Knowledge Management and Creativity Practices in Software Engineering

Broderick Crawford[1], Claudio León de la Barra[1], Ricardo Soto[1], Sanjay Misra[2] and Eric Monfroy[3]
[1]*Pontificia Universidad Católica de Valparaíso, PUCV, Valparaíso, Chile*
[2]*Federal University of Technology, Minna, Nigeria*
[3]*LINA, Université de Nantes, Nantes, France*

Keywords: Knowledge Management, Creativity, Software Engineering, Agile Methodologies.

Abstract: An increasing number of organizations are trending to teams for innovation and creativity. In software engineering it is the same, in the last years the traditional perspective on software development is changing and agile methods have received considerable attention. Among other attributes, the agilists claim that fostering knowledge sharing and creativity is one of the keys to response to common problems and challenges of software development today. The development of new software products requires the generation of novel and useful ideas. Here, we fixed some concepts from knowledge management and creativity in relation with new software engineering trends.

1 INTRODUCTION

Software engineering is a knowledge intensive discipline where its activities require the use and sharing of knowledge between the stakeholders. Then a better transfer and application of the knowledge aim to foster the software processes, whether these are done using traditional or agile approaches. In software organizations the knowledge held by the employees is the main asset and software development projects depends mostly on team performance: ”Software is developed for people and by people” (John et al., 2005). But surprisingly, most of software engineering research is technical and deemphasizes the human and social aspects. By other hand, the traditional development process of new products that is a fundamental part in the marketing, has been recently criticized by Kotler and Trías de Bes (Kotler and Trías de Bes, 2004). They point out that fundamental creative aspects are not considered at all and as a consequence this development is not useful, viable or innovative. In this context, it is interesting to consider the new proposals of agile methodologies for software development in order to analyse and evaluate them at the light of the existing creative expositions, mainly considering the teamwork practices. The agile principles and values have emphasized the importance of collaboration and interaction in the software development and, by other hand, creative work commonly involves collaboration in some form and it can be understood as an interaction between an individual and a sociocultural context. In relation with the joint work between users and software developers, there are very interesting cases in healthcare enterprises and medicine. The relationship between agile approaches and the health sector has a notable case with the work of Jeff Sutherland, the inventor of the popular Scrum Agile Development Process (Sutherland, 2001). Scrum, the most notorious competitor of eXtreme Programming XP (Beck, 2000), has attained worldwide fame for its ability to increase the productivity of software teams by several magnitudes through empowering individuals, fostering a team-oriented environment, and focusing on project transparency and results. There are recent studies reporting efforts to improve agile process (Crawford and Leon de la Barra, 2008). Agile software development addresses software process improvement within teams. The work in (Ringstad et al., 2011)(Moe et al., 2010) argue for the use of diagnosis and action planning to improve teamwork in agile software development. The action planning focused on improving shared leadership, team orientation and learning. The improvement project provided most new insight for the mature team. We believe that the innovation and development of new products is an interdisciplinary issue (Takeuchi and Nonaka, 1986), we are interested in the study of human and social factors to foster

Crawford B., León de la Barra C., Soto R., Misra S. and Monfroy E..
Knowledge Management and Creativity Practices in Software Engineering.
DOI: 10.5220/0004141002770280
In *Proceedings of the International Conference on Knowledge Management and Information Sharing* (KMIS-2012), pages 277-280
ISBN: 978-989-8565-31-0

software quality (Sanz and Misra, 2011)(Mishra and Misra, 2010).

2 CREATIVITY IN SOFTWARE DEVELOPMENT

Software engineering is a knowledge intensive process that includes some aspects of Knowledge Management (KM) and Creativity in all phases: eliciting requirements, design, construction, testing, implementation, maintenance, and project management (John et al., 2005). No worker of a development project has all the knowledge required to fulfill all activities. This underlies the need for communication, collaboration and 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 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 internally and between the team and the customers with a stronger emphasis on tacit knowledge rather than explicit knowledge (Beck, 2000).

Since human creativity is thought as the source to solve complex problem or create innovative products, one possibility to improve the software development process is to design a process which can stimulate the creativity of the developers. There are few studies reported on the importance of creativity in software development. In management and business, researchers have done much work about creativity and obtained evidence that the employees who had appropriate creativity characteristics, worked on complex, challenging jobs, and were supervised in a supportive, noncontrolling fashion, produced more creative work. Then, according to the previous ideas the use of creativity in software development is undeniable, but requirements engineering is not recognized as a creative process in all the cases (Maiden et al., 2004). In a few publications the importance of creativity has been investigated in all the phases of software development process (Gu and Tong, 2004)(Glass, 1995)(Crawford and Leon de la Barra, 2007)(Leon de la Barra and Crawford, 2007) and mostly focused in the requirements engineering (Robertson, 2005)(Mich et al., 2005). Nevertheless, the use of techniques to foster creativity in requirements engineering is still shortly investigated. It is not surprising that the role of communication and interaction is central in many of the creativity techniques. The most popular creativity technique used for requirements identification is the classical brainstorming and more recently, role-playing-based scenarios, storyboard-illustrated scenarios, simulating and visualizing have been applied as an attempt to bring more creativity to requirements elicitation. These techniques try to address the problem of identifying the viewpoints of all the stakeholders (Mich et al., 2005).

