Software Engineering and the Persistent Pursuit of Software Quality

Paul D. Nielsen, Software Engineering Institute

Abstract. Software engineering continually seeks to achieve software quality, as the field keeps pace with the changing role of software. The idea for software engineering was conceived nearly 50 years ago when it was observed that the schedule and performance of software used in IT data systems needed to be improved. Today, as software has become the underpinning of the cyber environment and essential to all aspects of DoD system capabilities and operations, software engineering also encompasses new technologies and practices for cybersecurity that enable informed trust and confidence in using information and communication technology.

Introduction

As it keeps pace with software's strikingly expanding and deepening role for the DoD and the Defense Industrial Base, as well as for civil government and industry more generally, software engineering has remained focused on the cause that launched it—the pursuit of software quality.

At the first NATO Software Engineering Conference in 1968, computer science pioneer Edsger Dijkstra captured a major concern about software-based data systems. "The dissemination of knowledge is of obvious value," Dijkstra said, "the massive dissemination of error-loaded software is frightening." The NATO Science Committee invited 50 experts in computer science, such as Dijkstra, to examine a prevailing perception of a "software crisis." The experts saw problems in the reliability of the large software-based data systems of the day, as well as in their cost and schedule management. Discussion at that conference about the crisis gave birth to software engineering [1]. Software engineering thought leaders subsequently pointed to achieving software quality as a solution for software issues. Watts Humphrey captured the perceptions of many leaders when he defined software quality as "a software product" that "must provide functions of a type and at a time when the user needs them" and "must work" [2].

Response to Software Size and Complexity

While observed in the useful (easy, safe, reliable) operation of a software-reliant system, software quality is determined by practices, tools, technologies, and methods that result from software engineering research and development. Software development methodology, for example, emerged as an area of software engineering research and development beginning in the 1970s, in response to the need to achieve quality in larger software-reliant systems. Incremental or iterative methods, such as the Waterfall approach and its modifications, were explored to determine whether activities that promote software quality are accomplished throughout the software creation process. In addition, software program managers demanded improved software cost estimation approaches. Research and development by Barry Boehm led to the initial Constructive Cost Model (COCOMO) in 1981. Boehm based his algorithmic estimation model on a study of several dozen projects of varying sizes. He continued to refine his cost estimation tool to keep pace with advances in software development, producing COCOMO II in the 1990s [3].

A decade or so after software development investigation began, the insight that software process improvement could also contribute significantly to software quality became an avenue for research and development. Process improvement places software development into a broader context, highlighting the organization and its practices and processes as a contributing factor to software quality. The idea of a capability maturity model emphasized that the software organization needed to be improved in order to attain software quality. Years of research into the linkages between organizational practices/processes and software quality culminated in 2002 with the establishment of the CMMI® (Capability Maturity Model Integration) framework in work sponsored by the Office of the Secretary of Defense (OSD) and the National Defense Industrial Association (NDIA) and carried out by the software community.

Around the time that the CMMI framework was first released, Agile software development methods also emerged as an alternative development process. Twelve principles underlie Agile software development, including software-quality-centric precepts such as "Working software is the principal measure of progress" and "Continuous attention to technical excellence and good design" [4]. Defense leaders want the agility and velocity that Agile methods bring, but they also often need assuredness and scale that challenge Agile methods. One active area of research now is how to scale Agile to defense-class systems, a goal that may require adjustments in software engineering and acquisition.[1] In addition, Boehm's early work in balancing agility and discipline, his incremental commitment model, and recent work in DevOps are all efforts to find the right blend of these approaches, a blend that might need to be unique to each project and that might vary during the system's acquisition lifecycle [5].

