Chapter 1
Is Modeling a Treatment for the Weakness of Software Engineering?

Janis Osis
Riga Technical University, Latvia

Erika Asnina
Riga Technical University, Latvia

ABSTRACT

The authors share with some other experts the opinion that the way software is built is primitive. Therefore, this chapter discusses a role of modeling as a treatment for software engineering. The role of modeling became more important after appearance of principles proposed by Model Driven Architecture (MDA). The main advantage of MDA is architectural separation of concerns that showed necessity of modeling and opened the way to software development to become engineering. However, the weakness is that this principle does not demonstrate its whole potential power in practice, because of a lack of mathematical formalism (or accuracy) in the very initial steps of software development. Therefore, the question about the sufficiency of modeling in software development is still open. The authors believe that software development in general, and modeling in particular, based on mathematical formalism in all its stages together with the implemented principle of architectural separation of concerns can become Software Engineering in its real sense. The authors introduce such mathematical formalism by means of topological modeling of system functioning.

INTRODUCTION

Software developers' community understands and forcedly accepts that software development in its current state is rather art than an engineering process. This means that qualitative 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 qualitative and relatively cheap product. This means that software *development* must become software engineering. The word "engineering" intends a theory approved, completely realized and reused many times in practice that gives a qualitative and relatively inexpensive end product in accurately predictable timeframes.

DOI: 10.4018/978-1-61692-874-2.ch001

Software development's way to software engineering is quite long. Things that make this way long are very different. From one viewpoint, software development lacks commonly accepted theoretical foundations. From another viewpoint, software developers do not want to use "hard" theory (especially mathematical) because in order to win on the market they must provide operating software as fast as possible and even faster, but a lack of theory just slower getting an operating product. From the third viewpoint, clients do not want to pay a powerful lot of money for a product that, first, exists only as a textual document, second, includes "intellectual" work that is hard to measure and to evaluate, and third, usually it is not the same as clients wanted. Clients cannot check how work proceeds, since they cannot see the product at whole before integration of its parts and cannot evaluate (or even understand) the size of introduced efforts.

The content of this chapter is our vision of how to shorten this long way. First, we discuss effectiveness and quality of software engineering, and then differences between traditional engineering disciplines and software engineering. Next, we consider a modeling process and discuss benefits and issues, which could and could not be solved by modeling. At the end, we discuss our vision on what must be done in order to get really revolutionary improvement of software development.

BACKGROUND

Effectiveness and Quality of Software Engineering

First, let us discuss effectiveness and quality of software engineering. Our discussion is grounded on the very important results of the research performed by Capers Jones and presented in (Jones, 2009). Currently, Capers Jones is a president of Capers Jones & Associates LLC. He is also a founder and a former chairman of Software Productivity Research LLC (SPR). Jones and his colleagues from SPR have collected historical data (since 1977 till 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 quality, productivity, and schedules are parts of the proceedings. Jones also frequently works as an expert witness in software litigation. In brief, Capers Jones is an authority in software engineering.

The main result obtained during analysis of this historical data can be expressed in one sentence - *"The way software is built remains surprisingly primitive"* (Jones, 2009, p. 1). This statement 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 ap-