Knowledge Support for Software Projects

Birinder Sandhawalia, National Centre for Project Management, Middlesex University, Trent Park, Bramley Road, London N14 4YZ, UK; E-mail: b.sandhawalia@mdx.ac.uk

Darren Dalcher, National Centre for Project Management, Middlesex University, Trent Park, Bramley Road, London N14 4YZ, UK; E-mail: d.dalcher@mdx.ac.uk

ABSTRACT

The unpredictable nature of software projects and the need for effective communication within project teams requires a framework for social interaction and feedback that results in better decision-making. This paper analyses the creation and capture of knowledge within software development projects and discusses the central role of decision making in the development process. The paper views how the knowledge generated within a software project can be provided greater visibility and communicated effectively, and to achieve this, presents a framework to facilitate social interaction and feedback during the development process.

1. INTRODUCTION

The use of knowledge is expected to result in better decision-making, innovation and competitive advantage within software projects. Software development projects are life-cycle driven and are organised around teams that are assembled specifically for the limited duration of the project. The software development process relies on the knowledge and creativity of individuals and teams, and the formation of these teams requires the involvement and participation of all team members in the development process. There is also an increasing need to involve users early in the software development life-cycle since designing software requires extracting detailed knowledge of the users. Effective communication is the basis for discussion between users and developers during the requirements definition process that is essential to provide an understanding of the software requirements. However, problems of communication occur due to the diversity of professional expertise and organisational roles that confer users' different views and expectations of the system to be developed.

The unpredictable nature of software projects and the need for effective communication within project teams necessitates a framework for social interaction and feedback that results in better decision-making. This paper analyses the creation and capture of knowledge within software development projects. The paper discusses the central role of decision making in the development process and how the effective use of knowledge helps to improve decision-making during the development process. The knowledge created and decisions implemented need to be effectively communicated across the entire process. Social interaction and feedback are key factors that facilitate the effective use of knowledge within software projects. The paper views how the knowledge generated can be provided greater visibility within the projects and communicated effectively, and also presents a framework to facilitate social interaction and feedback during the development process.

2. KNOWLEDGE

Knowledge is the capacity for effective action. Alavi and Leidner (1999) define knowledge as 'a justified personal belief that increases an individual's capacity to take effective action.' While 'personal' implies the contextual nature of knowledge, action requires competencies and know-how, and implies the dynamic nature of knowledge. Knowledge is fluid and formally structured, and it exists within people, processes, structures and routines, (Davenport and Prusak 1998). Polanyi (1967) suggests that knowledge exists as tacit and explicit. Tacit knowledge comprises an individual's mental models, and while it is personal and in the mind of an individual, it is also context specific and difficult to articulate, formalise and verbalise, and is therefore hard to communicate and share. The factors that influence an individual's mental model include the individual's education, expertise, past experiences, perceptions, biases, prejudices and environment. Explicit knowledge can be easily articulated and codified and therefore transmitted and communicated. Polanyi (1967) contends that human beings acquire knowledge by actively creating and organising their own experiences and sums it up by stating that "we can know more than we can tell."

The importance of knowledge is increasing as organisations recognise that they posses knowledge and increasingly learn to view this knowledge as a valuable and strategic asset. Knowledge assets include knowledge which resides within the individuals, systems, processes, documents and structures of the organisation. Davenport and Prusak (1998) recommend that to remain competitive, organisations must efficiently and effectively create, capture, locate and share their organisations knowledge and expertise, and have the ability to bring that knowledge to bear on problems and opportunities.

2.1 Knowledge Management

The American Productivity and Quality Center (1996) defines knowledge management as "a conscious strategy of getting the right knowledge to the right people at the right time and helping people share and put information into action in ways that strive to improve organisational performance." Knowledge management, therefore, requires that it is imperative to identify what knowledge needs to be managed, how, when, where, by whom, and for whom. Consequently, the key elements of KM are collecting and organising the knowledge, making it available through knowledge infrastructure, and then using the knowledge to improve decision making and gain competitive advantage. Alavi and Leidner (1999) refer to knowledge management as a systematic and organisationally specified process for acquiring, organising and communicating both tacit and explicit knowledge of employees so that other employees may make use of it to be more effective and productive in their work and decision-making while improving product and process innovation.