In the 1990s, software engineering thought leaders also began to define concepts and practices in software architecture, a core activity that permits reasoning about properties that enable or inhibit a system's desired qualities, such as availability and security [6]. Dewayne Perry and Alexander Wolf, for instance, examined architecture's place in software process management in 1992 [7]. A few years later, Mary Shaw and David Garlan wrote Software Architecture: Perspectives on an Emerging Discipline, introducing key abstractions such as components, connectors, and styles [8]. Subsequent research and development in architecture produced a number of models and methods to describe a software architecture and to evaluate it relative to the system's goals for such quality attributes as modifiability, security, and reliability. Philippe Kruchten, for instance, developed a model to view the architecture called 4+1. Kruchten's model gives a description of the system from the viewpoint of users, project manager, and other stakeholders. His 4+1 views are the logical, process, physical, and development views of the system and the use cases for the system [9]. In addition, the Architecture Tradeoff Analysis Method (ATAM) is a widely used method for architecture evaluation. The

primary output of an ATAM is a set of issues of concern about the architecture. When performed early in the development lifecycle, an ATAM has been shown to help a program avoid costly and schedule-consuming problems that might not emerge until the testing phase or even in fielding [10].

Having grown from an "aspiration" and "rallying cry" [11], software engineering today not only features practices, technologies, tools, and methods for software acquisition and development but also boasts of an established body of knowledge [12] and several international standards. In addition, many colleges and universities offer coursework or degree programs in software engineering, and there are scores of professional conferences annually on aspects of the field.

Software and Cyberspace

However, the march forward in software engineering of practices and standards has not resolved all software quality challenges, because software's role continues to grow and deepen. The world has become reliant on software-enabled systems and components. In addition, software is now embedded in the cyberspace domain that enables defense military, intelligence, and business operations [13]. As a result, the DoD is keenly aware of the increasing importance of software and the critical need to achieve software quality.

Software is important for the DoD because it promotes lower cost and improved agility in deploying and reconfiguring systems [14]. One result is reflected in the DoD's ability to now program systems that were once fixed-function to meet changing mission needs. Sensor networks, field programmable gate arrays, software-defined networking, software-defined radios, and embedded controllers represent a few of these now-programmable areas. Another result is that software enables the interconnectivity that is central to accomplishing system-of-systems configurations. Systems of systems support network-centricity, aiding DoD mission goals for information superiority [15]. A third result is that software enables a shift from stovepipe ("platform-centric") systems to modular ("framework and apps") approaches [16]. To exploit the flexibility of modular development, the DoD continues to explore the use of an open systems architecture approach that will shift development focus more to payloads and less to platforms.[2]

The overwhelmingly large role of software in safety-critical air systems (defense and commercial) provides an appropriate illustration. The Air Force vision document Global Horizons traces the percentage of capability in air systems reliant on software through generations of aircraft. By the mid-1970s, when the F-16 went into production, software accounted for about 40 percent of capability. A generation later, the F-22 relied on software for 80 percent of capability. Software may contribute 90 percent of capability for today's premier fighter, the F-35. In addition, millions of lines of software are required to support F-35 Lighting II ground functions [17]. Software's critical role in delivering capability is driving commercial aircraft makers to seek a new development paradigm. The new paradigm follows an architecture-centric "integrate then build" engineering approach rather than the traditional "build then integrate" one in an effort to reduce software rework costs. In the System Architecture Virtual Integration (SAVI) project, aircraft makers and other organizations (including DoD) created a model of software (development and rework) costs. Based on trends and traditional development approaches, the SAVI COCOMO II estimate predicted a cost of $10 billion to develop the millions of SLOC[3] required for aircraft built in this decade, an unaffordable amount [18]. SAVI figures also predict that, without a change, software cost will consume an overwhelming portion of total system costs (see Figure 1). In addition, it is not only in development/rework cost that software looms large. Aircraft now remain in use beyond their original expected service lives. A 2011 U.S. Air Force Scientific Advisory Board study found that the cost of software sustainment for defense weapons systems nearly doubled between 2002 and 2011 [19].

