Addressing Software Sustainment Challenges for the DoD

Michael McLendon, SEI Carnegie Mellon University
Bill Scherlis, SEI Carnegie Mellon University
Douglas C. Schmidt, SEI Carnegie Mellon University

Introduction

Software is essential to the DoD. It delivers enhanced capability to warfighters and provides competitive performance advantage across the full spectrum of DoD systems. These systems range from business information systems to complex C4ISR systems to major defense weapon systems and cyber capabilities [1]. To attain and maintain this advantage, it is imperative—and increasingly urgent—to create and execute an enterprise strategy for software innovation, development, and evolution that enhances affordability and continually optimizes warfighter effectiveness.

This enterprise DoD strategy must recognize the extent to which :

- Mission effectiveness depends on the ability of software developers and teams to deliver capability affordably and support the continual adaptation and enhancement of that capability
- Great value is provided to warfighters by enabling software-intensive functionality across the lifecycle so systems can operate interdependently and dependably in net-centric and cyber environments

It is hard to achieve these goals, however, due to rapid changes in mission environments and technology infrastructure, along with a challenging fiscal environment.

As DoD systems continue to age [2]—and sequestration and other budget constraints and uncertainties place greater emphasis on efficiency and productivity in defense spending [3]—it is increasingly important to create more efficient and effective approaches to sustaining and advancing the competitive edge that software provides. Software sustainment involves coordinating the processes, procedures, people, information, and databases required to support, maintain, and operate software-reliant aspects of DoD systems [4]. This article summarizes key software sustainment challenges faced by the DoD and highlights key R&D activities needed to address these challenges.

Software Sustainment Trends and Challenges

The software acquisition process delivers operational performance to meet identified warfighter requirements. Henceforth, systems transition into the sustainment phase. During sustainment, software-engineering processes and practices are continuously applied to (1) assure the ongoing competitive military advantage of a system and (2) ensure its seamless operation in helping to evolve net-centric and cyber infrastructures and environments. Various trends shape DoD policies and infrastructure for sustaining software, including:

- rapid performance advances associated with Moore's Law and associated hardware innovations (cost and capacity for storage, processing, and communications, and the consequent influence on computing systems architectures) that accelerate technology refresh cycles,
- the ever-increasing connectedness of systems, in which each system becomes a node in a vast, complex information network,
- the prevalence of closed-source and open-source off-the-shelf software technologies and practices, which commoditizes the market for software engineers with modern skills but creates gaps for projects that need staff with expertise in older technologies,
- the need to adapt software to address diminishing manufacturing sources stemming from the loss of producers or suppliers of hardware used in DoD systems,
- the challenges of modernizing and recapitalizing legacy DoD systems in a constrained budget environment that increasingly emphasizes greater efficiency and productivity in defense spending,
- the repurposing of systems to meet new threats, mission requirements, and coalition configurations, and
- the increasing requirements for interoperability in net-centric environments.

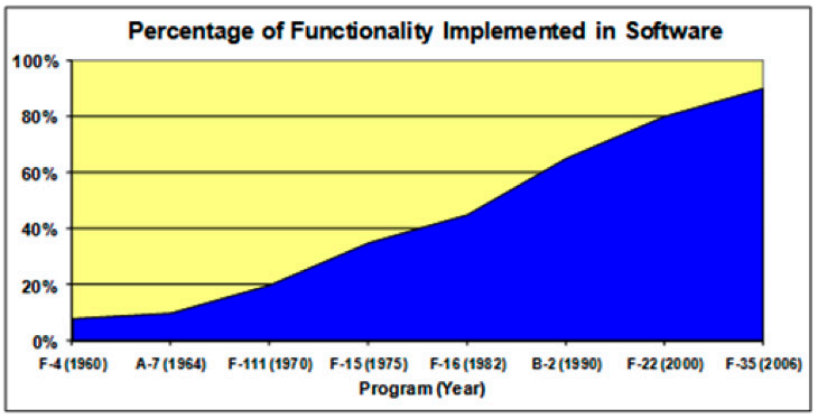

Figure 1: The Increasing Role of Software in Avionics Systems

The confluence of these and other trends impact the spectrum of acquisition and sustainment policies, programs, and infrastructure. These trends also exacerbate the growth in total ownership costs across program lifecycles [6].