3. THE NEED TO MANAGE SOFTWARE PROJECT KNOWLEDGE

Compared to organisations which are permanent structures and have routines, projects are temporary by nature and their implementation requires creative actions, practitioner's experience, and the ability to apply knowledge to development problems. Projects are designed to achieve specific objectives within a predetermined time frame, budget and resources. Projects involve planning for non-routine tasks to be conducted in several phases, and can be characterised as unique, goal-oriented and complex undertakings steeped in uncertainty, which aim to produce a meaningful product or service in order to satisfy a need, (Dalcher 2003).

Software projects are life cycle driven and follow the sequence of going from concept through definition and development, to testing and delivery. However, unlike other projects, the requirements of software projects are subject to frequent change. As a product, software can be changed, and it is therefore assumed that this change is possible at even the later stages of the development process. Such change and uncertainty make software projects more unpredictable than other projects, and are therefore organised around teams, relying upon the knowledge and creativity of the individuals and the teams. Software projects are typically implemented by teams assembled specifically for the project and often disbanded upon its completion. Requirements evolve and team members often change during the course of projects, while feedback from one phase of the project to another rarely provides team members with an opportunity to learn from their good decisions or mistakes. Team members often come together for the first time at the outset of the project and therefore it is difficult to create the right knowledge culture and locate the knowledge assets. Moreover, project implementation effort is often

focused on immediate deliverables with no emphasis on how the experience and insights gained would help and benefit future projects.

The amount of knowledge required to manage a project depends upon the novelty and uniqueness of the required outcome. Love et al (1999) argue that even though a project is unique, the processes involved in delivering the final outcome are similar in projects and, therefore, most projects do not need to start from scratch as they can utilise existing processes and learn from the experiences acquired from previous projects. Projects are required to be completed within a specific schedule and budget, which makes the reuse and harnessing of knowledge desirable. Without the reuse of existing knowledge or the ability to create new knowledge from existing solutions and experiences, project organisations have to create new solutions to every problem they encounter, potentially leading to delays and inefficiencies. With the reuse of knowledge, projects can be planned more efficiently to be delivered within budget and on time. Koskinen (2004) suggests a metaphor of a project tree to visualise the entire knowledge required by a projects, and states that the types of knowledge that a project may require are tacit, explicit, additive or substitutive. Koskinen (2004) further refers to additive and substitutive knowledge as knowledge that is new to the project and is either invented internally or acquired from external sources. This is similar to Bredillet's (2004) view that project teams need to know what knowledge is available to complete the project based on past experience, and what knowledge needs to be acquired or will emerge as a result of the unique nature of the project tasks, especially within software projects.

The implementation and outcome of projects depends upon a large extent on the knowledge of individuals, their access to local and global knowledge resources, and recognition and integration of existing knowledge. Problem solving within unique project instances generates further knowledge, and the knowledge assets thus created, combined with the experience gained by implementing the project, can benefit subsequent projects. Certain software process improvement approaches, for example the Capability Maturity Model, suggest that the development process be optimised to deliver the most of the software organisation's capability. Such approaches often suggest that knowledge be managed or leveraged, but do not bring it down to an operational level. However, the knowledge requirements make it imperative to identify what knowledge needs to be managed, how, when, where, by whom, and for whom. Consequently, the key requirements for managing knowledge within software projects are collecting and organising the knowledge, making it available through knowledge infrastructure, and then using the knowledge to improve the execution of projects.

