Software and Systems Engineering Integration

Rick Gibson
American University, USA

INTRODUCTION

With software an increasingly significant component of most products, it is vital that teams of software and systems engineers collaborate effectively to build cost effective, reliable products. This article will identify the key aspects of software engineering and systems engineering in an effort to highlight areas of consensus and conflict to support current efforts by practitioners and academics in the both disciplines in redefining and integrating their professions and bodies of knowledge.

In response to increasing concerns about software development failures, the Software Engineering Institute (SEI) pioneered a software process improvement model in 1988, with the fully developed version of their Capability Maturity Model for Software (SW- CMMâ) appearing in 1993. Since the early nineties, there have been comparable improvement models introduced in the systems engineering community as well, some of which have been published and widely accepted include: Systems Engineering Capability Maturity Model (SE-CMM), also known as the Electronic Industries Alliance Interim Standard (EIA/IS) 731, Systems Engineering Capability Model (SECM), and the Integrated Product Development Capability Maturity Model (IPD-CMM). The resulting avalanche of models and standards has been described by Sarah Sheard (Software Productivity Consortium) as a "Framework Quagmire". In December of 2000, the SEI initiated the Capability Maturity Model–Integrated (CMMI^{SM}) project, which combines best practices from the systems and software engineering

Table1. Software and system engineering similarities and differences

Similarities	Differences
Definition and analysis involves manipulation of symbols.	Software is not subject to physical wear or fatigue.
Highly complex aggregation of functions, requiring satisfying (though not optimizing) multiple criteria.	Copies of software are less subject to imperfections or variations.
Decisions driven by need to satisfy quality attributes such as reliability, safety, security, and maintainability.	Software is not constrained by the laws of physics.
Easy and dangerous to suboptimize solutions around individual subsystem functions or quality attributes.	Software interfaces are conceptual, rather than physical—making them more difficult to visualize.
Increasing levels of complexity and interdependency.	Relative to hardware, software testing involves a larger number of distinct logic paths and entities to check.
	Unlike hardware, software errors arrive without notice or a period of graceful degradation.
	Hardware repair restores a system to its previous condition; repair of a software fault generally does not.
	Hardware engineering involves tooling, manufacturing, and longer lead times, while software involves rapid prototyping and fewer repeatable processes.

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

disciplines. (Note: CMMâ and CMMI[SM] are copyrights and service marks of the Software Engineering Institute.)

Recent studies (Carter et al., 2003; Goldenson & Gibson, 2003) have validated the SEI's assertion the each of the disciplines benefit from incorporation of principles from the other. Moreover, there appears to be no fundamental differences between the disciplines that would prevent their integration.

BACKGROUND

There is great hope that the SEI imitative will provide the impetus to overcome some long-standing discipline boundaries. The nature of the systems and software engineering work has led to terminology differences rooted in the very descriptions of the disciplines. One important problem with software is the difficulty in understanding its inherent level of quality.

Issues and concerns regarding such an integration were articulated by Barry Boehm and Fred Brooks as early as 1975. Boehm suggested that the adoption of systems engineering reliability techniques by software engineers was counterproductive. Moreover, Brooks' Law suggests that a common systems engineering solution to schedule slippage (add more people) will only make late software projects even later.

More recently, Boehm (1994) expressed concerns that, in spite of the central function of software in modern systems, the two engineering disciplines have not been well integrated. Boehm articulated similarities and differences as shown in Table 1.

Software engineering, as defined by the Institute of Electrical and Electronics Engineers (IEEE, 2001), is: (1) the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, that application of engineering to software; (2) The study of approaches as in (1)—and further identifies the body of knowledge for software engineering to be: software requirements, software design, software construction, software testing, software maintenance, software configuration management, software engineering management, software engineering process, software engineering tools and methods, and software quality.

A useful definition of systems engineering resides in an in-process body of knowledge document by the International Council on Systems Engineers (Leibrandt, 2001, p. 3), which defines systems engineering in terms of product and process: "…product oriented engineering discipline whose responsibility is to create and execute an interdisciplinary process to ensure that customer and stakeholder needs are satisfied in a high quality, trustworthy, cost effective and schedule compliant manner throughout a system's lifecycle". The process starts with customer needs, and consists of stating the problem, investigating alternatives, modeling, integrating, launching the system, and assessing performance. Moreover, the system engineer is responsible for pulling together all the disciplines to create a project team to meet customers' needs. The complete systems engineering process includes performance, testing, manufacturing, cost, schedule, training and support, and disposal. The body of knowledge recognizes that systems engineering processes often appear to overlap software and hardware development processes and project management. Thus, systems engineering is a discipline that focuses on processes; it develops structure, and efficient approaches to analysis and design to solve complex engineering problems. In response to concerns about integrated development of products, the system engineer plans and organizes technical projects and analyzes requirements, problems, alternatives, solutions and risks. Systems engineering processes are not specific to a particular discipline; they can be applied in any technical or engineering environment.

In short, software engineering is defined by IEEE Standard 610.12 as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software—that is, the application of engineering to software. Eisner (2002) adopts the International Council on Systems Engineering (INCOSE) definition of systems engineering as an interdisciplinary approach and means to enable the realization of successful systems.

When different process models are in place within developer groups, say for systems engineering and software engineering of an organization, the organizations will have communication problems, be unable to improve their processes, and if the combined performance of one advances beyond the other in capability, then the problems are even more profound (Johnson, 1998).

In 2002, the SEI released a single integrated capability model for systems engineering and software engineering, integrated product and process development and supplier sourcing. The new model, Capability Maturity Model Integrated (CMMI), is intended to improve organizations' development and maintenance of products. The CMMI will eventually replace the SEI's Software Capability Maturity

4 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/software-systems-engineering-integration/14099

Related Content

Learning IT: where do Lectures Fit?
Tanya McGilland Samantha Bax (2008). *Information Communication Technologies: Concepts, Methodologies, Tools, and Applications (pp. 2708-2717).*
www.irma-international.org/chapter/learning-lectures-fit/22843

A Simulation-Based Decision Support System for Managing Information Technology Project Portfolios
Zohar Lasloand Gregory Gurevich (2013). *International Journal of Information Technology Project Management (pp. 1-17).*
www.irma-international.org/article/simulation-based-decision-support-system/77875

The Inclusion of CIOs in Top Management Teams: A Longitudinal Study of the Strategic Role of IT
Wenhong Luo (2016). *Information Resources Management Journal (pp. 37-52).*
www.irma-international.org/article/the-inclusion-of-cios-in-top-management-teams/163243

Minorities and the Digital Divide
Lynette Kvasnyand Fay Cobb Payton (2005). *Encyclopedia of Information Science and Technology, First Edition (pp. 1955-1959).*
www.irma-international.org/chapter/minorities-digital-divide/14544

Decision-Tree Models for Predicting Time Performance in Software-Intensive Projects
Nermin Sökmenand Ferhan Çebi (2017). *International Journal of Information Technology Project Management (pp. 64-86).*
www.irma-international.org/article/decision-tree-models-for-predicting-time-performance-in-software-intensive-projects/177292