great assets for the success of software intensive projects, that may not be enough. People need to be motivated to do the job, which often requires doing activities that are not specifically included as part of the original task. Depending on the complexity of the task that may include acquiring new skills, learning new subjects, cooperating, teaching and helping others. Motivated people perform beyond their formal duties.

Depending on their behaviour[4] style, people respond differently to situations that are presented to them, some producing better results than others. Managers, for example, when facing a late project, may force the team to work harder, or may offer help to address what is causing the delay. Certainly, these actions would cause different reactions and a different response from each individual team member. While constructive attitudes are likely to motivate the team, aggressive behaviour tends to decrease motivation.

Problem-Solving Groups

Software intensive projects can be considered as problem-solving groups. Problem-solving groups have two principal objectives: (1) quality of results, that requires the maximum utilisation of resources brought by each individual member of the group; and (2) acceptance of the solution, that implies a high level of motivation for carrying out the group's decision (Hoffman, 1979). These objectives create pressure on each member of the group to decide between bringing their unique, and possibly controversial, contribution to the problem-solving task, or to conform to group's decisions (Hoffman, 1979).

The effectiveness of problem-solving groups depends on the group's interaction style, which can be constructive, aggressive or passive (Cooke & Szumal, 1994). Groups with a constructive style tend to maximise the

[4] Behaviour refers to how people react in response to a stimulus.

contribution of each group member, fostering cooperation between the group members to achieve both individual and group objectives. Members of groups with a passive style behave in ways that promote their security and acceptance, avoiding conflict, limiting information sharing, and allowing themselves to become dominated by the group. In the aggressive group style, members of the group will behave to promote their status and position to fulfil their own needs; members also tend to demonstrate their competence doing things perfectly, without seeking help or helping others and often losing sight of the group's goals.

Across problem-solving groups, solution quality increases when the group shows a constructive interactive style and decreases with passive interaction style; and the acceptance of the solution increases with constructive interaction style and decreases with both passive and aggressive interaction styles; aggressive interaction has been shown to be unrelated to the quality of the solution (Cooke & Szumal, 1994).

The Influence of Stakeholders

Another important aspect to be observed is the influence that stakeholders have over the project. Senior management, whether from customer or provider organisations, have high influence over the project and are capable of support or ending the project. Senior managers are influenced by other stakeholders that are external to the project, like politicians, the media, marketing personnel, competitors, etc. These influences can also support or end the project. Internal stakeholders, such as project managers, senior engineers and natural leaders can also influence the success or failure of the project.

The increase in the level of abstraction and extended design of software intensive projects expands physical expenditure (e.g. cost, schedule, resources) which delays the physical or tangible results. When projects are not performing as expected and interests are at risk, the nature of engineering and management activities, i.e. functional vs. physical, creates tensions between engineering management and project management (Aslaksen, 1996). Management decisions are then made within constraints that are unlikely to address the lack of knowledge, which often produces undesirable social behaviour that decreases motivation, discourages learning and cooperation and worsens the situation. Experience also shows that in the end, political and business factors are favourable to the sort of project management that holds the control over the project and determines what to do and when. As a result, the design process is compromised, the quality of the solution decreases and originates rework, which in turn increases cost and delays the schedule.

KNOWLEDGE MANAGEMENT

Knowledge management is the process of creating, disseminating and applying knowledge, often represented graphically as a pyramid, shown on Figure 2. The first layer of knowledge management comprises of gathering *data*, which after being interpreted and put into context becomes *information*, the second layer of the process. The third layer focuses on how information is used and becomes *knowledge.* The top two layers include *understanding* and *wisdom*; the former reflects the comprehension and awareness of the consequences that the use of knowledge can bring; the latter is the result of the ability of conscientiously taking advantage of possessing and applying knowledge.

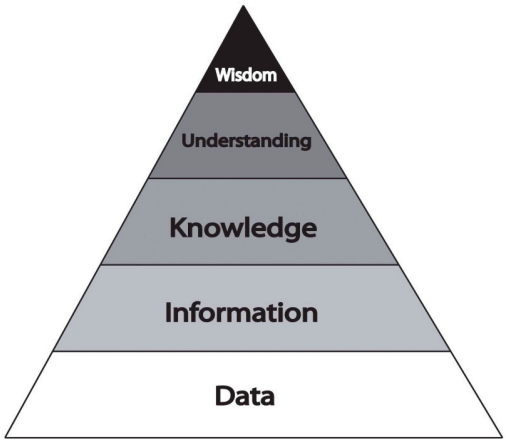

Figure 2: Knowledge Management Philosophy.