Unfortunately, DoD acquisition programs have traditionally discounted design and program planning considerations for system sustainment until late in the acquisition phase (if at all). This attitude stems partly from the difficulty involved in measuring "sustainability attributes" in early phases of design and implementation. This difficulty, in turn, impedes a style of evolutionary enhancement during sustainment, where increments of investment in a system yield increments of immediate value in enhanced functionality, improved performance, etc.

Increasingly, however, the costs of software sustainment are becoming too high to discount for several reasons:

- Sustainment costs account for 60 to 90 percent of the total software lifecycle effort [5], which motivates the need to address sustainment throughout acquisition program lifecycles and improve the ability to measure—and ultimately reward—design and quality attributes applied during development that favor sustainability.
- In an era where DoD new-start programs are being reduced in favor of prolonging legacy systems, significant software sustainment cost increases are themselves unsustainable [6].

The growing expense of legacy systems—and their prolonged use—necessitate greater discipline, a sense of urgency, and attention to methods and technologies designed to improve sustainment.

To meet these challenges, software sustainment organizations must have a resilient and properly resourced infrastructure that integrates processes, practices, and people with evolving competencies, tools, information, databases, and system-integration lab capabilities. These infrastructure elements, in turn, must be systematically refreshed throughout the life of a system to support, maintain, and operate in accordance with unique properties of software in DoD systems.

For example, software does not follow the laws of physics that bound hardware design and define failure [1]. Legislative and DoD policies, however, have historically mandated a depot-centric maintenance paradigm based on relatively discrete hardware aging/replacement models. Unfortunately, these models are not well-suited to understand the cost, effort, and quality drivers of software sustainment, which is a continuing software engineering process that lasts for decades.

The Impact of Supply Chains on Software Infrastructure and Sustainment

Compared to legacy systems, newer DoD systems tend to rely more on software as a primary means to deliver functionality [1]. There are good reasons for this trend, which has rapidly accelerated over the past decade in both national security systems and commercial systems. In particular, the increasing use of—and dependency on—software means there are fewer limits on what capabilities can be enhanced and created in the future. For example, the percentage of avionics specification requirements that rely on software has risen from approximately 8 percent of the F-4 in 1960 to 45 percent of the F-16 in 1982, 80 percent of the F-22 in 2000, and 90 percent of the F-35 in 2006 [7], as shown in Figure 1.

Software is ubiquitous in DoD systems, and it is increasingly hard to identify sub-systems and components that are not controlled or enabled by software. Ironically, in this increasingly software-reliant environment, there is a growing bow wave of software sustainment demands (of unknown size, complexity, characteristics, and technical debt) that are neither recognized nor understood by the acquisition community and the DoD enterprise.

For example, not only are we dealing with a growing software base, but also the constantly evolving infrastructure in which software runs. This infrastructure includes commercial and open-source components, frameworks, and libraries, all of which are increasingly necessary for modern software systems. Moreover, there is increasing reliance on software supply chains that provide and support this infrastructure.

For example, there are supply chains for hardware/firmware components, as well as integrated components, such as network routers, operating systems, databases, and middleware [16]. A supply chain for COTS software products includes product development organizations and their suppliers. Likewise, the supply chains for custom-developed DoD acquisition systems can include the prime contractors, subcontractors, and supply chains for the COTS products used.

Software infrastructure typically evolves at a rapid pace, driven by opportunities to increase capability, improve performance, provide repairs and security enhancements, and exploit growth in underlying hardware capability. This upgraded capability must be integrated into existing systems. Likewise, software defects and performance bottlenecks must continually be identified, fixed, and optimized to provide full functionality.