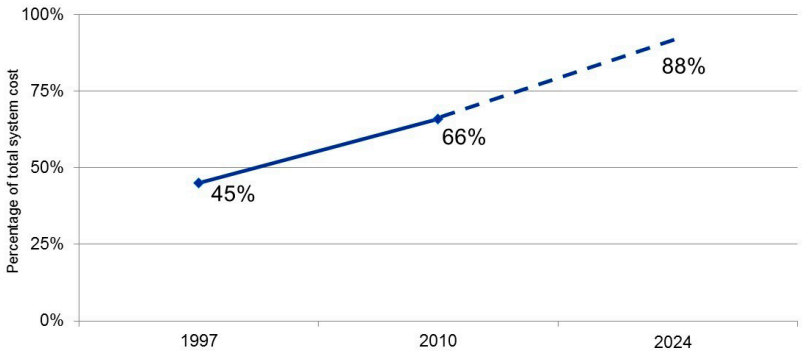

Figure 1. Aircraft Software Development and Rework Cost

With this increased dependence on a software-enabled cyberspace have come new risks and challenges. The size and complexity of software, as well as the interconnectedness of software-enabled systems, mean possible exposure to disruptive, damaging events. Size makes it more likely that software code will include vulnerabilities. Complexity means that organizations may be impacted by emergent behavior—problems almost impossible to foresee during software development or deployment.[4] Outdated legacy code bases, patches installed too late, new applications added to legacy systems, and interdependencies between systems with different levels of software quality—all could reveal hidden vulnerabilities. Legacy system cybersecurity is an acute concern for the DoD, where critical systems are not easy to modify or patch [20].

When mission-critical systems were standalone entities, security was an afterthought. With software engineering practices, tools, technologies, and methods used to produce complex software that delivers advanced, innovative capabilities that are increasingly integrated and interconnected, cybersecurity can no longer be an afterthought in software engineering.

Cybersecurity Expands Software Quality

Indeed, given the defining role of software in the cyber world, software engineering and cybersecurity are now inseparable. Cybersecurity is now not only one of a software system's essential qualities but also a factor that expands the meaning of software quality. The pursuit of software quality now also must consider the risks from potential actions of an adversarial/malicious user throughout the software lifecycle (see Figure 2). Cybersecurity needs to be included in activities from the onset of the acquisition, designed and built into the software system, and considered a prime concern as the system is fielded and sustained [21].

