Chapter 53
Software Development Crisis: Human-Related Factors' Influence on Enterprise Agility

Sergey Zykov
National Research University Higher School of Economics, Russia

ABSTRACT

Software development is critically dependent on a number of factors. These factors include techno-logical and anthropic-oriented ones. Software production is a multiple party process; it includes customer and developer parties. Due to different expectations and goals of each side, the human factors become mission-critical. Misconceptions in the expectations of each side may lead to misbalanced production; the product that the developers produce may significantly differ from what the customers expect. This misbalanced vision of the software product may result in a software de-livery crisis. To manage this crisis, the authors recommend using software engineering methods. Software engineering is a discipline which emerged from the so-called "software crisis" in the 1960s: it combines technical and anthropic-oriented "soft" skills. To conquer the crisis, this chapter discusses general architecture patterns for software and hardware systems; it provides instances of particular industries, such as oil and gas and nuclear power production.

INTRODUCTION

This chapter focuses on human factor-related project lifecycle estimation and optimization, which are based on software engineering methods and tools.

Enterprise systems are usually large-scale and complex; they combine hardware and software. Managing development of large and complex software systems is a key problem in software engineering discipline. In the 1960s, this discipline emerged as a result of the so-called "software crisis". This term originated from the critical development complexity, which happened due to the rapid growth of computational power. The challenge was so dramatic that the NATO had to arrange an invitation-only conference, which brought together leading researchers and practitioners from the US and Europe to

DOI: 10.4018/978-1-7998-3016-0.ch053

search for a remedy. The conference was held in 1967 in Germany; its key participants were such famous computer science professors and Turing Award winners as Alan Perlis from the USA, Edsger Dijkstra from Holland, Friedrich Bauer from Germany, and Peter Naur from Denmark. Many of these researchers were also the NATO Science Committee representatives of their countries. At that time, the computing power of the machines (including the IBM B-5000) became so overwhelming that a number of software development projects were late, over budget or totally unsuccessful. Irrespective of human efforts, the complexity of the hardware and software systems was hard to cope with by means of the old methods and techniques. At the same time, the term "crisis" was coined by F.Bauer; later on E.Dijkstra also used it in his Turing Award lecture.

However, not only did the participants recognize the crisis state of software production management, but they also announced a remedy. This was software engineering. The term was suggested by the same F. Bauer, and the idea was that software developers could apply the engineering methods used in material production to the emerging domain of large-scale software systems in order to make the software projects more measurable, predictable and less uncertain. It appeared that this software engineering approach was feasible, though the methods and practices used had to differ substantially from those used in large-scale and complex material production. The fundamental difference between large-scale software and material production was the distribution of time and cost by the development lifecycle phases. In the case of software, maintenance was the most time and cost consuming activity; it often exceeded 60% of the expenses for the entire project (Schach, 2011). This is why the new software engineering discipline was in need of new methodologies, techniques and tools.

Another distinct feature of software product development was that it involved a number of parties with clearly different goals and expectations. These were end users or customers, developers and their management. Due to multiple sides' participation in the development of the software products, these sides usually lacked common understanding of the resulting product: the customers often used business terms, while the developers preferred technological jargon.

Currently, this same lack of common vision complicates the development processes; it is a possible source of a local production crisis in terms of a certain software project. To deal with this kind of vision incompatibility crisis, software engineers should add to their purely technical abilities a very special kind of skillset also known as "soft" skills. These are teamwork, communications, negotiations, and basics of risk management, to name a few. Thus, in order to address the crisis, which has a human factor-related root cause, a high quality software engineer should possess a carefully selected blend of technical and managerial skills.

BACKGROUND

Issues related to software and hardware system development, often referred to as systems of systems, tend to become even more essential and critical in the enterprise context. One positive solution is creating uniform architectural patterns for such complex systems. To verify the patterns, specific instances are required for particular industries, such as oil and gas and nuclear power production.

The general principles of describing the architecture, i.e. key components and relationships of the software system (for instance, in terms of components and connectors), commonly used in software engineering (Lattanze, 2008), are generally applicable for enterprise system engineering. Therewith, top level architectural design is critical; it determines the key concepts and interfaces for software and