Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 44 (2015) 708 – 717

Procedia Computer Science

2015 Conference on Systems Engineering Research

Exploring the Relationship between Systems Engineering and Software Engineering

Art Pyster[a,]*, Rick Adcock[b], Mark Ardis[a], Rob Cloutier[a], Devanandham Henry[a], Linda Laird[a], Harold ‘Bud’ Lawson[c], Michael Pennotti[a], Kevin Sullivan[d], Jon Wade[a]

[a]*Stevens Institute of Technology, Hoboken, NJ 07030, USA*
[b]*Cranfield University, UK*
[c]*Lawson Konsult AB, Stockholm, Sweden*
[d]*University of Virginia, VA, USA*

Abstract

In an effort to explore the relationship between the disciplines of systems engineering and software engineering, professionals from academia, industry, and government gathered for a workshop to deliberate on the current state, to acknowledge areas of inter-dependence, to identify relevant challenges, and to propose recommendations for addressing those challenges with respect to four topical areas: 1) Development Approaches, 2) Technical, 3) People, and 4) Education. This paper presents the deliberations and recommendations that emerged from that workshop, and the proposed project to be launched.

Keywords: systems engineering; software engineering; development approaches; people; education

1. Introduction

Software is fundamental to the performance, features, and value of most modern engineering systems. It is not merely part of the system, but often shapes the system architecture; drives much of its complexity and emergent behavior; strains its verification; and drives much of the cost and schedule of its development. Given how significant an impact software has on system development and given how complex modern systems are, one would expect the relationship between the disciplines of systems engineering (SE) and software engineering (SWE) to be well defined. However, the relationship is, in fact, not well understood or articulated.

doi:10.1016/j.procs.2015.03.016

In order to examine this inter-relationship between the two disciplines, 29 professionals from academia, government and industry gathered for a workshop at Stevens Institute of Technology in Hoboken, NJ on June 12-13, 2014. The motivation for the workshop was:

- Most of today's interesting software is tightly integrated with hardware in systems that must operate in the physical world. This tight coupling of systems and software creates many challenges, including for system architecture; integration and verification; and development cost and schedule.
- SE and SWE could be practiced in a way that reflects their mutual dependence, but there are many challenges to doing so. For example, modern software development methods largely favor rapid and agile development, while hardware development methods often must address lifecycles with longer lead times and that adapt more slowly to change; and large companies often silo systems engineers and software engineers into different departments, with separate career paths, training and tools.
- SE and SWE academic programs could be structured to reflect their mutual dependence, but usually aren't. There are few undergraduate or graduate degrees we are aware of, that are offered jointly between SWE and SE programs; and we are aware of a number of universities where the relationship between the SWE and SE programs is strained at best.
- A focused workshop to explore these challenges in both practice and education could spawn efforts that might help address them.

Four topical areas were identified for further exploration during the workshop:

1. *Development Approaches*: the ways in which systems and software engineers could collaborate on lifecycle approaches that emphasize speed, agility, and other increasingly important characteristics, and how SE and SWE relate to other management and technical disciplines
2. *Technical*: the relationship between SE and SWE methods, processes and tools
3. *People*: the relationship between those who perform SE and those who perform SWE; their personalities and career paths; and the relationship between how they are organized, measured, rewarded and motivated
4. *Education*: the relationship between how systems engineers and software engineers are educated and what they learn; the relationship between how universities organize their curricula, faculty and other resources; what systems engineering should be included in software engineering curricula and vice versa

During the workshop, the plenary session included two keynote speakers who each offered their perspectives on the relationship between SE and SWE; and also included four panel sessions, one on each of the topical areas, that explored: 1) the current state of that area; 2) perceived shortfalls and root causes of those shortfalls; and 3) specific efforts that could address those shortfalls. The keynote speeches and the panel sessions set the stage for the primary workshop activity in which participants divided into four groups, one for each topical area, and deliberated more deeply into each topic.

