KNOWLEDGE SHARING IN TRADITIONAL AND AGILE SOFTWARE PROCESSES

Broderick Crawford
Pontificia Universidad Católica de Valparaíso, Universidad Técnica Federico Santa María, Chile

Claudio León de la Barra, José Miguel Rubio León
Pontificia Universidad Católica de Valparaíso, Universidad de las Américas - Laureate International Universities, Chile

Keywords: Knowledge Sharing, Agile Development, Software Development Techniques, Knowledge Management.

Abstract: The software development community has a wide spectrum of methodologies to implement a software project. In one side of the spectrum we have the more Traditional deterministic software development derived from Tayloristic management practices, and in the other side, are the Agile software development approaches. The Agile processes are people oriented rather than process oriented, unlike the Traditional processes they are adaptive and not predictive. Software development is a knowledge intensive activity and the Knowledge Creation and Sharing are crucial parts of the software development processes. This paper presents a comparison between Knowledge Sharing approaches of Agile and Tayloristic software development teams.

1 INTRODUCTION

In Software Engineering a vision of the methodologies that improve productivity and quality of its software products is absolutely necessary. Furthermore, Software Engineering is a knowledge intensive process that includes some aspects of Knowledge Management (KM) in all phases: eliciting requirements, design, construction, testing, implementation, maintenance, and project management. No worker of a development project possess all the knowledge required for fulfilling all activities. This underlies the need for knowledge sharing support to share domain expertise between the customer and the development team (Chau et al., 2003). The traditional approaches often referred to as plan-driven, task-based or Tayloristic, like the waterfall model and its variances, facilitate knowledge sharing primarily through documentation. They also promote usage of role based teams and detailed plans of the entire software development life-cycle. It shifts the focus from individuals and their creative abilities to the processes themselves. In contrary, agile methods emphasise and value individuals and its interactions over processes. Tayloristic methods heavily and rigorously use documentation for capturing knowledge gained in the activities of a software project life-cycle (Chau and Maurer, 2004). In contrast, agile methods suggest that most of the written documentation can be replaced by enhanced informal communications among team members and among the team and the customers with a stronger emphasis on tacit knowledge rather than explicit knowledge (Beck et al., 2001). We believe that in software development projects, a better understanding of knowledge sharing and transfer, from a Knowledge Management perspective, offers important insights about the use of Software Engineering methodologies.

2 KNOWLEDGE MANAGEMENT IN SOFTWARE ENGINEERING

The main argument to Knowledge Management in software engineering is that it is a human and knowledge intensive activity. Software development is a process where every person involved has to make a large number of decisions and individual knowledge has to be shared and leveraged at a project level and organization level, and this is exactly what KM proposes. People in such groups must collaborate, communicate and coordinate their work, which makes knowledge management a necessity. In software de-

Crawford B., León de la Barra C. and Miguel Rubio León J. (2008).
KNOWLEDGE SHARING IN TRADITIONAL AND AGILE SOFTWARE PROCESSES.
In *Proceedings of the Third International Conference on Software and Data Technologies - PL/DPS/KE*, pages 376-379
DOI: 10.5220/0001892203760379

velopment one can identify two types of knowledge: Knowledge embedded in the products or artifacts, since they are the result of highly creative activities and Meta-knowledge, that is knowledge about the products and processes. Some of the sources of knowledge (artifacts, objects, components, patterns and templates) are stored in electronic form. However, the majority of knowledge is tacit, residing in the brains of the employees. A way to address this problem can be to develop a knowledge sharing culture, as well as technology support for knowledge management. In (Rus and Lindvall, 2002) there are several reasons to believe that knowledge management for software engineering would be easier to implement than in other organizations.

3 TWO APPROACHES TO KM: PRODUCT AND PROCESS

Knowledge Management has been the subject of much discussion over the past decade and different KM life-cycles and strategies have been proposed. One of the most widely accepted approaches to classifying knowledge from a KM perspective is the *Knowledge Matrix* of Nonaka and Takeuchi (Nonaka and Takeuchi, 1995). This matrix classifies knowledge as either explicit or tacit, and either individual or collective. Nonaka and Takeuchi also proposes corresponding knowledge processes that transform knowledge from one form to another: *socialisation* (from tacit to tacit, whereby an individual acquires tacit knowledge directly from others through shared experience, observation, imitation and so on); *externalisation* (from tacit to explicit, through articulation of tacit knowledge into explicit concepts); *combination* (from explicit to explicit, through a systematisation of concepts drawing on different bodies of explicit knowledge); and *internalisation* (from explicit to tacit, through a process of learning by doing and through a verbalisation and documentation of experiences). Traditional methods of software development use a great amount of documentation for capturing knowledge gained in the activities of a project life-cycle. In contrast, the agile methods suggest that most of the written documentation can be replaced by enhanced informal communications among team members and customers with a stronger emphasis on tacit knowledge rather than explicit knowledge. In the KM market a similar situation exists and two approaches to KM have been mainly employed; we will refer to them as the *Product* and the *Process* approaches. These approaches adopt different perspectives in relation to documentation and interactions among the stakeholders (Mentzas, 2000).

