A Cognitive Perspective on Software Development Methods: The Case of Extreme Programming

Orit Hazzan
Department of Education in Technology and Science
Technion – Israel Institute of Technology
Haifa, Israel
(9724) 829 3107
oritha@tx.technion.ac.il

Yael Dubinsky
Department of Computer Science
Technion – Israel Institute of Technology
Haifa, Israel
yael@cs.technion.ac.il

ABSTRACT
One of the main dilemmas with which software development teams face is how to choose a software development method that suits the team as well as the organization. This short paper suggests an approach that may help in this process. Specifically, Extreme Programming (XP) is analyzed from a cognitive perspective. We suggest that such an analysis can support software development processes.

Categories and Subject Descriptors
D.2.9 [Management]: Life cycle, Programming teams, Software process models.

General Terms
Management, Measurement, Human Factors.

Keywords
agile software development methods, extreme programming, cognitive theories, constructivism.

1. INTRODUCTION
Software development is a complicated task that is based strongly on the people that carry it out. When deciding on a software development method that suits the organization and the team, suitability of the method to the people involved should be addressed as well (Dines, 2003).

Methods are usually selected according to organization and team traditions and according to practical-professional-technical considerations, such as programming languages and tools. This situation motivated us to examine the analysis of software development methods from other perspectives, such as social and cognitive ones. Specifically, this short paper illustrates how theories taken from different fields can be used in the analysis of software development methods. Focus is placed on a cognitive theory.

Specifically, in this paper we use the constructivism theory (presented in Section 2), which is based on Piaget's work (Piaget, 1977) and addresses learning processes, for the analysis of software development methods. In the analysis that follows (Section 3), we use Extreme Programming (Beck, 2000, 2005) to illustrate our arguments. In a similar manner, in a previous work (Hazzan and Dubinsky, 2005) we analyzed software development methods from a social perspective and illustrated the analysis of software development methods within a game theory framework – the Prisoner's Dilemma – which is usually used for the analysis of cooperation and competition.

We view the perspectives presented in this short paper as part of our research on the human aspects of software engineering in general, and on cognitive and organizational aspects of agile software development, both in industry and in academia, in particular (see for example, Hazzan and Dubinsky, 2003A, 2003B; Tomayko and Hazzan, 2004; Dubinsky and Hazzan, 2004).

2. A COGNITIVE THEORY: CONSTRUCTIVISM
Constructivism is a cognitive theory that examines the nature of learning processes. A central tenet of the constructivist approach is that learners construct new knowledge by rearranging and refining their existing knowledge (cf. Davis, Maher and Nodding, 1990; Smith, diSessa and Roschelle, 1993). More specifically, the theory suggests that new knowledge is constructed gradually, based on the learner's existing mental structures. Mental structures are developed in steps, each elaborating on preceding ones, although there may of course be regressions and blind alleys.
Leron and Hazzan (1997) refer to this process as "learning by successive refinement". The term successive refinement is borrowed from the domain of Computer Science, where it refers to a methodology that guides a gradual elaboration of complex programs (Dijkstra, 1972). This use of successive refinement is based on the assumption that successive refinement is an especially effective way for the human mind, with its particular strengths and limitations, to deal with complexity.

It is well known that, from the cognitive perspective, software development is a complex process (Cf. Hamlet and Maybee, 2001). Therefore, according to the constructivism theory, we suggest analyzing software development methods according to their ability to support a gradual process of knowledge construction related to software development processes. In what follows we demonstrate the use of this criterion by analyzing Extreme Programming (XP) which is the most prevalent agile software development method. Specifically, we explain how the XP values of feedback and communication, as well as several of the XP practices, support gradual construction of knowledge related to the development of software systems. This illustration reflects how these values and practices make the development process clearer for software developers and customers, and, consequently, the task of software development might be able to be carried out more confidently.

3. CONSTRUCTIVIST-BASED ANALYSIS OF EXTREME PROGRAMMING

In this Section we analyze XP based on the constructivism theory, focusing on two values and three practices of the method.