This paper integrates the findings and recommendations of the several workshop sessions; expands and refines those findings and recommendations based on the white papers submitted by many of the workshop attendees in advance of the workshop itself; and expands and refines exchanges among the workshop leadership that took place subsequent to the workshop itself.

2. The Changing Nature of Systems and Software and its Impact on the Systems and Software Engineering Disciplines

As observed at the workshop by Dr. William Scherlis, "Software is the 'building material of choice' for today's complex systems." This has impacted the nature of work for both systems engineers and software engineers. The growing dominance of computation as a source of functionality, novelty, complexity and risk in the conception, design, operation and evolution of modern systems, and of software as a primary material for the construction of such systems, are significantly increasing the overlap in the roles, responsibilities and required expertise of the historically separate disciplines of SWE and SE. Even "classic" software programs, such as those used to plan enterprise resources or manage inventories, are typically components of larger enterprise systems. Increasingly ubiquitous cyber-physical systems depend on tightly integrated hardware and software (e.g., smartphones, automobiles, and even electric razors), while socio-technical systems typically contain massive amounts of software, hardware, organizations and people (e.g., healthcare systems, infrastructure systems, and education systems). In a world in which systems and software are so inextricably linked in a fundamental way, the disciplines of SE and SWE must inform each other, and practicing systems and software engineers must work seamlessly with each other. Indeed, we must revisit the whole notion of how systems engineers and software engineers are educated, developed and managed, and the methods, processes and tools they use.

Analyzing the current state of the art and practice in SWE and SE has to begin with an attempt to clarify the nature and roles of the two disciplines in major systems projects. This analysis helps characterize the sources and nature of the overlaps that are arising from a combination of the histories of the fields and current trends in technology, and the key gaps in capability that remain inadequately addressed today.

A clear, shared understanding of the roles, responsibilities and competencies of SWE and SE, of their relationships to each other, and of the gaps that the two fields together do not yet adequately address, remain somewhat elusive. Understanding the current state of the art and practice and the major remaining gaps requires thoughtful analysis of these issues.

To support this analysis, we introduce a two-dimensional ontology into which we map the two disciplines as they exist today. The first dimension distinguishes between the vertical and horizontal dimensions of a system. The second dimension distinguishes three classes of systems that we call physical, computational and cyber-physical. Mapping the two disciplines as they exist today into this framework serves to clarify the sources and nature of the increasing overlap between them and the key areas that remain inadequately addressed.

The overlap and the gaps that remain between the capabilities of the two disciplines and what the effective development of modern systems actually require must be much better understood and addressed to enable effective development of future systems.

2.1. Horizontal and Vertical Engineering Aspects of Systems and Disciplines

We begin by separating what we call the horizontal and vertical dimensions of systems, and the corresponding horizontal and vertical engineering disciplines. The vertical dimensions of a system are those that modularize around technically focused engineering disciplines. For example, electrical engineering focuses on electrical and electronic aspects of systems, mechanical engineering on mechanical aspects, chemical engineering on chemical aspects, etc. We thus refer to these disciplines as being vertical, or at least as playing vertical roles in most complex systems projects.

The horizontal technical dimensions of a system, by contrast, involve crosscutting concerns at the systems level. Such concerns include evolving customer preferences that impact across entire systems: systems-level quality attributes, tradeoffs and optimization; system architecture, decomposition and integration issues; system development processes; and system economics: cost, schedule and risk. We use the term horizontal to characterize these technical concerns, and to characterize the engineering disciplines that address them.

2.2. Three Classes of Systems

The second component of our ontology separates systems into broad classes, distinguished by the primary sources of novelty, functionality, complexity and risk in their conception, development, operation and evolution. The classification is based on the extent to which computation and software are the principal drivers in these dimensions. We call the three classes of systems physical, computational, and cyber-physical to reflect the technological "center of gravity" within the system. The relationship between SE and SWE is most important in large-scale computational systems and in all but the most trivial cyber-physical systems.