Infrastructure also evolves due to improvements in its own underlying infrastructure, (i.e., lower layers of the software/hardware stack). A common example involves improvements in underlying operating systems, cloud architectures, and storage and processing capabilities that enable improvements to a database framework. An important consequence of this—and a principal driver of component-based and service-oriented software paradigms—is the speed and efficiency with which new capabilities can be manifested. For example, talented undergraduate students can apply modern software and hardware infrastructure in a matter of weeks to create highly capable mobile software apps that access dedicated cloud resources and can be widely deployed and supported.

The increasing reliance of DoD systems on software supply chains extends well beyond the defense industrial base. This trend is the subject of a 2007 Defense Science Board report [8] regarding the challenges of testing and evaluating these supply chains. Although software does not wear out, firmware and hardware become obsolete rapidly, thereby driving changes in software applications and infrastructure.

In mainstream commercial systems, these changes are planned for and provide end users a steady flow of improvements in performance and reliability derived from the underlying infrastructure. Just as importantly, these changes create headroom for improvements in function and capability.

The Relationship of Software Sustainment to Modernization Efforts

The majority of software sustainment activity is better described and managed as a modernization effort. This shift in perspective is consistent with commercial development practices and shifts in the business environment for defense systems [10]. The technical drivers discussed below—along with the ongoing rapid growth in capability of software infrastructure discussed above—have also enabled this move toward modernization.

In general, software sustainment involves the following pattern of repair, enhancement, and adaptation:

- Repair in response to defects and vulnerabilities related to functional, quality, and security attributes.
- Enhancement in response to demands for increased functional capability and performance, driven by competitive pressure (in the commercial world) and changes in mission profile (in defense).
- Adaptation in response to improvements, changes, new opportunities in the underlying stack of software and hardware infrastructure, and the mission benefits of increased interoperability among software-reliant systems in the enterprise.

This pattern is pervasive in commercial software. In recent years, this software sustainment pattern—and the tempo at which that pattern has been applied—has been amplified because many applications and data repositories have migrated to cloud-based systems [9]. This transformation is evident across the spectrum, from mobile apps (which tend to rely on cloud-based resources) to large-scale data-intensive applications.

The sustainment community has shifted from primarily emphasizing repair to focusing on enhancement and adaptation [6]. This shift stems from various mission and business considerations, not the least of which is the reduced deployment of new systems in favor of sustaining legacy systems. It is also a result of the DoD's growing ability to manifest increasing levels of functionality in software, which in turn is a consequence of the rapid pace of innovation in tools, languages, models, and processes.

Indeed, cloud-based software applications may have a much greater tempo in their update cycle. The term "DevOps" arose in the context of commercial systems to refer to the rapid iteration of development, quality assurance, and operations. This iteration is most evident in cloud-based applications due to the relative ease—and transparency—of deployment, especially when quality practices are integrated into development efforts.

Understanding and Mitigating the Cost Drivers of Software Sustainment

To craft a more effective and efficient approach to software sustainment, organizations must examine and understand the complexities and costs of the software infrastructure environment. This complex nexus of activities has historically been neglected. Recent studies [2][6], however, indicate that the DoD is expending more time and effort sustaining software, often more than originally anticipated due to uncertainties encountered during initial program cost estimation.

For example, a 2011 Air Force Scientific Advisory Board study [6] showed that total weapon system software sustainment costs have doubled in less than 10 years, as shown in Figure 2. Likewise, software sustainment hours at the three Air Logistics Centers over the past eight years have also increased significantly.