The knowledge that is created requires a strategy or model that facilitates the cross leveling of this knowledge across the software development process, and globalises the knowledge created within the software project. Process models for software development depict sequential, incremental, prototyping or evolutionary approaches. Developmental models help simplify and reduce the complexity within software projects by providing a perspective to organise the different stages or phases of the development process. The following section presents and discusses the Dynamic Feedback Model, which underlines the relationships and interactions between the various entities and phases of the software development process.

4. THE DYNAMIC FEEDBACK MODEL

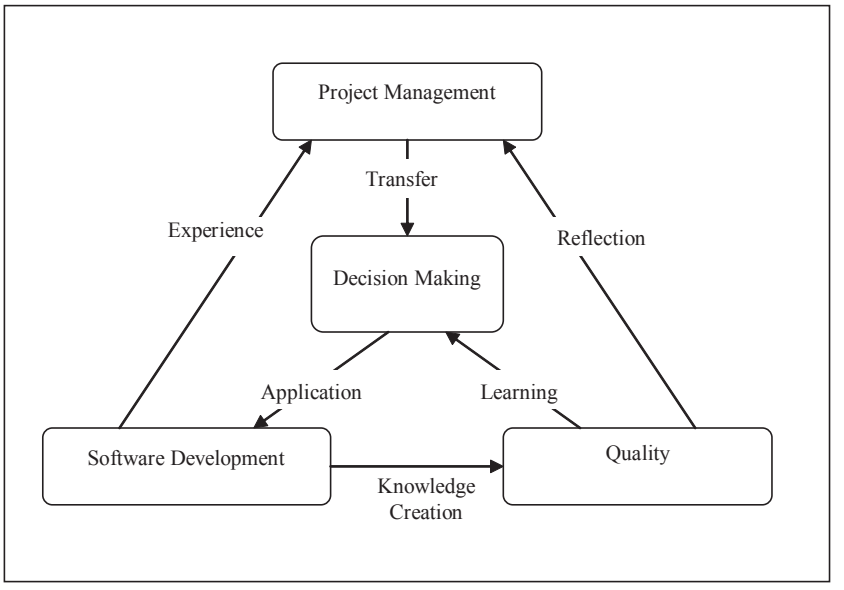
Complex and uncertain software development situations require a model that can account for the knowledge needed to plan and implement decisions within the development process. An example of such a model is the Dynamic Feedback Model (DFM) that underlines the relationships and interactions between the entities by depicting the feedback loops operating between them. The model, as depicted in Figure 1, focuses on four different functional areas that are intertwined throughout software development. The DFM models the relationships in a non-linear fashion amongst the functional areas and allows a continuous view of the development process. The four areas are management, technical, quality and decision-making.

4.1 Functional Areas

The management area involves the planning, control and management of the software development process. It also pertains to the strategy and operation of the project. Key concerns revolve around identifying performance gaps, assessing progress, and allocating resources to accomplish tasks. As technical development and knowledge creation are on going activities, the management area also takes on a continuous view. It does not limit its focus to delivering a product, but to the continuous need for generating and maintaining an on-going flow of knowledge required for continuous development.

The technical area deals with the creation and continuous improvement of the software system. The area recognises the changing needs and perceptions of the development process. The activity in this area includes evolution and maintenance of the software, while also maintaining its functionality and utility. Experimentation, learning and discovery take place as the software goes from inception to evolution. The development and design of the software form the basis for interac-

Figure 1



tion between team members, and the knowledge created through the interaction provides the raw material for decision making within the process.

The quality area is perceived as a dynamic dimension, which continuously responds to perceived mismatches and opportunities reflected in the environment. It is concerned with assuring the quality of the product developed and the process used to develop it. Being an area of assessment, it provides the basis for learning.

The decision-making area lies at the core of the model as software development is described as a decision making process (Dym and Little 2000). This area attempts to balance knowledge, uncertainty and ambiguity with a view to maximise the expected returns on an on-going basis. Knowledge acquired from implementing decisions is used within the process either as background knowledge available to support future decisions, or as a formalised part of an integral body of knowledge which can be used to optimise the decision making process. Decision-making helps manage opportunity and risk and therefore this area can also be considered the risk management area. Risk assessment and planning are key activities within this area, which also ensures the implementation of decisions and the monitoring of their execution on a continuous basis. The knowledge required for the implementation, execution and monitoring of decisions is provided by the interaction and feedback loops of the model.