2.2.1. Physical Systems

The first class of systems are those in which physical aspects are the main sources of novelty, functionality, complexity and risk at the overall systems level, i.e., in the horizontal dimension. The primary purpose of these systems is to operate on and generate matter or energy. We call such systems physical systems. While they often utilize computation and software technologies as components, those components are not dominant in the horizontal dimension of engineering. Rather, in such systems, they are viewed and handled as vertical concerns.

Examples of such systems include older generations of a broad range of systems: bridges, vehicles, weapons systems, and healthcare delivery systems from the past that were relatively "dumb" compared to today's "smart" systems. Challenges in the horizontal dimension of engineering are driven by complexity in such areas as physical structure and tradeoffs involving form factor lock-in, energy and dynamics, properties of materials and structures, analog control and system economics.

With respect to physical systems, SE has long been involved in the full range of horizontal engineering activities, created to bridge the gaps between the more traditional engineering disciplines. Such horizontal activities include requirements engineering, systems architecture, specification and balancing tradeoffs among quality attributes, human factors issues and SE economics. SE's roots go back to the development of systems in which software either did not exist or was simply not central to system operation and success. SWE has played a much smaller role in such systems, limited to whatever software components may be present rather than overall system performance and characteristics.

2.2.2. Computational Systems

The second class of systems includes those in which computational behavior and, ipso facto, software are the dominant sources of functionality, complexity, novelty and risk at the systems level. The primary purpose of these systems is to operate on and produce data and information. We call such systems computational. While these systems always include physical and human elements, these are not the predominant challenges in system development, operation and evolution. Examples include operating systems, database management systems, systems middleware, desktop software systems and even IBM's "Watson." The major challenges in such systems arise from the difficulties involved in conceiving, realizing and evolving complex computational behaviors and their software representations.

Whereas in physical systems, software and computational elements are largely addressed as vertical software engineering issues, a very different picture emerges in computational systems. In these systems, for all intents and purposes, software and computation are the system. The individual software components of such systems are still best viewed as presenting vertical software engineering challenges. However, for this class of systems, software engineering has also historically addressed the full range of horizontal engineering challenges; i.e., software engineers have typically performed SE activities relying on methods, processes and tools that were tailored for the engineering of computational systems; e.g., architectural representations in which physical constraints were relatively unimportant or buffered through a layered architecture, or perhaps not captured at all, instead focusing on software artifacts, their behaviors and their interactions with users.

In addition to being a vertical discipline, SWE has historically been the systems engineering discipline for computational systems. The term software systems engineering has often been used to describe the blend of SE and SWE for these systems and the term software systems engineer used to refer to its practitioners. For computational systems, software systems engineers typically ascertain evolving customer preferences, specify and manage tradeoffs among quality attributes, develop the software systems architecture, define the overall development process and perform engineering economics. They are often responsible for handling SE issues for computational systems.

SWE is rare or unique among engineering disciplines in having both a vertical role in physical and computational systems and the major horizontal role in computational systems. The source of the increasing overlap between the disciplines of SWE and SE now becomes clear. First, both fields have historically evolved to play overall SE roles, albeit for two historically separate classes of systems. Second, the separation of these classes is now disappearing. Rather than purely computational systems with relatively simple human-physical components, or physical systems with relatively simple computational/software elements, we are now entering an era of incredibly smart cyber-physical systems that depend on the synergistic collaboration of hardware and software for their functionality. In these systems, physical and computational elements are nearly equal partners in realizing the system objective. Neither SE nor SWE has traditionally addressed the horizontal complexities of systems in this realm, and neither discipline is adequately equipped by itself to do so.