Within the context of software intensive projects, the process described by the five layers of the knowledge management pyramid is usually seen as a philosophy rarely put in practice. Although knowledge is a fundamental element for the success of software intensive projects, it is unusual to include a Knowledge Management Plan in the myriad of plans required to start a project. The adoption of project specific Knowledge Management Plan can bring benefits to software intensive projects.

Specific Knowledge Management Plan

The objective of a project specific Knowledge Management Plan (KMP) is to identify the knowledge required to execute the project, the knowledge that is available, and the knowledge that is missing; and to create a plan to harvest existing knowledge and to acquire the knowledge that is not readily available. Through an analysis of user and system requirements, proposed system design and the technology required to execute the project, the KMP identifies the various areas of knowledge, where the existing knowledge is located, and establishes strategies to create a "collective knowledge".

Some of the key aspects included in the KMP are methods and processes for measuring and assessing intangible assets, such as knowledge, skills and experience, and the policies for selecting organisations and staff for the project.

Software intensive projects are often executed by multiple organisations that extend from the customer to prime contractor and subcontractors. Effective results

will emerge when specific knowledge management is applied and extended to all organisations involved in the software intensive acquisition as a whole.

KMP for a Naval Helicopter

The following example, based on the specification of a naval helicopter illustrates the process of identification of areas of knowledge and their dependencies (Peculis et al, 2007). The specified roles for this naval helicopter are: Surveillance (SV), Anti-Ship Warfare (ASW), Anti-Submarine Warfare (ASbW), Search & Rescue (S&R), Support Missions (SM) and Training Support (TrS). Table 1 represents the user's requirements in the form of the aircraft's roles and capabilities expressed by the Operational Concept Document (OCD). The "dots" on Table 1 show the dependencies between the aircraft's roles and capabilities, and correspond to the tasks that need to be executed to prepare the OCD that accurately expresses the customer's need. Each task corresponds to a capability placed within the context of the aircraft's roles, and requires a specific knowledge that combines two domains of knowledge: aircraft's operational roles and aircraft's capabilities. The Human/Machine Interface (HMI) for Surveillance (HMI/SV) and HMI for Anti-Submarine Warfare (HMI/ASbW) comprise two distinctive domains of knowledge. The first combines the knowledge of HMI with the knowledge of Surveillance, while the latter applies the knowledge of HMI into Anti-Submarine Weapons. Similarly, to specify Tactical Data Management (TDM) is required knowledge of TDM/SV, TDM/ASW, TDM/ASbW and TDM/TrS. Table 1 thus suggests that the preparation of the OCD for the naval helicopter of the example is required knowledge of 58 specific areas, determined by the number of "dots" on Table 1.

Aircraft Roles ►	SV	ASW	ASbW	S&R	SM	TrS
Aircraft Capabilities ▼						
HMI for crew of two (HMI)	•	•	•	•	•	•
Aircraft Monitor & Supervision (M&S)	•	•	•	•	•	•
Mission Preparation Support (MPS)	•	•	•	•	•	•
Mission Debrief Support (MDF)	•	•	•	•	•	•
Electronic Navigation (NAV)	•	•	•	•	•	•
Radio Communications (COM)	•	•	•	•	•	•
Surveillance Sensors (SVS)	•	•	•	•	•	•
Electronic Warfare (EW)	•	•	•			•
Tactical Data Management (TDM)	•	•	•			•
Tactical Navigation Mgmt and AFC (TNM)	•			•	•	•
Anti-Ship Weapons (WAS)		•				•
Anti-Sub Weapons (WASb)			•			•

Table 1: OCD: Aircraft's Roles and Capabilities

Table 2 shows the dependencies between the specification of the aircraft's system components and the specification of the aircraft's capabilities. Table 2 corresponds to what is included in the Functional Performance Specification (FPS), prepared by 155 tasks, indicated by the "dots" on Table 2. To specify the requirements for the Radar, for example, it has to be placed within the context of several capabilities (HMI, M&S, MPS, MDS, SVS and TDM), and requires knowledge of the operation of a specific Radar applied to these capabilities. The example suggests that 155 tasks, determined by the number of "dots" on Table 2, are needed to prepare the FPS. The tasks to prepare the OCD and FPS call for the application of 213 specific areas of knowledge.