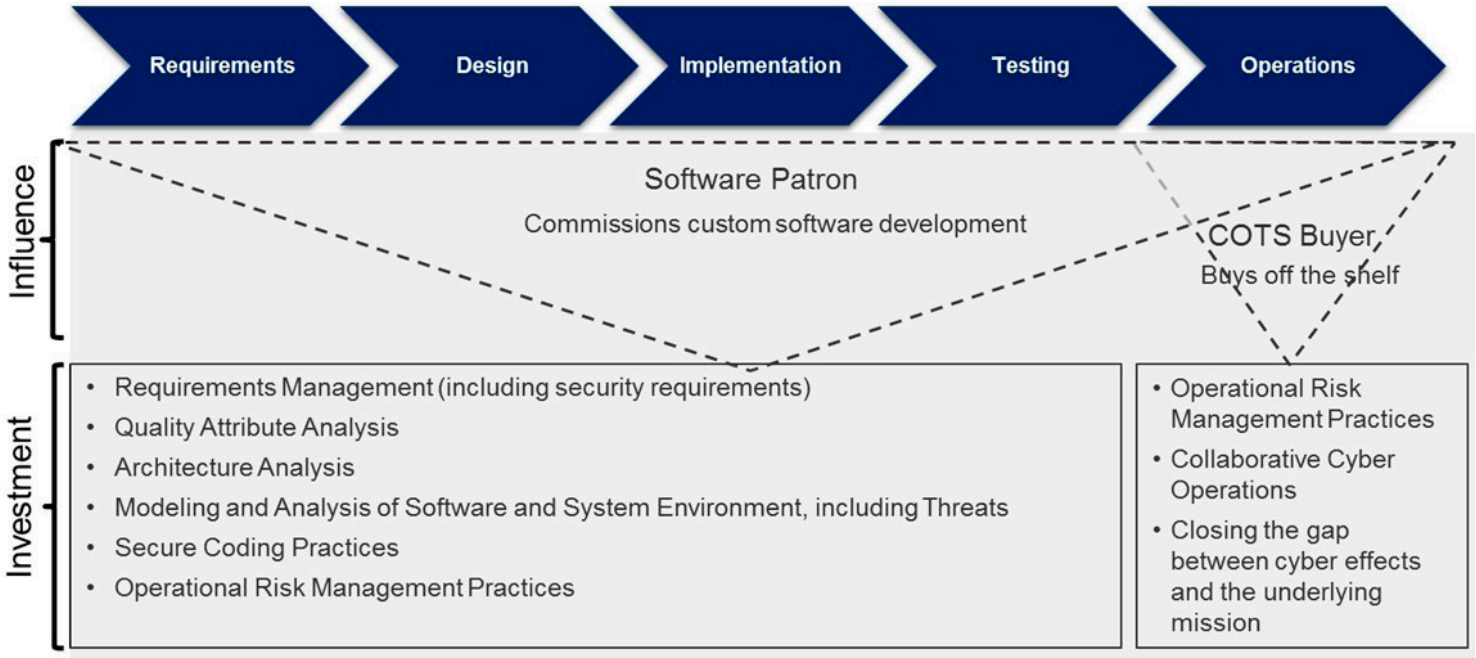

Figure 2. Cyber Risk Must be Addressed Across the Lifecycle for Custom or COTS-based Development

It is vital to approach security requirements in a systematic way early in the lifecycle, in the requirements and design (architecture) phases. Research by Nancy Mead has shown that security requirements can be overlooked or remain implicit until it is much more costly to address them. It is a better practice to perform a risk assessment regarding the system in the context of the expected operating environment and then elicit security requirements [22]. In addition, designing quality architectures involves the use of the fundamental and related concepts—tactics and patterns. Some security tactics or techniques involve detecting, resisting, and reacting to attacks; others aim to help the system recover from attacks [6]. An architectural security pattern is a "piece of design" that provides a proven solution for achieving a particular quality attribute. A typical pattern for system security is authentication and authorization [6]. Including cybersecurity concerns early in development will pay off later on in terms of software quality that is reflected in reliability and maintainability, as well as in user satisfaction.

Likewise, it is important to evaluate for cybersecurity during coding and testing activities. Many exploitable software vulnerabilities occur because of common coding errors. MITRE Corporation sponsors and maintains the Common Weakness Enumeration (CWE) dictionary, under the leadership of Robert Martin. The "common" software flaws, faults, and other errors in code, design, architecture, or implementation in the CWE could result in vulnerabilities others will exploit [23]. These weaknesses—such as buffer overflows, authentication errors, and insufficient data validation—are likely to be as easy to find and mitigate as they are to exploit. Eliminating common vulnerabilities during software development can result not only in more secure software but also in a large cost reduction, because less effort will be expended to repair code. Government, industry, and academic cybersecurity researchers are forming and promoting the adoption of international secure coding standards for some common software programming languages, including C, C++, Perl, and Java[5] [24]. It is important to prevent errors through adherence to secure coding standards; however, rigorous testing is also advisable. For instance, vulnerabilities may emerge as software components are integrated, in commercial off-the-shelf (COTS) and custom-developed software, or in patches sent out to eliminate already discovered vulnerabilities. An advanced level of software testing would include full penetration testing by organic or external experts.

Cybersecurity concerns for software quality must also account for a software supply chain that is diverse and complex—even global. Consider the variety in these supply chains: physical components, integrated components such as network routers, software, the prime contractor organization, subcontractor organizations, and other supply chains for the commercial products used [25]. Each component might be deemed to have sufficient quality, but the integration of components with different levels of software quality ratchets up cybersecurity—and mission—risk for the system [13]. The complicated software supply chain has become an avenue for cyber intrusions, as well. For example, in 2014 a counterfeit, malware-containing Netflix app was preinstalled somewhere in the supply chain on new Android devices available from several vendors [26].

The introduction of malware by a supply chain partner also suggests insider threat concerns. Recent high profile incidents such as Edward Snowden's actions and the Target Corporation breach heighten awareness of the threat that insiders (malicious or unintentional) pose from fraud, sabotage, or theft of intellectual property. While Snowden, working as an NSA contractor, appears to have acted intentionally, the theft of credit card information from Target is reported to have resulted from a mistake by an employee at a supplier that had access for electronic billing to the firm's network [27]. For more than a decade, the CERT Insider Threat Center—collaborating with the DoD, U.S. Department of Homeland Security, the U.S. Secret Service, other federal agencies, the intelligence community, private industry,

academia, and the vendor community—has researched insider threats and built tools for their mitigation. An understanding of insider threat mitigation is especially important for U.S. government agencies working to establish programs to meet Executive Order 13587 requirements.