4.2 Feedback Loops

The DFM is in essence a set of interactions and feedback loops governing and controlling the development of software form a continuous perspective. The decision making perspective of the DFM ensures that rational and reasoned choices are made from the alternatives available during the development process.

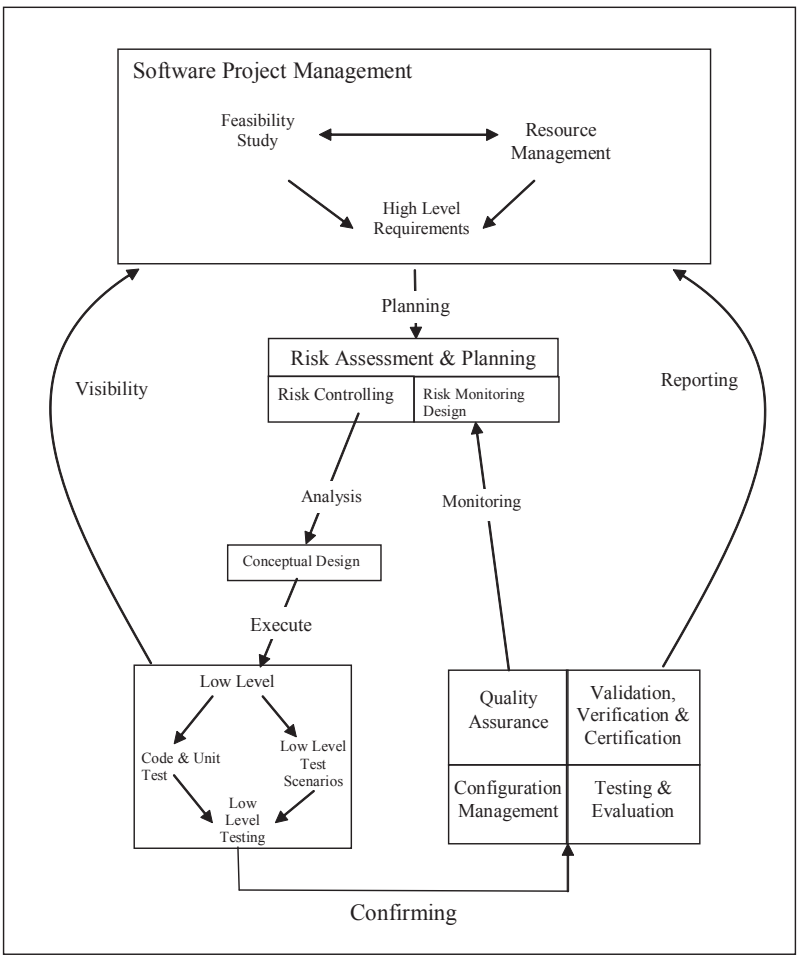
The basic loop in the dynamic system is the knowledge transfer-application-experience loop. This loop helps to plan and control the production, evolution and growth of the software in association with project management and decision making. The loop enables the continuous generation of new information as well as feedback knowledge and experience gained while executing the project. The use of this knowledge is crucial in changing plans to adapt to reality and opportunities, modifying the decisions and re-examining the assumptions. The visibility of this basic feedback loop provides a continuous process to ensure the system remains relevant with regard to its objectives.

The knowledge creation loop links the knowledge created and learning with effective application. The knowledge generated during the design process within the decision-making area is applied to help develop the software. The knowledge created in the technical area helps in quality assurance, while the learning that emerges from identifying and correcting mismatches is fed back to the decision making area for use in subsequent development.