The value of feedback: The importance of feedback in learning processes is explained by Piagetian theory as follows: Children come to know their environment by acting on it and by noting the feedback that comes from the (physical or computational) constraints that are built into the environment. Based on these experiences, they construct theories, mental models and explanations that help them make sense of the environment and predict its behavior in relation to their future actions on it. In cases where the behavior of the environment is observably different from their predictions, children are moved to adjust their theories and explanations to account for the new experience. This acting-theorizing-predicting-testing-refining cycle, which may repeat itself many times, is the basic mechanism for gradual learning by successive refinements.

The analogy to the case of XP-guided software development process seems to be immediate. XP practices, which are derived from the value of feedback, provide developers with various kinds of feedback received from the different participants in the development environment (the customer, the teammates, the code) as well as with a technical feedback received from the development environment (accepted, for example, from unit testing and refactoring activities). Through these feedbacks, developers improve and refine their understanding of what is being developed.

The value of communication: The second XP value that supports gradual construction of knowledge is communication. By encouraging and eliciting communication between the different parties participating in the development environment, this value provides feedback, which, as described previously, guides and directs gradual construction of knowledge.

We now illustrate how three of the XP practices – small releases, refactoring and metaphor – support gradual construction of knowledge related to software development processes.

Small Releases: The analysis presented in this section suggests that developing the software in short releases guides the customer, as well as the developers, in a gradual process of knowledge construction with respect to the developed software. It is a known fact that customers find it difficult to determine all of the required features of the software in advance. Still, some software development processes require customers to define software requirements in detail at the beginning of the development process. In the case of XP, iteration and release planning games are conducted frequently in accordance with the practice of short releases. These planning games provide customers with the opportunity to rethink, refine and improve their understanding of the software they require. Consequently, customers are able to define and communicate their software requirements to the software developers in a more precise and clear manner.

Refactoring: This practice encourages programmers to keep improving code structure and readability through refactoring activities that do not add functionality to the code. As can be observed, the essence of refactoring is a gradual process of code improvement. More specifically, for cases in which the final "correct" structure of the code (and design) cannot be predicted in advance, refactoring serves as a tool that leads developers in a gradual learning process through which the code (and design) are improved. From the constructivist perspective, it is suggested that this process of code improvement is based on, and conducted in parallel to, a mental process in which developers improve their understanding of the structure of the developed software. We suggest that the inclusion and legitimization of refactoring as part of the software development method is based on the acknowledgment that non-trivial knowledge, related to the structure and shape of the software, can be constructed only by means of a gradual process during the actual process of software development.

Metaphor: The common use of metaphors is to bridge between a known domain and an unknown domain. Accordingly, metaphors are used to understand and experience one specific concept using the terms of another concept (Lakoff and Johnson, 1980). Within the XP framework, metaphors are used to improve communications between the different parties participating in the process of software development (customer, development, business). The rationale for using this practice can be explained from a constructivist perspective, as follows: New knowledge is constructed based upon existing knowledge. If an attempt is made to understand a new topic related to the developed software, but the basis on which the new knowledge can be constructed is lacking, then a metaphor can serve as the basis on which the new knowledge can be built, and thus the metaphor supports the gradual construction of the said knowledge.

We believe that the readership can apply a similar analysis with respect to other XP practices, such as test-driven development.
4. CONCLUSION

This short paper examines the application of cognitive perspectives for the analysis of software development methods, by illustrating the analysis of Extreme Programming (XP) from a constructivist perspective. In general, this short paper suggests that a software development method can be analyzed, not only by referring to its technical benefits but, also, by viewing it from a cognitive perspective.

The scope of this paper is limited to the examination of a software development method in general, and of XP in particular, using a single cognitive framework. It is suggested that similar examinations in at least two directions can be carried out. First, other software development methods can be analyzed using this framework; second, the analysis of software development methods can be examined based on additional theories borrowed from other disciplines.

ACKNOWLEDGMENTS

We would like to thank The Samuel Neaman Institute for Advanced Studies in Science and Technology and the Technion MANLAM Fund for their generous support of this research.

REFERENCES