2.2.3. Cyber-Physical Systems

The third class of systems, cyber-physical, has a complex combination of computational and physical dimensions. Such systems are innovative, functionally complex and risky in both their cyber and physical dimensions. They pose major horizontal engineering challenges across the board. Examples of cyber-physical systems increasingly abound – smart automobiles, power grids, robotic manufacturing systems, defense and international security systems, supply-chain systems, the so-called internet of things, etc. In cyber-physical systems, cyber and physical elements collaborate in complex ways to deliver expected system behavior. The systems and software engineers who create them must do likewise.

Challenges abound for those who conceive of new cyber-physical systems and then develop, manufacture, operate and evolve them; e.g., adapting them at a rate that can keep up with outside opportunities and threats in technology, customer preferences, politics and competition; dealing with unanticipated interactions between the system and its external environment; responding to unanticipated and emerging behaviors of the system itself; maintaining critical system quality attributes, such as system security and safety; and mitigating against the brittleness and complexity in both the system and the development environment that can come from making such changes.

Cyber-physical systems pose the greatest challenge to SE and SWE and to their practitioners. For example, many university programs in SWE have grown out of computer science and math departments and require little in the way of classical engineering courses in physics, electronics, chemistry, and mechanics. Analogously, many university SE programs have grown up in classical engineering departments and require little in the way of software engineering expertise from their students. The result is that graduates of SE programs are well prepared to work on physical systems, graduates of SWE programs are well prepared to work on computational systems, but far fewer graduates are well prepared to work on cyber-physical systems. Industry must then compensate for the "silo-ed" education of such graduates by offering them broadening assignments, mentoring and training, but companies may lack the ability to do this easily and well.

There are many practical impacts of the gap between SWE and SE, especially for cyber-physical systems. For example, software and hardware tend to evolve at different speeds, causing many changes that occur throughout the system lifecycle to be implemented in rapidly evolving software. Mature lifecycle models and supporting tools that fully reflect the different speeds at which hardware and software evolve do not exist. Methods, processes and tools to understand and predict system-wide behavior in such systems caused by even small changes in software are lacking.

More specific aspects of the challenges faced by the two disciplines are explored in the next two sections.

3. The Relationship between Systems Engineering and Software Engineering Educational Programs

Today most universities manage SE and SWE programs separately, leading to separate masters and doctoral degrees. Such programs are often housed in different academic units; e.g., software engineering degrees are often offered by computer science departments that are not part of an engineering school; systems engineering degrees, on the other hand, are usually offered within an engineering school, sometimes by a standalone systems engineering department, but often by another department such as industrial engineering.

SE and SWE programs could draw from two community-developed bodies of knowledge: the Guide to the Systems Engineering Body of Knowledge (SEBoK)[1] and the Guide to the Software Engineering Body of Knowledge (SWEBOK)[2]; and they could draw from related curricular recommendations, the Graduate Reference Curriculum on Systems Engineering (GRCSE)[3] and the Graduate Software Engineering 2009 (GSwE2009)[4]. All of these documents attempt to draw out some of the relationships between SE and SWE, but none do a complete job and none are universally used. Given the critical role that software plays in today's most interesting systems, the academic separation of SE and SWE programs is artificial and leads to students graduating with missing skills, understandings and perspectives that are important to their success on the job.

We believe that there is sufficient differentiation between the SE and SWE disciplines that separate degree programs continue to be appropriate, but also believe that new programs should be developed that merge the education of these two disciplines, especially to educate students about the differences between cyber-physical systems and the systems the respective disciplines have traditionally addressed. Such an integrated SE-SWE degree will require instructors who have broad experience, as well as substantial cooperation from industry.

We see the major outcome of such an integrated educational program as the development of individuals who possess:

- T-shaped skills; i.e., having breadth in many technologies, systems types and disciplines and depth in a small number of them; and,
- Individualized and deep skills that support self-paced and life long learning, yet also support broad-based team-orientation to foster both leadership and follower-ship.

Moreover, we believe that elements of such a program would be beneficial for all engineers, not just systems engineers or software engineers.