The process of identification of areas of knowledge continues into the specification and design of the system and the software until the development of software components. For each of the 26 system components, specification and design documents are prepared to address the system (SSS/SSDD[5]), equipment interfaces (IRS/IDD[6]) and human interfaces (HIS/HIDD[7]). The specification and design of each system component (e.g. Radar, EW, Weapons, etc.) calls for specific knowledge of that component in the context of equipment interfaces, human interfaces and the equipment operation within the system. Thus, each system component calls for six specific areas of knowledge to develop the SSS, SSDD, IRS, IDD, HIS and HIDD, which for 26 system components adds the need for a total of 156 areas of knowledge.

The specification and design of the system and its interfaces become the input for the software specification and design (SRS/SDD[8]) for each of the 26 CSCIs[9]. Each CSCI (e.g. Radar, EW, Weapons, etc.) requires specific knowledge for the specification and design of the software, creating a need for 52 areas of knowledge, two (i.e. specification and design) for each CSCI. Finally, with a complete software specification and design, the CSCIs are coded and tested, which

[5] SSS: System/Subsystem Specification; SSDD: System/Subsystem Design Document.

[6] IRS: Interface Requirements Specification; IDD: Interface Design Document.

[7] HIS: Human Interface Specification; HIDD: Human Interface Design Document.

[8] SRS: Software Requirements Specification; SDD: Software Design Document.

[9] CSCI: Computer Software Configuration Item.

should require only knowledge of software development. The software specification and design that allows software developers to do their job thus contain information derived from over 420 areas of knowledge, that were required to develop the OCD, FPS, and SSS, SSDD, IRS, IDD, HIS and HIDD for 26 systems components and CSCIs.

The KPM applies the Knowledge Management philosophy in a specific and practical way to bring the required knowledge into the project. The KMP not only identifies the areas of knowledge required to do the job but also whether the knowledge is available and where it is located. Knowledge of operational aspects of the aircraft, including roles and capabilities, are likely to be available in the Situation domain. It is unlikely for systems engineers to know everything about the Situation domain, and a close cooperation between the users and engineers is often needed. Similarly, due to the fact that equipment manuals and interface documents are often incomplete and misleading, cooperation between systems engineers and equipment manufacturers is needed to specify and design system components. The KMP includes a plan to harvest the available knowledge through workshops and working groups that gather users, engineers and experts together. The KPM also includes plans for formal training and time for self learning.

Capabilities ► / Components ▼	HMI	M&S	MPS	MDS	NAV	COM	SVS	EW	TDM	TNM	WAS	WASb
Dual Data Processor	•	•	•	•	•	•	•	•	•	•	•	•
Data Entry & Display	•	•	•	•	•	•	•	•	•	•	•	•
MDLR	•	•	•	•	•	•	•	•	•	•	•	•
Aircraft Sensors	•	•	•	•	•							
ADC	•	•	•	•	•							
INS/GPS	•	•	•	•	•							
VOR/ILS	•	•	•	•	•							
DME	•	•	•	•	•							
RADALT	•	•	•	•	•							
LF ADF	•	•	•	•	•							
IFF	•	•	•	•		•						
Intercom	•	•	•	•		•						
HF Radio	•	•	•	•		•						
U/VHF Radio	•	•	•	•		•						
Radar	•	•	•	•			•		•			
FLIR	•	•	•	•			•		•			
ESM	•	•	•	•			•	•	•			
Radar Warning System	•	•	•	•			•	•				
Missile Warning System	•	•	•	•			•	•				
Laser Warning System	•	•	•	•			•	•				
CMDS	•	•	•	•				•				
Link-11	•	•	•	•					•			
AFCS	•	•	•	•						•		
Anti-Ship Missile	•	•	•	•							•	
Torpedo	•	•	•	•							•	•
Depth Bomb	•	•	•	•								•

Table 2: FPS: Aircraft's Capabilities and System Components.