However, in requirements engineering the answers do not arrive by themselves, it is necessary to ask, observe, discover, and increasingly create requirements. If the goal is to build competitive and imaginative products, we must make creativity part of the requirements process. Indeed, the importance of creative thinking is expected to increase over the next decade (Maiden and Gizikis, 2001). In (Robertson, 2005)(Robertson, 2002) very interesting open questions are proposed: Is inventing part of the requirements activity? It is if we want to advance. So who does the inventing? We cannot rely on the customer to know what to invent. The designer sees his task as deriving the optimal solution to the stated requirements. We cannot rely on programmers because they are far away from the work of client to understand what needs to be invented. Requirements analysts are ideally placed to innovate. They understand the business problem, have updated knowledge of the technology, will be blamed if the new product does not please the customer, and know if inventions are appropriate to the work being studied. In short, requirements analysts are the people whose skills and position allows, indeed encourages, creativity. In (Boden, 2004) the author, a leading authority on cognitive creativity, identifies basic types of creative processes: exploratory creativity explores a possible solution space and discovers new ideas, combinatorial creativity combines two or more ideas that already exist to create new ideas, and trans-formational creativity changes the solution space to make impossible things possible. Then, most requirements engineering activities are exploratory, acquiring and discovering requirements and knowledge about the problem domain. Requirements engineering practitioners have explicitly focused on combinatorial and transformational creativity.

3 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 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 development 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, templates and containers) 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. About knowledge sharing in plan-driven and agile development approaches the main different strategies are in the following dimensions (Chau and Maurer, 2004):

Eliciting Requirements and Documentation. 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.

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.

Trust and Freedom. As software development is a very social process, it is important to develop organisational and individual trust in the teams and also between the teams 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).

Team Work and Roles. 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).

4 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. An important finding is the need to not focus exclusively on explicit knowledge but also on tacit knowledge. 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. By other side, creativity and innovation are essential skills in almost any teamwork. Having a team that can solve problems quickly and effectively with a little creative thinking is beneficial to everyone. The performance of a team depends not only on the competence of the team itself in doing its work, but also on the organizational context. The performance that characterizes the team through certain advisable practices, from the perspective of creativity, constitutes the necessary basic conditions, although nonsufficient, in order to favor the group creative performance.

REFERENCES

Beck, K. (2000). *Extreme programming explained: embrace change*. Addison-Wesley Longman Publishing Co., USA.

Boden, M. (2004). *The Creative Mind: Myths and Mechanisms*. Routledge, USA.

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 versus tayloristic methods. *Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE*, pages 302–307.

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

Crawford, B. and Leon de la Barra, C. (2007). Enhancing creativity in agile software teams. *Lecture Notes in Computer Science*, 4536:161–162.

Crawford, B. and Leon de la Barra, C. (2008). Integrating creativity into extreme programming process. In Cordeiro, J. and Filipe, J., editors, *ICEIS (3-1)*, pages 216–219.

Glass, R. (1995). *Software creativity*. Prentice-Hall, USA.

Gu, M. and Tong, X. (2004). Towards hypotheses on creativity in software development. *PROFES*, 3009:47–61.

John, M., Maurer, F., and Tessem, B. (2005). Human and social factors of software engineering: workshop summary. *ACM SIGSOFT Softw. Eng., Notes*, 30:1–6.

Kotler, P. and Trías de Bes, F. (2004). *Marketing Lateral*. Editorial Pearson/Prentice Hall, Spain.

Leon de la Barra, C. and Crawford, B. (2007). Fostering creativity thinking in agile software development. *Lecture Notes in Computer Science*, 4799:415–426.

Maiden, N. and Gizikis, A. (2001). Where do requirements come from? *IEEE Software*, 18:10–12.

Maiden, N., Gizikis, A., and Robertson, S. (2004). Provoking creativity: Imagine what your requirements could be like. *IEEE Software*, 21:68–75.

Mich, L., Anesi, C., and Berry, D. (2005). Applying a pragmatics-based creativity-fostering technique to requirements elicitation. *Requir. Eng.*, 10:262–275.

Mishra, A. and Misra, S. (2010). People management in software industry: the key to success. *ACM SIGSOFT Software Engineering Notes*, 35(6):1–4.

Moe, N., Dingsoyr, T., and Dyba, T. (2010). A teamwork model for understanding an agile team: A case study of a scrum project. *Information and Software Technology*, 52:480–491.

Ringstad, M., Dingsoyr, T., and Moe, N. (2011). Agile process improvement: Diagnosis and planning to improve teamwork. *Communications in Computer and Information Science*, 172:167–178.

Robertson, J. (2002). Eureka! why analysts should invent requirements. *IEEE Software*, 19:20–22.

Robertson, J. (2005). Requirements analysts must also be inventors. *IEEE Software*, 22:48–50.

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.

Sanz, L. F. and Misra, S. (2011). Influence of human factors in software quality and productivity. In Murgante, B., Gervasi, O., Iglesias, A., Taniar, D., and Apduhan, B. O., editors, *ICCSA (5)*, volume 6786 of *Lecture Notes in Computer Science*, pages 257–269. Springer.

Sutherland, J. (2001). Agile can scale: Inventing and reinventing scrum in five companies. *Cutter IT Journal*, 14:5–11.

Takeuchi, H. and Nonaka, I. (1986). The new new product development game. *Harvard Business Review*.