Realizing that such a desirable future is a fairly tall order, we believe the best approach to developing such SE/SWE educational programs at the graduate level is to create a technical master's degree with a solid dose of technical leadership that is somewhat analogous to what would be found in an MBA. Following the typical MBA educational model, we believe that candidate students who want to enter such a program should come with strong technical expertise derived from a mix of prior education and practical experience. A new SE/SWE educational program should draw from many of the classroom mechanisms used in the MBA context, such as case studies, serious games and group experiences.

There are several critical factors for a successful integrated SE-SWE education, including:

- Offering group experiences to students in a realistic environment – SE and SwE are team sports.
- Relying on instructors with broad experiences, including using guest lecturers to supplement the experiences of the primary instructors.
- Mentoring students on soft skills such as communications, negotiation and leadership.
- Instructing with a balance between realism and openness, constraining students to ensure that they learn intended skills.
- Instructing students on historical approaches that are no longer popular with an emphasis on their limitations and why they have fallen out of use.

SE-SWE programs should be based on several guiding principles, including:

- Principles-based: Technologies change rapidly, but the underlying principles by which systems are developed and managed do not.
- Foundation stones: Repeat fundamental ideas in real world contexts to reinforce learnings.

- Agility: Quality course material should be developed in a way that quickly responds to rapid changes in the SE and SWE environment.
- Accelerated Learning: Compress multi-year lifecycle experiences into a much shorter period of time consistent with university schedules.
- Technology Savvy: Uses multi-media and multiple modes, such as gaming, online courses, social media, peer assessment and crowd sourcing, to provide a rich, interactive, effective and efficient educational experience.
- Context is critical: What we do, what is important, and how hard our actions are all depend on context, including the system and life-cycle state, the domain of the system, the nature of the system itself and the balance between opposing system properties.
- Reflection and analytical skills: These complement specific technical skills. We must balance learning from the past with anticipating the future.
- Continuous Life-long learning: Students must learn how to continuously learn and to appreciate the importance of doing so.
- T-Shaped Practitioners: Students must learn how to master breadth in many technologies, systems types and disciplines, while developing depth in a small number of them. This will require the ability to learn at one's own pace using any available materials, seeking mentors and having strong skills in leadership, follower-ship, communication, active listening, negotiation and other soft skills.
- Integrated: Instructors must offer a setting that encourages the integration of multi-disciplinary skills and a wide range of SE and SWE knowledge in a setting that recreates the essential characteristics of the practicing environment.

There are a number of knowledge gaps that need to be filled to support an integrated SE/SWE education, including:

- How to integrate the principles, tools and techniques of SE and SWE around such areas as measurement, architecture, technical debt, complexity, emergence and quality tradeoffs in ways that reflect scaling up and down, application to different domains and operating under different constraints.
- How to effectively use smaller in-class projects to offer important insights into the characteristics and challenges of large scale projects – including learning about specialty areas, such as contracts, learning about later phases of the life cycle, such as operations and maintenance, and thinking from an enterprise perspective, such as conceiving of a product line.
- How to teach soft skills and team dynamics in a form relevant for students in classroom settings that are heavily constrained.
- Creating an extensive, widely available array of case studies that expose students to the challenges of developing, fielding and operating diverse systems in many domains, while relying on diverse technologies and operating under a variety of constraints.

4. The Relationship between Systems and Software Engineering Practice

We identified several key characteristics of the current state of practice among systems engineers and software engineers. Broadly speaking, these factors address the way systems engineers and software engineers are developed, the way they are organized and the way they are measured, as well as the processes they follow and the tools they use.

Perhaps somewhat surprisingly, but consistent with the theme that Jeff Wilcox struck in his keynote address at the workshop, we found that there are far more similarities between systems engineers and software engineers in these regards than there are differences.