3.1 Knowledge as a Product

The product approach implies that knowledge can be located and manipulated as an independent object. Proponents of this approach claim that it is possible to capture, distribute, measure and manage knowledge. This approach mainly focuses on products and artefacts containing and representing knowledge.

3.2 Knowledge as a Process

The process approach puts emphasis on ways to promote, motivate, encourage, nurture or guide the process of learning and abolishes the idea of trying to capture and distribute knowledge. This view mainly understands KM as a social communication process, which can be improved by collaboration and cooperation support tools. In this approach, knowledge is closely tied to the person who developed it and is shared mainly through person-to-person contacts. This approach has also been referred to as the *Collaboration* or *Personalisation* approach. Choosing one approach or other will be in relation to the characteristics of the organization, the project and the people involved in each case (Apostolou and Mentzas, 2003).

4 AGILE METHODS

A new group of software development methodologies has appeared over the last few years. For a while these were known as lightweight methodologies, but now the accepted term is Agile methodologies (Fowler, 2001). There exist many variations, but all of them share the common principles and core values specified in the Agile Manifesto (Chau and Maurer, 2004). Through its work they have come to value individuals and interactions over processes and tools. Working software over comprehensive documentation. Customer collaboration over contract negotiation. Responding to change over following a plan.

These new methods attempt a useful compromise between no process and too much process, providing just enough process to gain a reasonable payoff. The result of all of this is that agile methods have some significant differences with the former engineering methods (Fowler, 2001): Agile methods are adaptive rather than predictive and people oriented rather than process oriented. The different software development approaches implicitly consider a simple linear life cycle model in which the algorithm undergoes a design/tuning phase and is then employed in production. Beside being the simplest life cycle model, the linear model is also the building block composing more complex models: Iterative, Hybrid (such as the Spiral) and Agile. Spiral, in this context, can be described as a sequence of interleaved design/tuning phases and production phases: At the end of each production phase, sufficient information is gathered which can be employed in the following design/tuning phase for improving the algorithm. Agile methods attempt to minimize risk by developing software in short iterations, each iteration is like a miniature software project of its own, and includes all of the tasks necessary to release the mini-increment of new functionality: planning, requirements analysis, design, coding, testing, and documentation. That is, some agile teams use the waterfall model on a small scale, repeating the entire waterfall cycle in every iteration.

Knowledge Management implementations in Software Engineering can extract knowledge from its sources of knowledge: documentation, artifacts, objects, components, patterns, templates and code repositories, exploiting this knowledge in future software developments. But, software reuse is not a technology problem, nor is it a management problem. Reuse is fundamentally a Knowledge Management problem. In (Highsmith, 2008) Jim Highsmith explains how over the last ten or so years, by packaging objects into components and components into templates, we have made the problem bigger, not smaller. Objects, patterns, templates, and components are packaged (explicit) knowledge: *the larger the package, the greater the encapsulated knowledge. The greater the encapsulated knowledge, the harder it is to transfer*. Additionally, the essence of problem solving, innovation, creativity, intuitive design, good analysis, and effective project management involves more tacit knowledge, the harder it is to transfer. By putting tacit knowledge in a principal role and cultivating tacit knowledge environments, KM can play an important role in application development, and particularly in reuse. A second aspect of the explicit knowledge problem, observed by Highsmith, is the fallacy that documentation (explicit knowledge) equals understanding. When, in order to successfully reuse a component, we seek understanding in the documentation, the larger and more complex the component, the harder it is to gain the required understanding from documentation alone. Understanding, in this context at least, is a combination of documentation and conversation about the component and the context in which that component operates. No writer of documentation can anticipate all the questions a component user may have.

5 KNOWLEDGE SHARING IN AGILE AND TAYLORISTIC METHODS

About knowledge sharing in plan-driven and agile development approaches the main different strategies are in the following dimensions (Chau et al., 2003):

5.1 Eliciting Requirements

Common to all software development processes is the need to capture and share knowledge about the requirements and design of the product, the development process, the business domain and the project status. In Tayloristic development approaches this knowledge is externalised in documents and artifacts to ensure all possible requirements, design, development and management issues are addressed and captured. One advantage to this emphasis on knowledge externalisation is that it reduces the probability of knowledge loss as a result of knowledge holders leaving the organisation. Agile methods advocate lean and mean documentation. Compared to Tayloristic methods, there is significantly less documentation in agile methods. As less effort is needed to maintain fewer documents, this improves the probability that the documents can be kept up to date. To compensate for the reduction in documentation and other explicit knowledge, agile methods strongly encourage direct and frequent communication and collaboration.