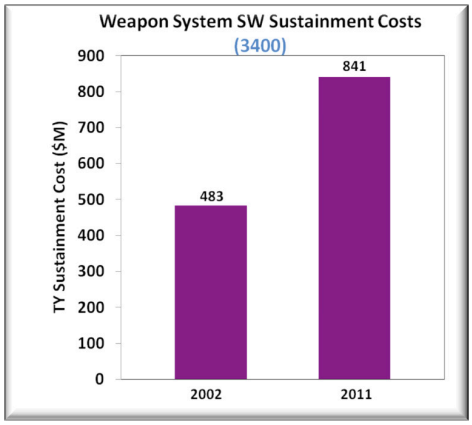

Figure 2: Increase in Software Sustainment Costs Over the Past Decade

On a larger scale, the Office of the Deputy Assistant Secretary of the Army for Cost and Economics—in collaboration with the Air Force and the Navy—is sponsoring critical and foundational research into understanding the myriad of activities that occur in what the DoD calls "software depot maintenance." SEI at Carnegie Mellon University has also initiated research [14] that addresses the uncertainty of cost estimates early in the lifecycle and the dynamics of decision making associated with choices about sustainment strategies.

Various factors contribute to the high costs of software sustainment. For example, functionality (such as fly-by-wire) originally provided by hardware may be replaced by software, which must then be sustained. Periodic software upgrades and enhancements throughout the lifecycle of DoD systems may also result in unanticipated increases in sustainment costs. Moreover, software maintainers must expend costly and time-consuming effort to understand original designs and carefully make changes to avoid degrading design integrity or negatively impacting key quality attributes. In addition, the scale and complexity of software are growing significantly to meet the expanded threat spectrum [11], which exacerbates sustainment costs.

As sustainment costs have increased, the DoD has struggled to support all its legacy systems—especially its weapon systems platforms—many of which will remain in the operational inventory much longer that planned due to budget constraints. Examples

of weapon systems platforms include the physical airframes, hulls, chassis, and their associated parts such as engines, weapons, sensors, and computing/communication units. Economic strategies for understanding and addressing these rising costs are affected by a key difference between the software running in DoD weapon system platforms and the platforms themselves.

Sustainment costs have historically been attributed to the following factors:

- **the number of systems in the operational inventory,**
- **the operating tempo (optempo) of systems (flying hours, driving miles, number of deployments, etc.),**
- **the number of different configurations,**
- **parts count, and**
- **failure rates.**

The wear and tear on hardware at the platform-, sub-system-, and component-levels represents a significant maintenance expense. Through the years, the DoD has developed a finely tuned set of heuristics for estimating field and depot maintenance costs, budgets, and the relationships of maintenance funding and backlogs to operational readiness. In the face of declining budgets, the DoD has traditionally handled these costs by shrinking its force structure inventory and the operational tempo of forces, (e.g., by retiring and/or reducing the numbers of aging aircraft, ship, and vehicle platforms) [6].

This approach worked when sustainment costs were largely a function of the hardware for weapon system platforms. In contrast, software has essentially no expenses related to manufacturing or wear-and-tear. As a result, software sustainment costs are insensitive to the traditional hardware-maintenance cost drivers. In fact, software sustainment costs are primarily driven by the function a system or sub-system exists to perform, the multiple configurations of systems in the inventory (each with their own software variant), and the increasing degree of interoperability among systems in net-centric environments.

For example, a class of ships, planes, or vehicles may have scores of software variants reflecting different sensor, processing hardware, operating system, and network/bus configurations; different algorithms; and different security profiles for customers from different countries. Sustaining all these variants affects the time and effort required to assure, optimize, and manage the system throughout the lifecycle. These factors then inform the size, configuration, and capabilities required of the software sustainment infrastructure.

A critical workforce challenge is the need to reconsider current legislative and policy mandates concerning the organic and contractor share of sustainment across the DoD enterprise [6]. The pace of technological change—coupled with the continuous need to deliver greater performance to the warfighter at an affordable level of investment—creates significant pressure to assess, at the DoD-enterprise level, how to plan, organize, and perform software sustainment. This assessment should create more effective, efficient, and continually refreshed software-sustainment strategies and organizations, and the alignment of those organizations around portfolio and product-lines.

