An Agile Systems Engineering Process The Missing Link?

Matthew R. Kennedy, DAU
David A. Umphress, Ph.D., Auburn University

Abstract: Today's systems are increasingly threatened by unanticipated change arising from volatility in user requirements, Information Technology (IT) refresh rates, and responses to security vulnerabilities. With the rapidly changing world of IT, long static development cycles of a Software Intensive System (SIS), a system in which software represents the largest segment in one or more of the following criteria: system development cost, system development risk, system functionality, or development time [1] may doom the system before development begins.

Delivering a SIS that is on time, within budget, and on schedule has been shown to be problematic [2]. This problem will only increase as the complexity of SISs within the DoD grows and more functionality within systems is relegated to software [3] [4].

Traditional systems engineering portrays systems development as a top-down, waterfall-centric process, one that relies on explicating requirements as early as possible. Such a perspective tends to postpone modifications until the maintenance phase [5], thus thwarting early insertion of technology or a nimble response to changes in user needs. Though the technology refresh rate varies from system to system, a report from the state of Michigan shows the following industry computer technology refresh trends:

1. 40% of companies are on a four-year cycle for refreshing personal computers (hardware), and
2. Microsoft plans a two-year cycle to release a new operating system (software) [6].

A report from the U.S. Army War College estimates that commercial electronics have a typical refresh rate of 12-18 months but may be less [7].

Cyber security further complicates the picture. The rate at which vulnerabilities are identified in a system cannot be predicted. According to the National Vulnerabilities Database, between 2000 and 2009 there was an average of 3,825 vulnerabilities reported each year due to software flaws alone [8]. The need for a responsive systems engineering process to rapidly address unforeseen vulnerabilities is imperative for the development of a secure system.

The Office of the Assistant Secretary of Defense for Network Information Integration conducted an analysis of 32 major information system acquisitions and found the average time to deliver the Initial Operating Capability was 91 months [3]. With the DoD's history of long delivery cycles and the short time required for technology refresh, the systems engineering process needs to be responsive to changes introduced both by the user and technology.

This inability to respond rapidly to change is nothing new. Software engineering recognized the pitfalls of a strictly sequential development process a number of years ago. The contemporary school of thought in software engineering has evolved away from considering a waterfall approach as the primary sequence of development activities and toward approaches that embrace change by segmenting software development into manageable change-resistant increments and allowing change to take place at increment boundaries [5]. Ultra-modern approaches—known as agile processes—have emerged to match the pace in which change is encountered during software development. Agility is "the speed of operations within an organization and speed in responding to customers (reduced cycle times)" (Massachusetts Institute of Technology). The degree of agility when developing an IT system is the organization's ability to respond to changing requirements and technology. With the quick technology refresh rate, long development cycles could place a system in a state of obsolescence prior to initial release. With the ever-changing world of technology, the need to change without notice throughout the development lifecycle is paramount to success.

Just as the software community has moved toward a more agile approach to become more responsive to changes throughout the development lifecycle, the systems engineering community needs to follow a similar approach to remain competitive in today's rapidly changing environment.

Past Performance

Failure to deliver a successful SIS can rarely be attributed to one project deficiency; however, the inability to rapidly adapt to change appears to be an underlying theme in many SIS development failures. A successful SIS is defined as a system that is on time, within budget, and contains all of the required features and functions [9]. Instead of steadily making improvements on the successful delivery of SISs, the Standish Group 2009 Chaos report showed a "marked decrease in project success rates," in which only 32% of projects were successfully delivered when compared to the 35% reported in their 2006 report [9] [10].

The U.S. Government Accountability Office (GAO), an "independent, nonpartisan agency that works for Congress," investigates how the government spends taxpayers' dollars [11]. The Air Force is developing an F-22 aircraft that is intended to provide increased capabilities over current aircraft. A GAO report found the program has undergone several changes since the development began in 1986 and the Air Force cannot afford to purchase the quantities of the aircraft that were initially anticipated. This was partially attributed to the Air Force adding more robust air-to-ground attack requirements in 2002. In addition to the change in requirements, the Air Force has determined that a revised computing architecture, as well as new computer processors were needed to support planned enhancements, both of which further increased program costs [12]. Previous experience shows that changes within a SIS are inevitable, whether or not there is a change in requirements or technology. Though predicting these changes may be difficult, processes can be structured to be more responsive to these unanticipated changes. Increasing agility within the systems engineering process is one mechanism that may result in increasing the successful delivery of a SIS.

Growth of SISs

The software within today's systems is only increasing. Examining the correlation between the Executable Software Lines of Code (ESLOC) and time in various DoD systems (Figure 1) shows a steady increase in ESLOC in related systems over time. The Aegis system introduced in the early 1980s had less than 2 million ESLOC. The Virginia SSN introduced roughly 20 years later contained over double the ESLOC and the estimated ESLOC for the DDX system is just under 10 million.