5.2 Training

With regards to disseminating process and technical knowledge from experienced team members to novices in the team, Tayloristic and agile methods use different training mechanisms as well. While it is not stated, formal training sessions are commonly used in Tayloristic organizations to achieve the above objective. Agile methods, on the other hand, recommend informal practices, for example, pair programming and pair rotation in case of eXtreme Programming (XP).

5.3 Trust

As software development is a very social process, it is important to develop organisational and individual trust in the team and also among the team and the customer. Trusting other people facilitates reusability and leads to more efficient knowledge generation and knowledge sharing. Through collective code ownership, stand-up meetings, on site customer, and in case of XP, pair programming, agile methods promote and encourage mutual trust, respect and care among developers themselves and to the customer as well. The key of knowledge sharing here are the interactions among members of the teams which happen voluntarily, and not by an order from the headquarters (Cockburn and Highsmith, 2001).

5.4 Team Work

In Tayloristic teams different roles are grouped together as a number of role-based teams each of which contains members who share the same role. In contrast, agile teams use cross functional teams. Such a team draws together individuals performing all defined roles. In knowledge intensive software development that demands information flow from different functional sub-teams, role based teams tend to lead to islands of knowledge and difficulties in its sharing among all the teams emerge. Learning, or the internalisation of explicit knowledge, is a social process. One does not learn alone but learns mainly through tacit knowledge gained from interactions with others. Furthermore, tacit knowledge is often difficult to be externalised into a document or repository. A repository by itself does not support communication or collaboration among people either. Due to the high complexity of the software process in general, it is hard to create and even more difficult to effectively maintain the experience repository (Rus and Lindvall, 2002).

6 CONCLUSIONS

In Software Engineering many development approaches work repeating the basic linear model in every iteration, then in a lot of cases an iterative development approach is used to provide rapid feedback and continuous learning in the development team. To facilitate learning among developers Agile methods use daily or weekly stand up meetings, pair programming and collective ownership. Agile methods emphasis on people, communities of practice, communication, and collaboration in facilitating the practice of sharing tacit knowledge at a team level. They also foster a team culture of knowledge sharing, mutual trust and care. Agile development is not defined by a small set of practices and techniques. Agile development defines a strategic capability, a capability to create and respond to change, a capability to balance flexibility and structure, a capability to draw creativity and innovation out of a development team, and a capability to lead organizations through turbulence and uncertainty. They rough out blueprints (models), but they concentrate on creating working software. They focus on individuals and their skills and on the intense interaction of development team members among themselves and with customers and management. Since software development is a knowledge intensive activity, we think that a better understanding from a Knowledge Management perspective offers important insights about Reusability and Software Engineering.

REFERENCES

Apostolou, D. and Mentzas, G. (2003). Experiences from knowledge management implementations in companies of the software sector. *Business Process Management Journal*, 9(3).

Beck, K., Beedle, M., Bennekum, A. V., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). Manifesto for agile software development. Available at http://agilemanifesto.org.

Chau, T. and Maurer, F. (2004). Knowledge sharing in agile software teams. In Lenski, W., editor, *Logic versus Approximation: Essays Dedicated to Michael M. Richter on the Occasion of his 65th Birthday*, volume 3075 of *Lecture Notes in Artificial Intelligence*, pages 173–183. Springer.

Chau, T., Maurer, F., and Melnik, G. (2003). Knowledge sharing: Agile methods vs tayloristic methods. In *Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE*, pages 302–307, Los Alamitos, CA, USA. IEEE Computer Society.

Cockburn, A. and Highsmith, J. (2001). Agile software development: The people factor. *IEEE Computer*, 34(11):131–133.

Fowler, M. (2001). The new methodology. Available at http://www.martinfowler.com/articles/newMethodology.html.

Highsmith, J. (2008). Reuse as a knowledge management problem. Available at http://www.informit.com/articles/article.aspx?p=31478.

Mentzas, G. (2000). The two faces of knowledge management. *International Consultant's Guide*, pages 10–11. Available at http//imu.iccs.ntua.gr/Papers/O37-icg.pdf.

Nonaka, I. and Takeuchi, H. (1995). *The Knowledge Creating Company*. Oxford University Press.

Rus, I. and Lindvall, M. (2002). Knowledge management in software engineering. *IEEE Software*, 19(3):26–38. Available at http://fc-md.umd.edu/mikli/RusLindvallKMSE.pdf.