The Importance of Architecture in Enabling Effective and Efficient Sustainment

Software variability inevitably grows in legacy systems unless a concerted effort is made to rein it in. Unchecked, it becomes increasingly hard to avoid adding unnecessary variability, re-implementing variation mechanisms more than once, selecting incompatible or awkward variation mechanisms, and missing required variations. This bloat can be overcome through explicit attention to architectural features and encapsulation of the various separate dimensions of variability [12], which is a principal feature of software architecture [13].

In modern software-reliant systems, the concept of architecture includes commitments regarding the structure and content of the interactions among system components [1]. Structural commitments generally focus on which components can interact and how information exchanged between components is represented, scaled, and transmitted via data models and protocols. Other commitments may include critical quality attributes, such as performance and availability expectations, security considerations, usability, and so on.

In short, architecture is the set of critical design commitments that regulate what may and may not happen within an overall system [12]. There are two key perspectives on architecture that are essential for effective and efficient software sustainment:

- From a management perspective, architecture embodies anticipation of change: in the rapidly evolving technology infrastructure, in capabilities that will be delivered to users over a period of 5 to ten years, and in policy and business rules. Interoperability problems are evidence of missing or inadequate architectural planning, often compounded by misaligned incentives among development teams or contractors.
- From a technical perspective, architecture provides a framework for coordinating data exchange within an enterprise and for systematically addressing quality attributes. Good architectural designs anticipate change by encapsulating variability to reduce cost and risk. In this approach, change-prone areas (such as hardware and communication infrastructures) are accessed via stable interfaces whose implementations can be replaced without undue side-effects on other software components. Many software patterns [13][15] exist entirely for this purpose.

Architectural decisions thus regulate the overall interplay among systems within an enterprise. In many enterprises, "architecture" may be the result of incremental decisions over time, where a sequence of local decisions determines overall organizational outcomes, for better or worse.

Failure to attend to architecture often leads to the loss of intellectual and configuration control that is manifested via terms such as "software rot" or "bloatware." In the absence of an architecture-centered approach, the DoD will face "sticker shock" because software sustainment costs are unlikely to decrease by shrinking inventory alone. For example, since the cost drivers for software sustainment relate more to the (combinatorial) number of configurations and variants, approximately the same level of effort is needed regardless of whether there are 100 or 10,000 hardware platforms.

To address these issues, the DoD needs different strategies for understanding and alleviating rising software sustainment costs by considering architecture-based approaches early

in the system-acquisition process. Architecture must therefore be an explicit consideration in the systems engineering trade-off process in advanced development planning and the technology development phase of the acquisition process. In particular, sustainment strategies based on managing software commonality and variability via software product lines should be considered when conducting systems engineering trade-off analyses [12].

Workforce Challenges Associated with Software Sustainment

In addition to the technical and economic challenges discussed above, the DoD faces challenges with recruiting, training, and retaining an efficient, productive, and continually refreshed workforce of engineers and technical managers to meet its sustainment needs [1][6]. Effective software sustainment requires this workforce to have expertise in older programming languages, operating platforms, and tools. It must also have deep domain knowledge, software architecture knowledge, and a full appreciation of the emerging software technologies that will form the basis of reengineered systems. More experienced members of the DoD workforce tend to possess this expertise, so retaining and replenishing this critical human resource is essential.

In general, the DoD's software sustainment activities rely on a combination of in-house expertise (so-called "organic sustainment") and external capability (accessed through contracting, consultancy, or advisory panels). A base of capable in-house expertise is essential in any technology-intensive organization, even those that outsource the bulk of actual technical work. In-house experts help ensure an organization is a smart customer on development projects. For example, these experts can identify needs and opportunities, create and manage relationships, structure incentives, evaluate risks and costs, and otherwise assure that the external (and internal) relationships are technically sound and aligned with organizational interests.