The reflection-transfer loop provides visibility to the project management area regarding the opportunities and mismatches present in the quality area, and also those provided by the implementation and execution of the decisions made.

The above mentioned loops depict relationships between the different functional areas. The DFM can therefore be used as a framework for understanding the dynamic nature of the interactions between entities in software development, and the knowledge that flows between them. The model moves away from linear thinking and offers a continuous perspective for understanding and implementing relationships, and the role these relationships play in software development. The model achieves this through the on-going feedback and interactions of the

Figure 2



loops, which present the framework to provide the knowledge flow required for software projects. The following section examines the feedback and interactions between the different phases of software development projects.

4.3 The DFM Process

The phases of the software development process can broadly be categorised as problem definition, requirements analysis, design, implementation and maintenance. The DFM views knowledge as a key asset in the development of software and focuses on its feedback within the functional areas of development. In doing so, the DFM encourages thinking about software development in terms of the different phases and their interactions. The feedbacks within the functional areas of the DFM are depicted in Figure 2, and the use of knowledge for decision making within the various activities of software projects.

The project management area facilitates project planning and execution, and is also where the user requirements are elicited and the problem defined. Planning involves resource management where the skills and competencies required to execute the project are identified and teams are formed. Proper requirements analysis and specification are critical for the success of the project, as most defects found during testing originate in requirements. In order to understand the customer requirements, the developers require insight into the domain of the business system and the technical concepts of the system to be developed. Knowledge is created while understanding the requirements by the interaction of the different team members, and also between the users and the developers. This knowledge provides the perspective for decisions made to implement the project. The project management area is where discussion takes place between the users and developers as software development requires that users are involved in the development of the software. A clear understanding is needed between the users and developers to build the software, and this understanding is established through dialogue and communication. The formalisation of such an understanding usually results in the form of proposals and contracts. The feasibility of the project and the cost involved in executing the project are the basis for the proposals and contracts. The project management area addresses the need to assess the feasibility of the project and its cost analysis.

Based upon the decisions made and the outcome of planning within the project management area, an analysis of the impact the project will have on the business and technical environment is made along with the possible risks involved in implementing the project. The analysis views the goals, scope and functionality of the system being developed and how they fit or respond to the existing processes with which they are required to interact. Risk assessment and planning are conducted and feature the two traditional components of risk identification and prioritisation. Identification tries to envision all situations that might have a negative impact on the project, while prioritisation involves analysing the possible effects and consequences of the risk in case it actually occurs. The project also requires crucial decisions to be made in the design stage. High level design is the phase of the life cycle that provides a logical view of the development of the user requirements. Design involves a high level of abstraction of the solution, through which requirements are translated into a 'blueprint' for constructing the software, and provides the architecture of the application and its database design. Decision making at this stage of the process helps transform the requirements into a set of software functions and a physical database structure. Scenarios are developed to test the acceptability of the design with relation to the requirements.

The technical activities of design, code generation and testing are performed in the technical area. The area includes the detailed design phase where the high level design is converted into modules and programs. A unit test plan is created for the conditions for which each program is to be tested. The required programs are coded or translated into the programming language, and the programs are tested using the unit test plans. The technical area ensures that the integration plan is implemented according to the environments identified for integration. The area also ensures the maintenance, functionality and utility of the software apart from its creation and evolution. The decisions made in this area relate to the technical activities and provide confirmation of the design and suitability of the requirements. The decisions made are verified during system testing within the quality assurance area.

Pressman (1997) states that quality assurance consists of the auditing and reporting functions of management, and that its goal is to provide management with the data necessary to be informed and assured about product quality. The quality assurance area involves system testing which validates that the software developed meets the requirement specification. This phase identifies the defects that are exposed by testing the entire system. A series of tests are performed together, each with a different purpose, to verify that the system has been properly integrated and performs its functionality and satisfies the requirements. The quality assurance area thus provides verification of the decisions made and tasks performed in the technical area while confirming the decisions made during the design phase, and validating the requirements.

