AGILE PRACTICES: A COGNITIVE LEARNING PERSPECTIVE

Mazni Omar[1] and Sharifah Lailee Syed Abdullah[2]

[1]*Universiti Utara Malaysia, Malaysia, mazni@uum.edu.my*
[2]*Universiti Teknologi MARA, shlailee@perlis.uitm.edu.my*

ABSTRACT. This paper highlights the theoretical aspect of agile practices from the cognitive learning perspective. Three cognitive strategies – elaboration, organization, and problem solving – underpin key strategies in agile practices to promote better understanding in learning software development activities. Agile practices such as planning games, pair programming, refactoring, coding standard and simple design, acts as a positive inducer to human brain for software developers to accept and develop software easily. By understanding theoretical aspects hinders in agile practices, educators are able to determine alternative approach to suit into internal potentials among students, and thus be able to develop high quality software.

Keywords: software engineering (SE), agile practices, extreme programming, cognitive learning, quality software

INTRODUCTION

Due to rapid demand of technological changes, agile methodologies have emerged to alleviate the uncertainty of business requirements. The need to deliver quality software in a timely manner and at economical cost is the main issue in software industry. Therefore, the Agile Alliance (2001) has expressed the values in Agile Manifesto as:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The major agile methodologies currently in use consist of Scrum, Dynamic Systems Development Method (DSDM), Crystal Methods, Feature Driven Development (FDD), Lean Development, and Extreme Programming (XP). According to the agile manifesto, people and communication are the key ingredients required for producing quality software. Agile employs a lightweight process, whereby communication plays an important role and takes precedence over comprehensive documentation. This method focuses more on people-oriented approach, which relies on tacit or interpersonal knowledge whilst developing software. The creators of Agile Alliance agreed that detailed project strategies should be continuously innovated in order to discover a larger set of agile software practices (Cockburn, 2007). Therefore, it is not surprising to find different set of practices that are similar and complementary to each other.

Beck (2000) introduced XP as a solution to the problems encountered when the formal methods are adopted in software development projects. In particular, the XP methodology was created to address requirement changes and project risk. XP is governed by four values—

Communication, Simplicity, Feedback, and Courage—that lead to definition of practices that XP projects should follow. XP has gained a great attention and is currently the most widely used methodology in SE industry. It was first developed and implemented in 1996, by Kent Beck at Chrysle Corporation. There were originally twelve major practices in XP; the planning game, pair programming, refactoring, simple design, continuous integration, test-first programming, collective ownership, coding standards, frequent releases, metaphor, sustainable pace and an on-site client (Beck, 2000). These practices were based on four XP values, which are communication, simplicity, feedback, and courage.

Integrating agile practices in SE education and application this methodology can stimulate human minds, thus helping the students in acquiring necessary knowledge and skills while attending SE courses. Furthermore, as agile methodology originated from industry, including this method into the academic curriculum would decrease the gap between educational and industrial requirements, thus preparing the students for the workplace by arming them with the knowledge of the latest developments in software engineering process environment. However, despite its benefits, the use of agile methodology in software engineering education is a relatively new phenomenon (Mahnic, 2012; Melnik & Maurer, 2003; Paasivaara et al., 2013; Rico & Sayani, 2009; Sharifah-Lailee, Mazni, Mohd Nasir, Che Latifah, & Kamaruzaman, 2009).

Although there is consensus that integrating agile methodology in education is beneficial and relevant, educators need to ensure that their students are fully trained in its application in order to maximize benefits of using agile methodology in learning process. Therefore, educators need to understand underlying theories that hinders benefits of agile practices.

