Chapter 2.21
Constructivist Learning During Software Development

Václav Rajlich
Wayne State University, USA

Shaochun Xu
Laurentian University, Canada

ABSTRACT

This article explores the non-monotonic nature of the programmer learning that takes place during incremental program development. It uses a constructivist learning model that consists of four fundamental cognitive activities: absorption that adds new facts to the knowledge, denial that rejects facts that do not fit in, reorganization that reorganizes the knowledge, and expulsion that rejects obsolete knowledge. A case study of an incremental program development illustrates the application of the model and demonstrates that it can explain the learning process with episodes of both increase and decrease in the knowledge. Implications for the documentation systems are discussed in the conclusions.

INTRODUCTION

One of the puzzling issues of software engineering is the nature of the knowledge that is needed in order to develop and evolve a program. The program itself is a repository of knowledge about the program domain and may contain knowledge that is not available elsewhere, as documented by Kozaczynski and Wilde (1992). It also contains knowledge of all design decisions that were made during the program development and consequent program evolution (Rugaber, Ornburn, & LeBlanc, 1990). When evolving or maintaining the program, it is necessary to recover this knowledge; otherwise, maintenance or evolution will be impossible. It is also necessary to communicate this knowledge to all new programmers who are joining an existing software project. The loss of the programming knowledge can be a serious

Copyright © 2009, IGI Global, distributing in print or electronic forms without written permission of IGI Global is prohibited.

problem and was identified as a leading cause of the code decay (Rajlich & Bennett, 2000).

Although the knowledge is embedded in the program, it cannot be easily recovered since it is encoded in programming structures and delocalized into different components of the program. Moreover, the consequences of the decisions, rather than the decisions themselves, appear in the code. In many ways, the recovery of knowledge from the code is similar to solving a puzzle and is laborious and error prone.

One of the most basic questions that concerns the nature of the programmer knowledge is the issue of its monotonicity. According to a naïve view, the knowledge steadily increases, as the new facts emerge and are absorbed by the programming team; many current documentation systems are geared towards that (Ye, 2006). However, in this article we show that there are also episodes of the knowledge retraction, and the documentation systems should provide an adequate support for that also.

Our approach in the article is based on cognitive informatics (CI). CI is a multidisciplinary study of cognition and information sciences, which investigates human information processing mechanisms and processes and their applications in computing (Wang & Kinsner, 2006); studying the knowledge and cognitive process involved in software development is one of the goals of cognitive informatics.

In order to understand the nature of programming knowledge and its acquisition, we adopted and further developed a constructivist model of programmer learning that is based on four basic cognitive activities: absorption, denial, reorganization, and expulsion of the knowledge. We validated this model in a case study of the pair programming that is a part of eXtreme Programming (Martin, 2002). In pair programming, two programmers work side-by-side at one machine as they collaborate in program design, implementation, and testing. The programming pair has to communicate and share the knowledge, and this gives an opportunity to analyze unobtrusively their dialog for the indications of the programmer knowledge and learning.

The first section of this article describes our theory of constructivist learning. The second section describes the case study. The third section contains the discussion of the results of the case study and the fourth section has an overview of the related literature. The fifth section contains general conclusions and future work.

THEORY OF CONSTRUCTIVIST LEARNING

The constructivist learning model is based on the work of Piaget (Piaget, 1954). The original aim of Piaget was to explain learning in children, but the constructivist theory extends to adult learning and to epistemology (von Glasersfeld, 1995). The theory assumes that the learners actively and incrementally construct their knowledge. They start from some preliminary knowledge, and they extend it by adding new facts to it; they may go through stages in which they may accept ideas that they will later discard as wrong. The two main activities are assimilation and accommodation, where assimilation describes how learners deal with new knowledge, and accommodation describes how learners reorganize their existing knowledge.

We modified this theory by dividing assimilation into two separate activities. Absorption means that the learners add new facts to their knowledge. However, if the new facts do not fit in, the learners may reject them; we call this second activity a denial. We also divided accommodation into two separate activities. Reorganization means that the learners reorganize their knowledge to aid future absorption of new facts. Expulsion is the process where part of the knowledge becomes obsolete or provably incorrect and the learners reject it. Of course, there are also mixed activities: learners may absorb a modified fact, rather than make an

11 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage: www.igi-global.com/chapter/constructivist-learning-during-software-development/29427

Related Content

A Survey on Using Nature Inspired Computing for Fatal Disease Diagnosis
Prableen Kaurand Manik Sharma (2017). *International Journal of Information System Modeling and Design (pp. 70-91).*
www.irma-international.org/article/a-survey-on-using-nature-inspired-computing-for-fatal-disease-diagnosis/199004

Analyzing Growth Trends of Reusable Software Components
Kuljit Kaur (2013). *Designing, Engineering, and Analyzing Reliable and Efficient Software (pp. 40-54).*
www.irma-international.org/chapter/analyzing-growth-trends-reusable-software/74873

Mitigating Type Confusion on Java Card
Jean Dubreuil, Guillaume Bouffard, Bhagyalekshmy N. Thampiand Jean-Louis Lanet (2013). *International Journal of Secure Software Engineering (pp. 19-39).*
www.irma-international.org/article/mitigating-type-confusion-java-card/77915

Code Clone Detection and Analysis in Open Source Applications
Al-Fahim Mubarak-Ali, Shahida Sulaiman, Sharifah Mashita Syed-Mohamadand Zhenchang Xing (2014). *Handbook of Research on Emerging Advancements and Technologies in Software Engineering (pp. 494-509).*
www.irma-international.org/chapter/code-clone-detection-and-analysis-in-open-source-applications/108633

Using a Systems Thinking Perspective to Construct and Apply an Evaluation Approach of Technology-Based Information Systems
Hajer Kefi (2010). *Emerging Systems Approaches in Information Technologies: Concepts, Theories, and Applications (pp. 294-309).*
www.irma-international.org/chapter/using-systems-thinking-perspective-construct/38186