Systems and Software Engineering in IT System Development

S

Marcel Jacques Simonette
Universidade de São Paulo, Brazil

Edison Spina
Universidade de São Paulo, Brazil

INTRODUCTION

In any development process of an Information Technology (IT) system, the team responsible for system implementation needs to acquire the necessary knowledge about the problem to be solved. However, this approach does not guarantee that the implemented system will yield the value expected by stakeholders. The development process in an IT system has a complexity that is process inherent, and System Engineering can provide a response that considers this complexity, and supports Software Engineering along the IT system life cycle.

Every development process used by Software Engineering to create the software of an IT system – be it Agile, Prototyping, Unified Process, or any other process or method – has activities that are always present, such as teamwork management, the management of the knowledge that is acquired through the interaction among the different team members, and between these IT system developers and stakeholders.

The presence of people throughout the IT system life cycle is a factor that brings complexity to system development, due to the different interests, desires, values and emotions of human beings. Another issue that brings complexity to system development is that there are several companies that do not have IT system development as its core business, although they have some kind of Information Technology (IT) department, which is built to be efficient in the development process.

This article is about System Engineering during IT software system life cycle, supporting Software Engineering activities, with focus on a socio-technical approach that promotes interactions among IT system development team and stakeholders, to create value, differentiation and contribute to organizational efficiency and performance.

DOI: 10.4018/978-1-4666-5888-2.ch726

BACKGROUND

Traditional engineering disciplines (e.g. electrical, mechanical, civil, chemical) are concerned with transforming physical entities from one form into another; they follow physical laws or properties and relations. System Engineering and Software Engineering are engineering disciplines that do not have a core theory, physical law or properties; these engineering disciplines deal with the work activities to develop a system or software.

System Engineering

In the United States, during the Second World War, the postwar period and the beginning of the Cold War, there was a need to develop techniques based on General Systems Theory to create great defense systems. Engineers, scientists and managers developed techniques that have become known by (1) Systems Engineering for the design and development of systems, (2) Operational Research to analyze the Armament Systems and (3) Systems Analysis to compare and evaluate projects. These techniques have been developed because the systems that were being created had a level of complexity that could not be fully understood with the knowledge that was taught by the classical schools of Engineering and Management (Hughes, 2005).

Since its formalization, Systems Engineering has been an activity based on practice; it is considered more a method than a discipline founded in books and formalisms. Unlike the traditional disciplines of engineering, Systems Engineering does not follow a set of fundamental phenomena based on physical properties and relations; instead, it is associated to knowledge to

orchestrate these physical properties and relations, to deal with system emergent properties.

Hitchins (2008) argues that Systems Engineering view systems as dynamic and open, potentially adaptable to other systems in the same environment, and capable of showing emerging properties, capabilities and behaviors. This approach emphasizes the dynamic interaction not only among the parts of the system, but also among a system and systems external to it. The emphasis is on performance, features, functionality, and dynamic processes. Hitchins also argues that when engineers deal with simple systems, the results of classical engineering approach or Systems Engineering approach may be similar, but the result in complex systems is different.

Kossiakoff & Sweet (2003), Sydenham (2004), INCOSE (2006), and Wasson (2006) define System Engineering using significant words, such as: interdisciplinary, iterative, socio-technical, and whole. However, these words refer to how System Engineering must be done; they are not a term definition. The authors of this work use the definition given by Hitchins (2008): "*Systems engineering is the art and science of creating whole solutions to complex problems*."

Software Engineering

The term "Software Engineering" was first coined in 1968, in a conference hosted by NATO Science Committee, devoted to the subject of producing large and complex software systems: The NATO Software Engineering Conferences (Naur & Randell, 1969). The Engineering metaphor was not selected at random. Engineering is about the functions, build, structures and architecture, and it aims to have a whole equal to the sum of the parts. Sommerville (2011) states that Software Engineer is an engineering discipline that is concerned with all aspects of software production, and that it encompasses a set of three fundamental elements: Methods, tools and procedures to enable engineers to control all the aspects of the software development process, a basis for the construction of high quality software. Software Engineering encompasses a process, a collection of methods (practice) and a collection of tools that allow professionals to build high quality software (Pressman, 2010).

IEEE (2010) specifies that there is the presence of scientific knowledge and experience during the software system development process, stating that Software Engineering is the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing and documentation of software.

Just like Systems Engineering, Software Engineering definitions refer to how it must be done. The SEMAT (Software Engineering Methods and Theory) Call for Action states that as it is today, Software Engineering is suffering from immature practices, and that it is necessary to redefine Software Engineering based on a solid theory, proven principles, and best practices (Jacobson at all, 2012). SEMAT founders evolved the call for action into a vision statement and, in accordance with this vision, SEMAT focused on the development of a foundation for Software Engineering, consisting of a kernel and a language to define methods, practices and kernel elements.

The first step of SEMAT was to identify a common ground for Software Engineering, a kernel of essential elements that are universal to all software system development efforts, and a simple language for describing methods and practices. The essential elements compose a kernel that contains a small number of "things we always work with" and "things we always do" when developing software systems. These elements and the relationship between them occur in three areas of concern: Customer, Solution, and Endeavor, as represented in Figure 1.

Figure 1 shows the seven essential elements present in a software system development project. They are called alphas (Alpha is an acronym for Abstract-Level Progress Health Attribute). An alpha is characterized by a set of states that represent their progress and health. As an example, the alpha Work moves through the states of: initiated, prepared, started, under control, concluded, and closed. Each state has a checklist that specifies the criteria that the system development team must identify to determine and control the current state of the alpha. This checklist is also used to identify what to do to reach the next state. These states indicate the progress and health of the endeavor to steer the project to a successful conclusion (Jacobson at all, 2013; SEMAT Community, 2013).