4.1. How they are Developed

- Systems engineers and software engineers are usually educated in different departments and often in different schools within their universities. They have little opportunity to get to know each other, let alone to collaborate during their educational endeavors. In fact, the same is often true for the professors who teach them.
- While systems engineers often start out in traditional engineering fields such as electrical and mechanical engineering, and software engineers frequently begin by developing code, by the time they assume responsibilities at the systems level, both are typically engaged in common activities such as architecting and integrating, rather than focusing on detailed design.
- There has been an alarming decline in the diversity of entry-level software engineers over the past 25 years. This is particularly true in the case of women, the percentage of whom among computer science graduates has fallen from 30% to 12% over that period.
- The career growth of both systems engineers and software engineers resembles that of business leaders more than it does that of traditional engineers. It is often leadership skills and competencies, such as influencing and persuading that are most responsible for their success as individuals and the success of their teams.
- Often systems engineers and software engineers must rely on their power of influence rather than positional authority to have an impact. Communications, negotiating and other leadership skills are central to that influence. However, those seeking roles as systems and software engineers may undervalue the importance of these skills relative to traditional technical skills.

4.2. How they are Organized

- When they enter the workforce, both systems engineers and software engineers often find themselves organized into the same type of stovepipes they encountered during their educational careers. Housed in separate departments, they once again have few opportunities to collaborate and to learn from one another.
- The difficulties that systems engineers and software engineers have in communicating with each other are compounded by the fact that project and business managers often don't understand what either does. These managers are thus in a poor position to facilitate communications and collaboration between them or to mediate their differences.
- Systems engineers and software engineers play quite varied roles, often without clear career paths or clarity about those roles. This makes it difficult for those who work with them to understand what to expect from them. Many organizations do not use the title "systems engineer" or "software engineer" for people who perform SE and SWE activities, making it harder for those around them to understand what they do or to create an esprit de corps among them.

4.3. How they are Measured

- Both individuals and organizations are frequently rewarded for adherence to established processes and precedents, rather than for risk-taking and innovation. This limits their adaptability in the face of rapidly changing market conditions and evolving enabling technologies.
- Despite continually stressing the importance of teamwork, managers continue to reward heroes for fixing problems, rather than effective team leaders who prevent the occurrence of such problems in the first place.
- When issues arise, as they inevitably do, systems engineers and software engineers too often find that more energy is expended on assigning blame than on solving problems.
- Cost and schedule metrics are often emphasized over measures of technical performance.
- Software metrics often focus more on how long it takes to write the code than on how long it takes for the code to work.

4.4. How they Work

- Both systems and software engineering have become more specialized, fragmenting both disciplines, diluting their ability to drive holistic systems solutions and driving a communications wedge between their various sub-specialties
- Both systems engineers and software engineers are being required to become more agile rather than to rely on traditional, plan-driven processes.
- SE and SWE tools significantly lag in maturity and capability when compared to the tools used by other "classical" engineers, such as mechanical and electrical engineers.
- There is a growing body of literature exploring the integration of SE and SWE methods, processes and tools. Among the earliest are Barry Boehm's 2005 paper[5] and Mark Maier's 2006 paper[6]. More recent work includes publication of the Software Engineering Method and Theory (SEMAT) and its supporting kernel and language for software engineering methods, ESSENCE; Boehm also recently published a book[7]; and a new book edited by Ivar Jacobson and Bud Lawson is expected soon, entitled 'Software Engineering in the System Context'.

5. Conclusions

The workshop on which this paper is based involved a cross-section of the systems and software engineering community, primarily from academia and those in the aerospace and defense community. We recognize that the ideas presented here are a work in progress. The report is a summary, consensus, and synthesis of views that emerged from the workshop and subsequent discussions among its participants and leaders. Building on the analysis and findings in this workshop report, a project co-sponsored by INCOSE and SERC is expected to be launched in early 2015. The objective of this project will be to gather data from the broad community of systems engineering and software engineering educators, practitioners, consumers, and employers about current practices, challenges, and gaps in education/training, people, and technology/development approaches. Data will be gathered from across a broader community than was represented at the workshop, to include business sectors outside aerospace and defense, which may have different perspectives on the ideas generated at the workshop. Based on the findings of this project, a roadmap will be laid out with relevant recommendations; multiple additional projects are then expected to follow.