In-house expertise is particularly essential for DoD programs, program offices, and services to address architectural sustainment issues that transcend individual systems, development activities, and acquisition programs. These broader issues involve how separately managed, contracted development efforts might interact. While external advice can (and should) be sought and followed, it is necessary—from the standpoint of vision, strategy, and accountability—that the core technical leadership come from within the organization [1][8].

For in-house sustainment activity, a high-quality technical workforce is essential to support rapid, informed, and agile responses to evolving mission requirements, operational needs, and changes in technology infrastructure. Fewer barriers exist for in-house teams to engage in modern iterative and incremental development practices to support rapid evolution. Unfortunately, although some in-house organizations [5] are dedicated to sustaining software, their efforts are often not as well recognized (or funded) by the DoD, especially in the face of an aging DoD inventory [2].

The DoD must also address other critical deficiencies to achieve and sustain a high-quality workforce. For example, software acquisition management and software engineering are not DoD career fields, even though expertise in these domains has proven critical to success. There is thus an urgent need to address critical and emerging workforce challenges stemming from current legislative and policy mandates concerning the organic and contractor share of sustainment across the DoD enterprise.

The rapid pace of technological change, coupled with the ever-increasing need to deliver greater performance to the warfighter at an affordable level of investment, creates significant pressure to objectively assess at the DoD enterprise level how to plan, organize, and perform software sustainment. This assessment should seek to create more effective, efficient, and continually refreshed software sustainment strategies, organizations, and alignment of those organizations around portfolio and product-lines.

Key R&D Activities Needed to Address Software Sustainment Challenges

The software research community has devised various approaches to improve software sustainability. For example, tools for detecting software modularity violations help identify eroding design structures (referred to whimsically as "bad code smells" by software developers and managers) so they can be refactored. Likewise, intelligent automated regression testing frameworks help ensure that changes to legacy software work as required and that unchanged parts have not become less dependable.

Over the past several decades, the SEI has created methods and guidelines for sustaining, migrating, and evolving legacy systems. For example, the SEI has devised strategies for modernizing legacy systems and reusing legacy components. These strategies employ risk-managed, incremental approaches that encompass changes in software technologies, engineering processes, and business practices. In addition, the SEI has created techniques for measuring the effectiveness of software-sustainment practices. These techniques can help decision-makers select between (1) continued sustainment versus replacement or (2) which of the multiple (redundant) legacy systems to keep and which to retire.

Conclusion

Despite its strategic importance to the DoD, software sustainment has received relatively little visibility and emphasis as an enterprise policy, program, and resource issue. The fact that our legacy weapon systems provide competitive advantage to the warfighter is due to the dedication and skills of the software sustainment workforce, both government and contractors, located at the services' software depot centers and at contractor facilities. We contend, however, that a greater sense of urgency is required to ensure DoD's sustainment capabilities can continue to deliver warfighter capability in the face of significant fiscal, technology, and workforce challenges [3].

This article just scratches the surface of the complex landscape of policy, program, people, and technical design and infrastructure challenges associated with sustaining software-reliant DoD systems. Other vexing, non-technical challenges affecting sustainment and total ownership costs are that DoD contracts often fail to procure source code, necessary licenses, and technical data rights, as well as technical data on design artifacts, testing facilities, and procedures during the acquisition process [10]. The DoD needs to adopt a holistic approach to software sustainment that addresses the technical, management, and business perspectives in a balanced manner.

ABOUT THE AUTHORS

Mr. McLendon currently serves as the Associate Director, Software Solutions Division for the Software Engineering Institute, Carnegie Mellon University. Prior to assuming this position, Mr. McLendon served as Senior Advisor in the Office of the Assistant Secretary of Defense for Systems Engineering. He also served as a principal in the Office of the Assistant Secretary of Defense for Program Analysis and Evaluation and in the Office of the Under Secretary of Defense for Policy. He later was a Professor at the Defense Systems Management College. He served as a career Air Force officer in a range of leadership and management positions in system and technology development and acquisition as well as the federal level and the private sector.

