Chapter 5
Theory Driven Modeling as the Core of Software Development

Janis Osis
Institute of Applied Computer Systems, Riga Technical University, Riga, Latvia

Erika Nazaruka (Asnina)
Institute of Applied Computer Systems, Riga Technical University, Riga, Latvia

ABSTRACT

Some experts opine that software is built in a primitive way. The role of modeling as a treatment for the weakness of software engineering became more important when the principles of Model Driven Architecture (MDA) appeared. Its main advantage is architectural separation of concerns. It showed the necessity of modeling and opened the way for software development to become an engineering discipline. However, this principle does not demonstrate its whole potential power in practice because of lack of mathematical accuracy in the very initial steps of software development. The sufficiency of modeling in software development is still disputable. The authors believe that software development in general (and modeling in particular) based on mathematical formalism in all of its stages and together with the implemented principle of architectural separation of concerns can become an important part of software engineering in its real sense. They propose the formalism by topological modeling of system functioning as the first step towards engineering.

1. INTRODUCTION

The software developer community understands and forcedly accepts that software development in its current state is rather an art than an engineering process. This means that quality software is a piece-work or a craftwork. Such an item usually is expensive and cannot be stock-produced. However, in the modern world software users want to see and to use a good quality and relatively cheap product. This means that software development must become software engineering. The word “engineering” implies an approach that is theory-approved, completely realized, reused many times in practice, and gives a qualitative and relatively inexpensive end product in accurately predictable timeframes.

DOI: 10.4018/978-1-7998-3016-0.ch005

The software development's way to software engineering is quite long. There are a lot of different factors that make this way long. From one perspective, software development lacks commonly accepted theoretical foundations. From another perspective, software developers do not want to use "hard" theory (especially mathematical), because winning the market requires providing operating software as fast as possible and even faster, but the lack of theory just delays getting an operating product. From the third perspective, clients do not want to pay a lot of money for a product that, first, exists only as a text document, second, includes "intellectual" work that is hard to measure and evaluate, and third, usually is not the same what clients wanted. Clients cannot check the work progress since they cannot see the product at whole before integration of its parts and cannot evaluate (or even understand) the size of made efforts.

The content of this article is an updated version of our vision (Osis & Asnina, 2011b) of how to shorten this long way. First, we discuss effectiveness and quality of software engineering, then – differences between traditional engineering disciplines and software engineering. Next, we consider a modeling process and discuss its benefits as well as development issues that can and cannot be solved only by modeling. At the end, we discuss our vision on what must be done to achieve a revolutionary improvement of software development including the security aspect.

2. EFFECTIVENESS AND QUALITY OF SOFTWARE ENGINEERING IS LOW

To improve the understanding of the motivation of this discussion, let us look at the effectiveness and quality of software engineering. Our discussion is grounded on the very important results of the research performed by Capers Jones (2009). Jones and his colleagues from SPR have collected historical data (between 1977 and 2007) from hundreds of corporations and more than 30 government organizations. This historical data is a key source for judging the effectiveness of software process improvement methods. This data is also widely cited in software litigation in cases where the proceedings concern quality, productivity, and schedules.

The main result obtained during the analysis of this historical data can be summarized in one sentence: "The way software is built remains surprisingly primitive" (Jones, 2009, p. 1). This statement is true also nowadays and is based on the following data:

- Budget and schedule overruns. Even in 2008 majority of software applications are cancelled, overrun their budgets and schedules, and often have hazardously bad quality levels when released. As time passes, the global percentage of programmers performing maintenance on aging software has steadily risen, until it has become the dominant activity of the software world.
- Product and process innovations. External product innovations (new or improved products) and internal process innovations (new or improved methods for reducing development resources) are at differing levels of sophistication. Even in 2008 very sophisticated and complex pieces of software are still constructed by manual methods with extraordinary labor content (jobs from the United States to India, China, etc.) and very distressing quality levels. Yet software quality and productivity levels in 2007 are hardly different from 1977.
- Positive and Negative Innovations. Capers Jones and his colleagues have introduced two interesting terms, namely, positive innovations and negative innovations (Jones, 2009). Their meaning is explained on the example of agile techniques. The Agile approaches and eXtreme Programming (XP) were developed to speed up the development of small projects, where small teams working