Acknowledgements

The authors wish to thank the Systems Engineering Research Center (SERC) and the International Council on Systems Engineering (INCOSE) for co-sponsoring the workshop on systems engineering and software engineering; and the School of Systems and Enterprises (SSE), Stevens Institute of Technology for hosting the workshop. The authors acknowledge the contributions of all participants and their parent organizations, who participated in the workshop; especially, keynote speakers Dr. William Scherlis from Carnegie Mellon University and Mr. Jeff Wilcox from Lockheed Martin.

Appendix A. List of Workshop Participants

Anne O'Neil, INCOSE
Art Pyster, Stevens Institute of Technology
Barry Boehm, University of Southern California
Cheryl McIntyre, Lockheed Martin
Chris Paredis, National Science Foundation
David Long, INCOSE / Vitech
Devanandham Henry, Stevens Institute of Technology
Dick Fairley, IEEE-CS / Software and Systems Engineering Associates
Doug Schmidt, Vanderbilt

Dov Dori, Technion, Israel / MIT
Gregg Vesonder, AT&T Research
Harold 'Bud' Lawson, Lawson Konsult AB, Sweden
James Clamons, Harris Corporation
Jeff Wilcox, Lockheed Martin
John McDermid, University of York, UK
John Watson, Lockheed Martin
Jon Wade, Stevens Institute of Technology
Kaushik Sinha, MIT
Kevin Sullivan, University of Virginia
Linda Laird, Stevens Institute of Technology
Linda Northrop, Software Engineering Institute
Linda Snow, Rockwell Collins
Mark Ardis, Stevens Institute of Technology
Michael Gries, Rockwell Collins
Mike Pennotti, Stevens Institute of Technology
Ralph Nelson, IBM
Rick Adcock, Cranfield University, UK
Rob Cloutier, Stevens Institute of Technology
Sarah Sheard, Software Engineering Institute
Tom McDermott, Georgia Tech Research Institute
William Scherlis, Carnegie Mellon University

References

1. BKCASE Editorial Board. 2014. The Guide to the Systems Engineering Body of Knowledge (SEBoK), v. 1.3. R.D. Adcock (EIC). Hoboken, NJ: The Trustees of the Stevens Institute of Technology. Accessed 12 July 2014. www.sebokwiki.org. BKCASE is managed and maintained by the Stevens Institute of Technology Systems Engineering Research Center, the International Council on Systems Engineering, and the Institute of Electrical and Electronics Engineers Computer Society.
2. P. Bourque and R.E. Fairley, eds., Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society, 2014 www.swebok.org.
3. Pyster, A., D.H. Olwell, T.L.J. Ferris, N. Hutchison, S. Enck, J. Anthony, D. Henry, and A. Squires (eds.). 2012. Graduate Referenc Curriculum for Systems Engineering (GRCSE®). Hoboken, NJ, USA: The Trustees of the Stevens Institute of Technology. Available at: www.bkcase.org/grcse/.
4. Pyster, A., et al (eds.). 2009. Graduate Software Engineering 2009 (GSwE2009): Curriculum Guidelines for Graduate Degree Programs Software Engineering. Hoboken, NJ, USA: Stevens Institute of Technology. Available at www.gswe2009.org.
5. Boehm, Barry. "The future of software and systems engineering processes." University of Southern California, Los Angeles, CA (2005): 90089-0781.
6. Maier, Mark W. "System and software architecture reconciliation." Systems Engineering 9.2 (2006): 146-159.
7. Boehm, Barry, et al. The Incremental Commitment Spiral Model: Principles and Practices for Successful Systems and Software. Pearson Education, 2014.