The increase in ESLOC means that more of the system's functionality is being performed by software. Functions performed by software in DoD aircraft (Figure 2) has increased from 8% for the F-4 Phantom II in 1960 to 80% for the F-22 Raptor in 2000. With the proliferation of software within current systems, problems that were inherently software are evolving into system problems [4].

DoD systems are not the only systems experiencing an increase in software; the automotive industry has also seen an increase. In 1977 the Oldsmobile Toronado contained the first productive microcomputer Electronic Control Unit used for only electronic spark timing [13]. Just a year later, the Cadillac Seville offered on its *Cadillac Trip Computer* a software-driven display of speed, fuel, trip, and engine information [13]. By 1981, GM was using microprocessor-based engine controls executing roughly 50,000 Software Lines of Code (SLOC); today it is estimated that a premium automobile takes dozens of microprocessors running 100 million SLOC [13].

When determining the impact of software on overall system cost, Broy notes that "the cost of software and electronics can

Figure 1: Increase in Software in DoD Systems

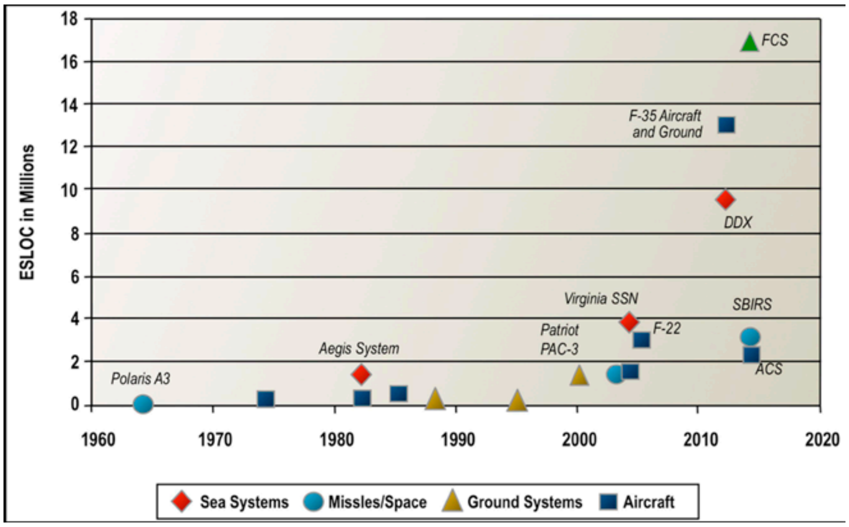
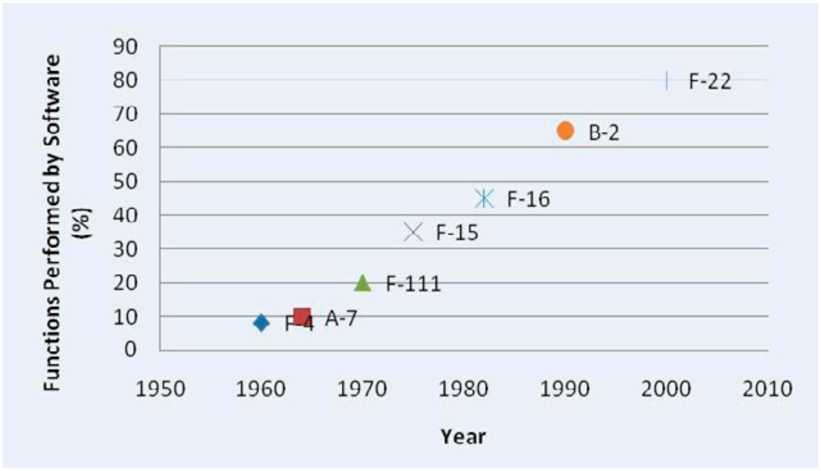


Figure 2: Functions Performed by Software (Nelson and Clark)



reach 35% to 40% of the cost of a car [13]. A study conducted by the Center for Automotive Research had similar findings [14] stating, "Software made up only 16% of a vehicle's total value in 1990, this figure had increased to 25% by 2001. By 2010, the share of a car's total value is expected to climb to almost 40%."

The inability to deliver a successful SIS will only be exacerbated as software continues to become an increased portion of a system's composition.

SIS Development

Development of a SIS can be envisioned as an amalgamation of three aspects: business, system, and software. Though there is some overlap among these aspects, general responsibilities can be attributed to each aspect.