Finally, despite efforts to assure software quality by preventing software vulnerabilities in development or patching them in system operation, and even with stepped-up insider threat monitoring, it is prudent to assume that systems may be under attack. In operation, a software system may be vulnerable to attack through the exploitation of previously unknown software vulnerabilities (zero-day attack), intrusion into a communications channel (man-in-the-middle), infection of a website visited by a targeted user (watering hole), and other avenues. Thus, an overarching aspect for software quality in the cyberspace domain is operational system resilience. It is appropriate to ask, "What truly needs to be protected? Even when compromised, can the software system continue to deliver capabilities that users need, when they need them?"

Software Quality is a Constant Purpose and Software is a Moving Target

The goal to provide software that must "provide functions of a type and at a time when the user needs them" and "must work" is a fixed point in the software universe. However, the changing and expanding role that software plays in cyberspace means that software engineering has to continue to evolve (even leap ahead) in the ongoing pursuit of software quality. Software engineering now needs to be proactive, not reactive.

When Dijkstra set off an alarm about "the massive dissemination of error-loaded software," systems relying on software touched DoD and other organizations in much more definable ways. Today, military, civil government, industry, and society more generally communicate and socialize in an environment that relies on software-reliant global IT architectures, applications, and services. It is inevitable that demands from users, program managers, and developers for software that delivers greater functionality, reliability, performance, security, autonomy, maintainability, and a host of other attributes will spur innovation and new paradigms that today are not yet conceived. In addition, research and technology trends (see some examples in Table 1) will continue to build on current levels of software complexity and capability. The Internet of Things (IoT), for example, is rapidly approaching, if not in place in some sectors. Already one can see steady progress toward realizing IoT constructs such as the smart grid, smart cities, and smart homes. People and technology will push the frontiers of software engineering forward.

Architecture	Cybersecurity	Process	Workforce	Market
Complexity	Global supply chain security	International standards	Globalization of software development capability	Internet of everything
Cyber-physical systems	Secure coding practices & tools	Data-driven decision-making about practices to use	Supply and demand issues	Autonomy
Strategies for technical debt	Automated software vulnerability discovery	Continuous delivery/velocity	Talent management	Big data/analytics
Affordable sustainment/evolution	Network situational awareness	Blending development and operations	Skills for managers and boards	Software-defined environments
Socio-adaptive systems	Insider threat mitigation	Improving early lifecycle cost estimation	Continuous education	Consumerization
Modeling/virtual integration	Malware analysis & databases	Model-based engineering and auto-code generation tools		Development velocity
Interoperability	Cyber intel for risk management	Assurance planning		
	Adaptive intrusion detection and remediation			
	Active defense			

Table 1. Selected Software Engineering Research Trends

Pursuing software quality in the highly connected (bordering on hyper connected) cyber world may present software engineering with different new frontiers, as well. It could call for greater appreciation of software quality outside the realm of software professionals (developers, architects, programmers, and the like). How can software engineering encourage software quality through broader education at all levels of an organization? For example, what do all employees in an organization now need to know about building a business case for new or updated software systems, securing the global software supply chain, information security/insider threat, appreciating what determines system behavior, or the use of Agile development approaches? Or, how should senior executives incorporate software in their risk assessments?

As software's size, complexity, security and interconnectedness grow, the role of software engineering will become more fundamental to the entire system lifecycle and system-of-systems integration. Software engineering has advanced significantly in its first 50 years, but the continued search for more integrated capabilities opens new opportunities and challenges for researchers, practitioners, and users.

Disclaimers:

Copyright 2014 Carnegie Mellon University

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

This material has been approved for public release and unlimited distribution.