BEHAVIOUR MANAGEMENT

Within the context of software intensive projects, the objective of Behaviour Management is to improve the project outcomes through the way that people and or organisations interact. Behaviour expectation driven by organisational culture can drive the level of cooperation in teams and groups; while constructive behaviour is likely to impel cooperation, non-constructive interaction styles can undermine attempts at knowledge exchange and of achieving knowledge management goals (Balthazard & Cooke, 2004). Constructive behaviour enhances the quality and acceptance of the solution produced by problem-solving groups (Cooke & Szumal, 1994). Behaviour Management thus intends to maximise constructive behaviours, promote cooperation and increase the effectiveness of Knowledge Management.

Specific Behaviour Management Plan

The objective of the proposed project specific Behaviour Management Plan (BMP) is to identify the likely behaviour within the project's social environment and create a plan to maximise the effectiveness of the organisation through stakeholder and organisational culture management.

Stakeholder management identifies people and organisations capable of influencing the project and develops strategies to create a culture of constructive behaviours. Stakeholder management attempts to ascertain stakeholder's interests and their likely behaviour, and to identify project risks that could come from stakeholders capable of swaying the project negatively. The BMP develops strategies to educate non-constructive stakeholders, managers and team members to change their interaction styles and attitudes towards more constructive behaviours. Stakeholder management is essential for effective project management (Bourne & Walker, 2003). The Stakeholder Circle (Bourne & Walker, 2006) is a stakeholder management tool to identify, prioritise and developing a stakeholder engagement strategy that better suites the project, or business enterprise, objectives.

The aim of Organisational Culture Management is also to maximise the effectiveness of the organisation by developing an organisational culture that best suites the organisation's objectives. Organisations that deal with software intensive projects and problem-solving groups in general, tend to increase their effectiveness through constructive cultures. Behaviour management tools such as the Organisational Culture Inventory (OCI) (Cooke &

Szumal, 2000) and the Life Style Inventory (LSI) (Cooke & Rousseau, 1987) use surveys to produce qualitative data about organisational culture and the behaviour profile of staff members that are capable of influencing the culture of the organisation. As the result of making people aware of their strengths and limitations, and through appropriate training, cultural change programs aim to reshape the organisation in ways to improve its effectiveness. When applied to software intensive projects, behaviour management should also be extended to all organisations involved in the software intensive acquisition to be able to produce effective results.

CONCLUSION

Experience shows that software intensive projects are prone to schedule delays, cost overruns and functional deficiencies when the product is delivered. The diversity of knowledge required, the functional and abstract nature of software products, together with human limitations to deal with large number of interconnected entities make the development of software intensive systems a challenging endeavour. As the complexity of software systems increase, the complexity of the social organisation required to engineer them also increases.

Software intensive projects are problem-solving groups, and success depends on how mental pictures are created and communicated and the way people interact and apply their knowledge. The chances of success are maximised when people behave constructively rather than aggressively or passively. The success of software intensive projects thus depends on knowledgeable, talented and motivated people. Learning and cooperation is of fundamental importance to acquire missing knowledge and to transform distributed knowledge into collective knowledge. Collective knowledge is an emergent property that should be addressed by the organisational design of software intensive acquisitions.

Knowledge and behaviour influence each other. Lack of knowledge often leads to errors and deficiencies that require rework that delays schedules and increase costs, often causing management intervention that is unlikely to address the root cause of the problem. This kind of aggressive attitude causes other undesirable social behaviour that lowers motivation to learn and cooperate and worsens the situation.

Specific Knowledge Plan and Behaviour Management Plan concepts are proposed to address knowledge and behaviour components of software intensive projects. The former aims to identify the knowledge required and available; and to develop a plan to acquire the knowledge that is missing and to create conditions to enable sharing of distributed knowledge. The latter intends to create conditions that will foster constructive behaviours, learning and cooperation; and to avoid non-constructive behaviours that will move the project away from desirable outcomes.

In conclusion, embracing Knowledge and Behaviour Management, through realistic plans and their effective application is an important element of socio-organisational design that can improve performance and chances of success of software intensive projects. Effective results will emerge when the commitment to Knowledge and Behaviour Management is embraced by all organisations involved in the software intensive acquisition as a whole.

REFERENCES

1. Aslaksen E. W. 1996, The Changing Nature of Engineering, McGraw-Hill Australia, Sydney.

2. Balthazard, P. A., Cooke, R. A., 2004, Organizational Culture and Knowledge Management Success: Assessing The Behavior–Performance Continuum, *Proceedings of the 37th Hawaii International Conference on System Sciences – 2004*.