COGNITIVE LEARNING IN AGILE PRACTICES

Agile practices underline the need for understanding and applying the principles of cognitive learning theory in order for software developers to apply the practices effectively. By gaining insight into the theoretical aspects of agile practices, developers can appreciate adhering to the practices and thus become self-affirming. Theory allows developers to manipulate and improve the practices in a reflective process. Therefore, when problems arise, understanding these theoretical strategies offers developers a platform for recognizing, analysing, and dealing with issues in an effective manner. Moreover, cognitive theory enables practitioners to share knowledge and best practices can be demonstrated. Lastly, given the widely accepted view that knowledge is power, understanding the theory underpinning each of these practices will empower practitioners to explain, justify, and promote the agile practices (Carlile & Jordan, 2005; Dingsøyr, Nerur, Balijepally, & Moe, 2012; Wang, Ali, Ramos, & Vidgen, 2013).

In general, the term ‘cognitive’ refers to development of intellectual abilities and skills (Borich, 2004). This includes the mechanism by which mental or brain processes information is received, stored, and retrieved. Similarly, cognitive learning is defined as the acquisition of knowledge and skill by mental or cognitive processes and the procedures for manipulating the information. Key theorists in cognitivism are Jean Piaget and Lev Vygotsky. The cognitivism of Piaget (Piaget, Brown, & Thampy, 1985) suggests that learners’ cognitive development is continuous process, based on the key concept of assimilation and accommodation. Assimilation is a process of responding to the environment in accordance to existing learners’ knowledge, whereas accommodation implies modifying the learners’ knowledge based on new information. Therefore, knowledge occurs when there is the interaction between the individual and the environment. Learning in agile practices demands developers to actively engage in software development activities. During these activities, developers require acquiring new knowledge and incorporated with developers’ current knowledge to develop

software that meet clients' requirements. Therefore, the process of knowledge is assimilated and accommodated through applying agile practices.

Vygotsky (1978) introduced the concept of the Zone of Proximal Development (ZPD), that is, the gap between what learners could accomplish alone and what could be achieved in cooperation with those more skilled or experienced. Thus, Vygotsky stressed the importance of collaborative learning. In agile practices, collaborative learning is incorporated through the need for the client and team members to actively participate in software development activities. Frequent feedback from the client as well as development team members in pair programming and continuous integration activities allow developers to learn, interact, and share knowledge to enhance learning process.

Learning requires humans to use their brains to think and solve problems. However, human brains have a limited memory capacity. Therefore, to help improve cognitive learning as a mental strategy, researchers, such as Arends (2008) and Borich (2004), proposed several cognitive learning strategies. The cognitive learning strategies that underpin the agile practices are:

Elaboration Strategy

This strategy allows developers to learn how to build internal connections between new information and existing knowledge by emphasizing systematic organization of knowledge. Elaboration is a process of adding details, so that new requirements from clients become more meaningful, and thus easier to understand and memorize. This, in turn, facilitates development of high quality software. There are several techniques that are based on elaboration strategy, which include note taking and analogies technique.

Note taking is similar to writing user stories during a planning game practice. During the planning game, developers write user stories (see Figure 1) that define client requirements. This strategy enhances learning by compactly accumulating the stories for later review and discussion between members and client. In addition, developers also incorporate coding standard practice with note taking technique by adding comments to their code. By applying this coding standard, the code written by different team members is easier to understand and helps software reuse in the future projects.