ATAM®, Capability Maturity Model®, Carnegie Mellon®, CERT®, CMM® and CMMI® are registered marks of Carnegie Mellon University. DM-0002001

ABOUT THE AUTHOR

Paul D. Nielsen is Director and Chief Executive Officer of the Carnegie Mellon University Software Engineering Institute (SEI). Prior to joining the SEI, Nielsen served in the U.S. Air Force, retiring as a major general and commander of Air Force research. Nielsen is a member of the U.S. National Academy of Engineering and a Fellow of both the American Institute of Aeronautics and Astronautics and the Institute for Electrical and Electronics Engineers. He has served on many government boards and advisory groups and is currently a member of the Defense Science Board.

Phone: 412-268-5800
E-mail: nielsen@sei.cmu.edu

NOTES

1. See Table 1 for a list of some current trends in software engineering research areas.
2. See, for example, Douglas Schmidt's blog about the Navy's Open Systems Architecture Initiative <http://blog.sei.cmu.edu/post.cfm/importance-automated-testing-open-systems-architecture-062>.
3. Source lines of code
4. See, for example, Douglas Schmidt's blog about the Navy's Open Systems Architecture Initiative <http://blog.sei.cmu.edu/post.cfm/importance-automated-testing-open-systems-architecture-062>.
5. Java is a trademark of Oracle, Inc.