The different phases of the process are validated and given visibility by the feedback loops. Controlling the execution of decisions generates knowledge (Dalcher 2003a). The feedback loops globalise this knowledge within the process and ensure that knowledge is available for decision making. The decisions made in the decision making area during design and risk analysis receive confirmation during technical development and quality assurance. Technical development provides the project management area visibility of the software being developed to meet the requirements. Quality assurance further reports and validates to project management the decisions made during design and technical development. The project management area is able to assess the feedback and incorporate it in planning to help address some of the change and uncertainty inherent within the software development process.

5. CONCLUSIONS

Software projects require knowledge to implement projects effectively. Software projects are organised around teams and rely upon the knowledge, creativity and competence of the individual team members. Effective knowledge management helps provide timely and required knowledge to team members, which results in better productivity and quality of the software processes and product. The DFM adopts a long-term perspective of software development that enables it to address the issues of uncertainty and ambiguity, and therefore benefit from the decisions made and knowledge created during the development process. The long-term perspective also enables the DFM to look beyond a single project and use the knowledge generated towards improvements in future software projects. The DFM is receptive to changes in the environment and tackles them by feeding acquired knowledge back into the decision making process. As software development becomes more integrated in management practices the importance of continuous learning, knowledge, and skill acquisition as underpinned by the DFM will remain central to improved control, visibility and management. The availability of a long-term view justifies the adoption of multiple perspectives, the reuse of knowledge and the utilisation of a decision making perspective, which underpin feedback and improvement.

The DFM provides a framework that facilitates social interaction and feedback, which further enhance the use of knowledge within the software development process. The feedback loops help facilitate the flow of knowledge created and insights gained within the processes and developmental activities of the functional areas. The continuous view of software development provided by the DFM enables the knowledge, both tacit and explicit, to be globalised through-out the software project organisation. In the domain of software development, the DFM provides software project organisations with an approach that focuses on the non-technical aspects, and the knowledge required to support the developmental effort. The DFM helps identify how and where knowledge is created, shared, transferred, applied and assimilated within the software project organisation. In doing so, the DFM provides a framework and culture that views knowledge as a valuable resource, and supports the effective implementation of software projects. Future work includes validating knowledge support provided by the DFM for software development projects.

6. REFERENCES

[1] Alavi M and Leidner DE (1999) 'Knowledge Management Systems: Issues, Challenges and Benefits,' Communications of the Association for Information Systems, Vol 1, Article 7.

[2] American Productivity and Quality Center (APQC) (1996), Knowledge Management: Consortium Benchmarking Study Final Report; Available from http://www.store.apqc.org/reports/Summary/know-mng.pdf

[3] Bredillet CN (2004) 'Projects are Producing the Knowledge which are Producing the Projects……' Proceedings of IPMA, Budapest.

[4] Dalcher D (2003) Computing Project Management, Middlesex University Press, London.

[5] Dalcher D (2003a) 'Software Development for Dynamic Systems.' Developments in Metainformatics. LNCS, Springer Verlag, Peter J Nurnberg (ed): pp 58-75.
[6] Davenport TH and Prusak L (1998) 'Working Knowledge,' Harvard Business School Press, Boston.
[7] Dym C L and Little P (2000) 'Engineering Design: A Project Based Introduction.' John Wiley, New York.
[8] Koskinen KU (2004) 'Knowledge Management to Improve Project Communication and Implementation,' Project Management Journal, Vol 35, No 1, pp 13-19.
[9] Love, P.E.D, Smith, J and Li, H, (1999) 'The Propagation of Rework Benchmark Metrics for Construction,' International Journal of Quality and Reliability Management, Vol 16, No 7, pp 638-658.
[10] Polanyi M (1967) 'The Tacit Dimension,' Routledge and Keon Paul, London.
[11] Pressman RS (1997) 'Software Engineering – A Practitioner's Approach,' The McGraw-Hill Companies, Inc.