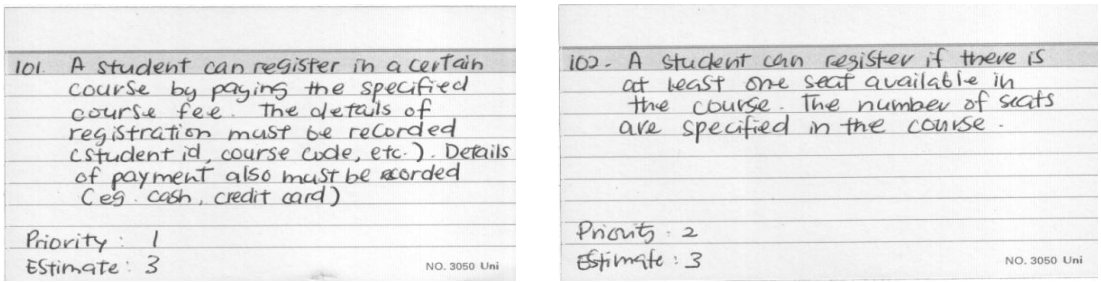

Figure 1. Example of Story Cards

The metaphor in agile practices uses analogies to allow comparison of features. During development activity, developers use metaphors, such as a comparison of registering for a course with add and drop features. The metaphor practice will increase understanding between developers by linking the ideas of a system with common or familiar features. The concept of metaphor also allows developers to understand details of the system architecture.

Organization Strategy

This strategy assists learning through reorganizing new knowledge by creating patterns and links in a large array of information, which will ensure that developers can remember and use the new knowledge efficiently (Arends, 2008; Borich, 2004). There are several techniques based on organization strategy, which include outlining, chunking, mnemonic, and mapping.

Outlining technique is implemented in agile practices through planning game practice. When defining stories in a planning game, developers outline ideas according to functions that represent client requirements. Therefore, the act of outlining user stories assists developers in relating a variety of functions according to client needs, helping them understand the requirements.

Chunking is a second technique in which the information is divided into smaller and meaningful collections of knowledge, which is also incorporated into the planning game. Through the use of story cards, it is easier for developers to group different stories according to specific functions. Chunking assists developers because humans have limited memory capacity, making it difficult to memorize a large amount of information. The process of chunking is also facilitated by identifying smaller functions to be developed and by continuously integrating, testing, and reviewing the completed work into a cohesive unit. These activities can help developers to incrementally develop quality software.

The third technique that helps organizing information is mnemonics, which consists of refactoring. Mnemonics is a technique that assists memory by helping developers to organize information according to a familiar pattern so that it can easily fit the schema pattern in the long term memory (Arends, 2008). Refactoring works in the same way as the mnemonic, i.e. through the reuse of the same program codes, albeit in a different functions. Therefore, complex codes can be understood amongst developers.

The fourth organization strategy is mapping technique as it allows complex client requirements to be transformed into simple design practice (see Figure 2).

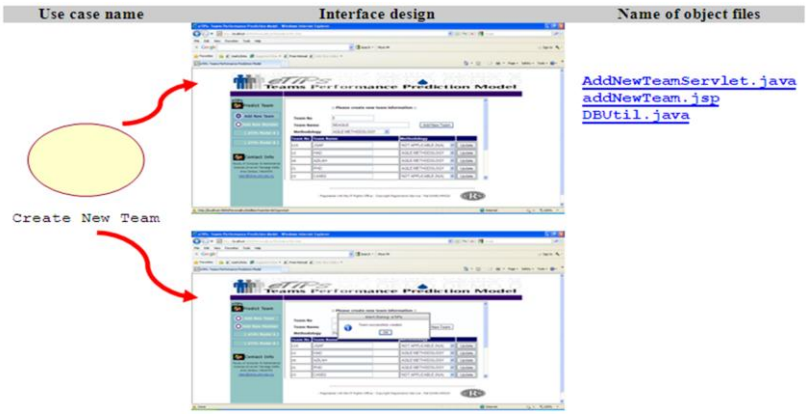

Figure 2. Simple Design Practice

In simple design, developers map system main functions using Unified Modelling Language (UML) use case diagram, which is subsequently associated to its interface design. Next, the interfaces are linked to object source codes. These activities facilitate developers in understanding the flow of system, making it easier to update for future reference.

Problem Solving Strategy

Problem solving strategy is one of the cognitive strategies that assist learners in solving ill-structured problems by relying on their knowledge sourced from several disciplines (Borich, 2004). Through pair programming activities (see Figure 3), agile practices enhance developers' ability to solve programming problems. In pair programming, developers are actively involved in discussing, listening, and observing. This practice is designed so that two developers are required to work side by side, which facilitates communication, exchanging, and sharing ideas, leading to faster code production. When two developers interact, rigorous discussion occurs, and thus learning process is enhanced and intrinsic motivation is sustained throughout the pair programming process.