REFERENCES

1. Peter Naur and Brian Randell, eds. Software Engineering: Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968, Brussels, Scientific Affairs Division, NATO, January 1969, pp. 17, 70. <http://homepages.cs.ncl.ac.uk/ brian.randell/NATO/nato1968.PDF>
2. Watts S. Humphrey. A Discipline for Software Engineering. New York: Addison-Wesley, 1995, p. 272.
3. Barry Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece. Software Cost Estimation with COCOMO II. Englewood Cliffs, NJ: Prentice-Hall, 2000.
4. Mary Ann Lapham, Suzanne Miller, Lorraine Adams, Nanette Brown, Bart Hackemack, Charles (Bud) Hammons (PhD), Linda Levine (PhD), and Alfred Schenker. Agile Methods: Selected DoD Management and Acquisition Concerns (CMU/SEI-2011-TN-002). Pittsburgh, PA: Carnegie Mellon University: Software Engineering Institute, 2011, pp. 1, 16 <http://www.sei.cmu.edu/reports/11tn002.pdf>
5. Barry Boehm and Richard Turner. Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley Professional, 2003
6. Len Bass, Paul Clements, and Rick Kazman. Software Architecture in Practice: Third Edition. New York; Pearson Education, 2013, p. 26.
7. Dewayne Perry and Alexander Wolf. "Foundations for the study of software architecture," ACM Sigsoft Software Engineering Notes, vol. 17, no. 4, pp. 40-52, 1992.
8. Mary Shaw and David Garlan. Software Architecture: Perspectives on an Emerging Discipline. Upper Saddle River, NJ (USA): Prentice Hall, 1996.
9. Philippe Kruchten. "Architectural Blueprints–the "4+1" View Model of Software Architecture." IEEE Software, vol. 12, no. 6, pp. 42-50, 1995.
10. Robert L. Nord, John Bergey, Stephen Blanchette, Jr., and Mark Klein. Impact of Army Architecture Evaluations (CMU/SEI-2009-SR-007). Pittsburgh, PA: Carnegie Mellon University Software Engineering Institute, 2009.
11. Mary Shaw. Prospects for an Engineering Discipline of Software (CMU-CS-90-165). Pittsburgh, PA: Carnegie Mellon University, September 1990.
12. IEEE. SWEBOK v3.0. <http://www.computer.org/portal/web/swebok>
13. Department of Defense (DoD). Department of Defense Strategy for Operating in Cyber space. Department of Defense, 2011. <http://www.defense.gov/news/d20110714cyber.pdf>.
14. National Research Council. Critical Code: Software Producibility for Defense. Washington, DC: National Academies Press, 2010.
15. David S. Alberts, John J. Gartska, and Frederick P. Stein. Network Centric Warfare: Developing and Leveraging Information Superiority: Second Edition. Washington, DC: CCRP, 2000.delivering
16. ADM Jonathan W Greenert, U.S. Navy. "Payloads over Platforms: Charting a New Course," Proceedings Magazine, 138/7/1313, 2012. <http://www.usni.org/magazines/proceedings/2012-07/payloads-over-platforms-charting-new-course>
17. U.S. Air Force. Global Horizons: United States Air Force Global Science and Technology Vision (AF/ST TR 13-01). U.S. Air Force, 2013; also Christian Hagen and Jeff Sorensen. Delivering Military Systems Affordably, Innovations in Systems and Software Engineering, vol. 7 issue 3, 2011, p. 161-170 <http://www.dau.mil/pubscats/ATL%20Docs/Mar_Apr_2013/Hagen_Sorenson.pdf>
18. D. Ward and S. Helton. "Estimating Return on Investment for SAVI (a Model-Based Virtual Integration Process)," SAE International Journal of Aerospace, vol. 4, no. 2, pp. 934-943, 2011; Peter Feiler, Jörgen Hansson, Dionisio de Niz, and Lutz Wrage. System Architecture Virtual Integration: An Industrial Case Study, Pittsburgh, PA: Carnegie Mellon University Software engineering Institute, Technical Report CMU/SEI-2009-TR-017, 2009. <http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=9145>; and Peter Feiler. An Incremental Life-Cycle Assurance Strategy for Critical System Certification, Proceedings and Presentations of the TSP Symposium, November 2014. <http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=423668>
19. U.S. Air Force Scientific Advisory Board. Sustaining Air Force Aging Aircraft into the 21st Century (SAB-TR-11-01). U.S. Air Force, 2011. <http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA562696>.
20. Joint Publication 3-12 (R). Cyberspace Operations. 5 February 2013. <http://fas.org/irp/doddir/dod/jp3_12r.pdf>
21. Defense Science Board. Mission Impact of Foreign Influence on DoD Software. Washington, DC: Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics, September 2007. <http://www.acq.osd.mil/dsb/reports/ADA486949.pdf>
22. Nancy Mead. SQUARE Process. Pittsburgh, PA: Carnegie Mellon University Software engineering Institute, 2006.
23. MITRE Corporation. 2011 CWE/SANS Top 25 Most Dangerous Software Errors. <http://cwe.mitre.org/top25/>
24. CERT. Secure Coding. <http://www.cert.org/secure-coding/>
25. Robert J. Ellison, Christopher Alberts, Rita Creel, Audrey Dorofee, and Carol Woody. Software Supply Chain Risk Management: From Products to Systems of Systems (CMU/SEI-2020-TN-026). Pittsburgh, PA: Carnegie Mellon University Software Engineering Institute, 2010.
26. Jeremy Kirk. "Pre-installed malware turns up on new phones." PC World, March 4, 2014. <http://www.pcworld.com/article/2104760/preinstalled-malware-turns-up-on-new-phones.html>
27. The Editorial Board of the NY Times. Edward Snowden, Whistle-Blower. January 1, 2014. <http://www.nytimes.com/2014/01/02/opinion/edward-snowden-whistle-blower.html?_r=0> and Marcelo Ballye and John Heggsteuen. Target's Data Breach Began with a Company that Does Heating and A/C Work on Its Stores. Business Insider. February 7, 2014. <http://www.businessinsider.com/target-data-breach-explained-2014-2>

Homeland Security

The Department of Homeland Security, Office of Cybersecurity and Communications (CS&C) is responsible for enhancing the security, resiliency, and reliability of the Nation's cyber and communications infrastructure and actively engages the public and private sectors as well as international partners to prepare for, prevent, and respond to catastrophic incidents that could degrade or overwhelm these strategic assets. CS&C seeks dynamic individuals to fill critical positions in:

- Cyber Incident Response
- Cyber Risk and Strategic Analysis
- Networks and Systems Engineering
- Computer & Electronic Engineering
- Digital Forensics
- Telecommunications Assurance
- Program Management and Analysis
- Vulnerability Detection and Assessment

To learn more about the DHS, Office of Cybersecurity and Communications, go to www.dhs.gov/cybercareers. To apply for a vacant position please go to www.usajobs.gov or visit us at www.DHS.gov.