3. Bourne, L., Walker, D. H. T., 2003, "Tapping the Power Lines: A 3rd Dimension of Project Management Beyond Leading and Managing", *7th Australian International Performance Management Symposium*, Canberra, Australia.

4. Bourne, L., Walker, D. H. T, 2006, "Using a Visualising Tool to Study Stakeholder Influence", *The Project Management Journal*, Vol 37 No. 1, 2006.

5. Cooke, R. A., Rousseau, D. M., 1987, "Thinking and Behavioral Styles: Consistency Between Self-Descriptions and Description by Others", *Educational and Psychological Measurement*, 1987, 47.

6. Cooke, R. A., Szumal, J. L. 1994, "The Impact of Group Interaction Styles on Problem-Solving Effectiveness", *The Journal of Applied Behavioral Science*, 1994, vol. 30, no. 4, Dec 1994, pp. 415-437.

7. Cooke, R. A., Szumal, J. L, 2000, "Using Culture Inventory to Understand the Operating Culture of Organizations", *Handbook of Organizational Culture & Climate,* 2000, pp.147-162, Neal M. Ashkanasy, Celeste Wilderom, Mark F. Peterson, editors.

8. DAF, Department of Air Force, 2000, *Guidelines for Successful Acquisition and Management of Software-Intensive Systems*, Version 3.0, May 2000, viewed 10 Nov. 2003, <http://web.nps.navy.mil/~menissen/mn3309/stsc-guidelines3.0/GSAMv3.pdf>.

9. Hoffman, L. R. 1979, "Applying Experimental Research on Group Problem Solving to Organizations", *The Journal Of Applied Behavioral Science*, vol. 15, no. 3, July 1979, pp. 375-391.

10. Newell, A., Yost, G., Laird, J., Rosenbloom, P., & Altman, E. 1990, *Formulating the problem space computational model.* Paper presented at the 25th Anniversary Symposium, School of Computer Science, Carnegie Mellon University.

11. Peculis, R., Rogers, D., Campbell, P., 2007, "A Task Model of Software Intensive Acquisitions: An Integrated Tactical Avionics System Case Study", *12th Australian International Aerospace Conference*, Melbourne, Australia.

12. Perkins, T. K. 2006, “Knowledge: The Core Problem of Project Failure”, *CrossTalk, The Journal of Defense Software Engineering*, Vol. 19 No. 6, June 2006.

13. SEI, Software Engineering Institute, 2001, People Capability Maturity Model, P-CMM, V2.0, *Carnegie Mellon University*, Pittsburgh, PA.

14. Standish, G 1998, *‘CHAOS: A receipt for success’*, Standish Group, viewed 13 Nov. 2003, <http://www.standishgroup.com/sample_research/PDFpages/chaos1998.pdf>.

15. Standish, G 2001, *‘Extreme Chaos’*, Standish Group, viewed 13 March. 2004, <http://www.standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf>.

BIOGRAPHY

Ricardo Peculis - Mr Peculis is a senior systems engineer at Computer Sciences Corporation, where he has been involved with major defence projects as a senior engineer and technical manager. The challenge to succeed in software intensive projects motivated him to further investigate the causes that drive undesirable results and ways to improve the effectiveness of such projects. Mr Peculis is currently a PhD student at the Systems Engineering Evaluation Centre of the University of South Australia and his research addresses the implications of socio-organisational design into the complexity of software intensive acquisitions.

Derek Rogers - Dr Derek Rogers is an adjunct senior research fellow at the University of South Australia and Product Development Manager for BAE Systems. He focuses in the areas of systems engineering, wireless telecommunications, innovation and commercialisation. Dr Rogers has published approximately thirty papers and also holds an adjunct role at the University of Adelaide and visiting lecturer role at the University of Canterbury, New Zealand.

Peter Campbell - Professor A. Peter Campbell is currently (2004-7) working under contract with CSIRO Complex Systems Science Initiative to introduce complex system simulation tools for agricultural landscape planning and critical infrastructure analysis. Since May 2004, Professor of Systems Simulation and Modelling with the Systems Engineering and Evaluation Centre (SEEC) at the University of South Australia working on the application of complex adaptive system simulation technology to large scale system integration projects at UniSA and development of the Centre for Defence and Industry Systems Capability(CEDISC). Recent work includes development of agent based simulations to support organizational change within the Australian Defence Organization.