Figure 3. Pair Programming Practice

In addition, agile practices emphasize the importance of frequent feedback from clients and participation of all developer in this interaction. Social nature of feedback encourages learning process by active interaction amongst developers and clients, governed by the need to understand the system. Thus, this process aids the developers in using their past experiences and integrated it with their new knowledge, based on information obtained from clients to solve programming problems. Therefore, this learning process develops problem-solving skills that are essential for producing quality software.

CONCLUSION AND RECOMMENDATIONS

The theoretical aspects hinder in agile practices specifically on extreme programming (XP) demonstrate its ability to stimulate human minds to accept and develop software easily. With this in mind, software developers are able to develop high quality software regardless the complexity of the software project. The cognitive learning strategies encapsulated in agile practices are the necessity for fostering dynamic smart software community that provides opportunities to enhance their skills and capability.

Educators need to take challenge to bring out human potentials among students by training and educating them to apply agile practices in their software projects. This alternative approach enables students to improve their learning process toward achieving high quality software. This, in turn, allows software industry to have world class ICT professionals.

ACKNOWLEDGMENTS

The authors wish to thank the Ministry of Education Malaysia for funding this study under Fundamental Research Grant Scheme (FRGS), S/O project: "12818" and Dana Kecemerlangan UiTM, code project: "600-RMI/ST/DANA 5/3/Dst (102/2009)".

REFERENCES

Alliance, A. (2001). Manifesto for agile software development. Retrieved 31 March, 2009, from http://www.agilemanifesto.org

Arends, R. I. (2008). *Learning to teach (8th ed.)*. Singapore: McGraw-Hill.

Beck, K. (2000). *Extreme programming explained: Embrace change*. USA: Addison-Wesley.

Borich, G. D. (2004). *Effective teaching methods (5th ed.)*. Upper Saddle River, New Jersey: Prentice Hall.

Carlile, O., & Jordan, A. (2005). It works in practice but will it work in theory? The theoritical underpinnings of pedagogy In G. O'Neill, S. Moore & B. McMullin (Eds.), *Emerging issues in the practice of university learning and teaching* Dublin: All Ireland Society for Higher Education (AISHE).

Cockburn, A. (2007). *Agile software development: The cooperative game (2nd ed.).* . USA: Addison Wesley.

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software, 85*(6), 1213-1221.

Mahnic, V. (2012). A capstone course on agile software development using scrum. *Education, IEEE Transactions on, 55*(1), 99-106.

Melnik, G., & Maurer, F. (2003). Introducing agile methods in learning environments: Lessons learned. In F. Maurer & D. Wells (Eds.), *Extreme programming and agile methods - XP/Agile Universe 2003* (Vol. 2743, pp. 172-184). New Orleans: Springer-Verlag.

Paasivaara, M., Lassenius, C., Damian, D., R, P., Tyet al. (2013). *Teaching students global software engineering skills using distributed Scrum*. Paper presented at the Proceedings of the 2013 International Conference on Software Engineering.

Piaget, J., Brown, T., & Thampy, K. J. (1985). *The equilibration of cognitive structures: The central problem of intellectual development.* Chicago: University of Chicago Press

Rico, D. F., & Sayani, H. H. (2009). *Use of agile methods in software engineering education.* Paper presented at the Agile Conference, 2009. AGILE '09.

Sharifah-Lailee, S.-A., Mazni, O., Mohd Nasir, A. H., Che Latifah, I., & Kamaruzaman, J. (2009). Positive affects inducer on software quality. *Computer and Information Science, 2*(3), 64-70.

Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes* MA: Harvard University Press.

Wang, X., Ali, N., Ramos, I., & Vidgen, R. (2013). *Agile and lean service-oriented development: Foundations, theory, and practice* (Vol. Hershey, PA, USA): IGI Global.