SEI Carnegie Mellon University
4500 Fifth Ave
Pittsburgh, PA 15213
E-mail: mmclendon@sei.cmu.edu

Dr. William L Scherlis is a Professor in the School of Computer Science at Carnegie Mellon. He is director of CMU's Institute for Software Research (ISR) in the School of Computer Science and the founding director of CMU's PhD Program in Software Engineering. From Jan 2012 to January 2013 he served as the Acting CTO for the Software Engineering Institute. His research relates to software assurance, software analysis, and assured safe concurrency. Dr. Scherlis chaired the National Research Council (NRC) study committee on defense software producibility, which released its final report Critical Code: Software Producibility for Defense in 2010. He is a Fellow of the IEEE and a lifetime National Associate of the National Academy of Sciences. Dr. Scherlis joined the Carnegie Mellon faculty after completing a Ph.D. in Computer Science at Stanford University, a year at the University of Edinburgh (Scotland) as a John Knox Fellow, and an A.B. at Harvard University.

SEI Carnegie Mellon University
4500 Fifth Ave
Pittsburgh, PA 15213
E-mail: scherlis@sei.cmu.edu

Dr. Douglas C. Schmidt is a Professor of Computer Science and Associate Chair of the Computer Science and Engineering program at Vanderbilt University. He is also a Visiting Scientist at the Software Engineering Institute, where he served as the CTO from September 2010 to December 2011. Dr. Schmidt has published 10 books and more than 500 technical papers on software-related topics, including patterns, optimization techniques, and empirical analyses of object-oriented frameworks and domain-specific modeling environments that facilitate the development of distributed real-time and embedded middleware and mission-critical applications running over data networks and embedded system interconnects. Dr. Schmidt received B.S. and M.A. degrees in Sociology from the College of William and Mary in Williamsburg, Virginia, and an M.S. and a Ph.D. in Computer Science from the University of California, Irvine (UCI).

SEI Carnegie Mellon University
4500 Fifth Ave
Pittsburgh, PA 15213
E-mail: dschmidt@sei.cmu.edu

REFERENCES

1. National Research Council's Critical Code: Software Producibility for Defense report, <http://www.nap.edu/openbook.php?record_id=12979&page=R1>
2. National Research Council's, Aging of U.S. Air Force Aircraft report <http://www.nap.edu/catalog.php?record_id=5917>
3. Ashton Carter, "Better Buying Power: Guidance for Obtaining Greater Efficiency and Productivity in Defense Spending," Memorandum for Acquisition Professionals, September 14, 2010.
4. Mary Ann Lapham, "Sustaining Software-Intensive Systems," SEI technical report, <http://www.sei.cmu.edu/library/abstracts/reports/06tn007.cfm>
5. United States Air Force Software Technology Support Center, "Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems, Command and Control Systems, and Management Information Systems (Condensed Version 4.0).," Ogden Air Logistics Center Hill AFB, UT, February 2003.
6. Air Force Science Advisory Board's Sustaining Aging Aircraft report, <http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA562696>.
7. Report from the Defense Science Board Task Force on Defense Software, November 2000.
8. Report of the Defense Science Board Task Force on Mission impact of Foreign Influence on DoD Software, September 2007.
9. Teri Takai, et al., "Department of Defense Cloud Computing Strategy," July 12, 2012.
10. Nick Guertin and Brian Womble, "Competition and the DoD Marketplace," Proceedings of the Ninth Annual Acquisition Research Symposium, April 30th, 2012.
11. Lind Northrop, et al., Ultra-Large-Scale Systems: The Software Challenge of the Future, Software Engineering Institute, 2006.
12. Paul Clements and Linda Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley, 2001.
13. Frank Buschmann, et al., Pattern-Oriented Software Architecture: A System of Patterns, Wiley, 1996.
14. Robert Ferguson, "An Investment Model for Software Sustainment," SEI Blog, July 22, 2013.
15. Erich Gamma et al., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
16. 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-2010-TN-026.