The business aspect is responsible for the overall acquisition of the system including contracting, funding, operational requirements, and overall system delivery structure. The system aspect is responsible for the overall technical and technical management aspects of the system and serves as the interface

Figure 3: Std 1220-2005 Systems Engineering Process

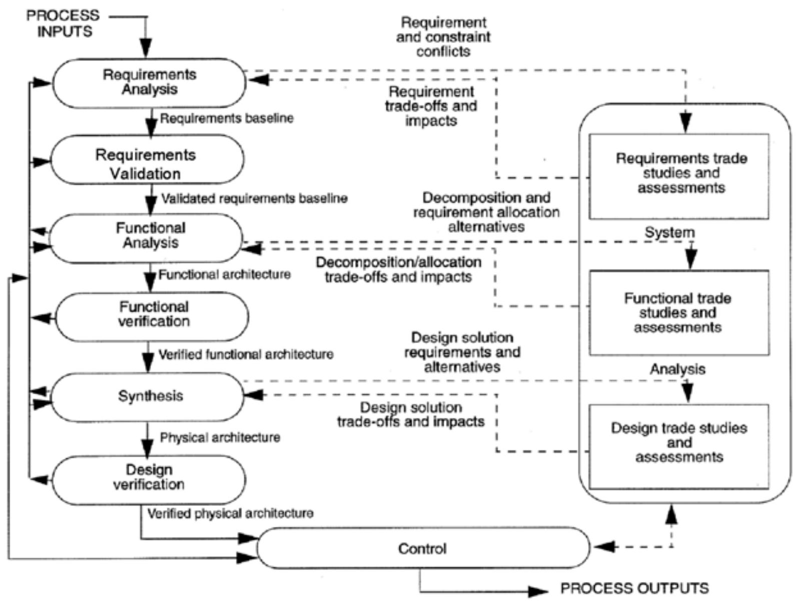
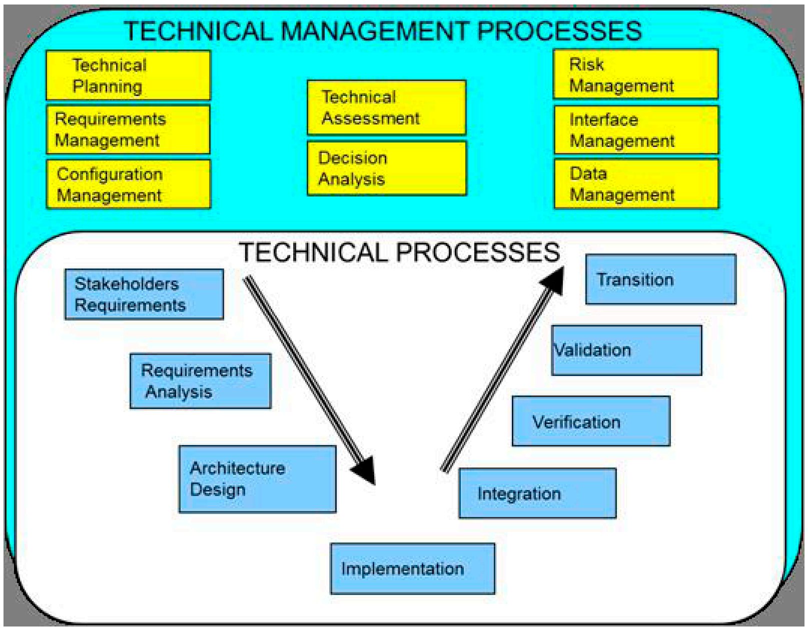

PROCESS INPUTS
Requirement and constraint conflicts
Requirements Analysis
Requirement trade-offs and impacts
Requirements baseline
Requirements Validation
Requirements trade studies and assessments
Validated requirements baseline
Decomposition and requirement allocation alternatives
Functional Analysis
System
Decomposition/allocation trade-offs and impacts
Functional architecture
Functional verification
Functional trade studies and assessments
Design solution requirements and alternatives
Verified functional architecture
Analysis
Synthesis
Design solution trade-offs and impacts
Physical architecture
Design trade studies and assessments
Design verification
Verified physical architecture
Control
PROCESS OUTPUTS

Figure 4: DAG Systems Engineering Processes (University, DAU Information Resource Management 202 Course)



between management and engineers. The software aspect is responsible for the software items contained in the SIS.

When developing a SIS, all three aspects need to work in harmony to produce a successful final product. Traditionally, when using a once-through development methodology, the business aspect would provide the funding and operational requirements to the system aspect. The system aspect would further decompose the requirements and allocate them to software or hardware. These items would then be developed and integrated resulting in a completed system. Given that major information systems average a 91-month gap from operational requirements definition to system delivery, defining requirements that far in advance of technology that is changing every 12 to 18 months suggests that the end result will not be an up-to-date system.

The need for increased agility has been identified within the business aspect and there are initiatives aimed at developing an agile framework within this aspect. Per the fiscal year 2010 National Defense Authorization Act, section 804, the U.S. Congress directed the Secretary of Defense to, "develop and implement a new acquisition process for IT systems" [15]. This new Defense Acquisition System process must include: Early and continual involvement of the user; multiple, rapidly executed increments or releases of capability; early, successive prototyping to support an evolutionary approach; and a modular, open-systems approach [15].

Moreover, this process should be based on the March 2009 report of the Defense Science Board (DSB) Task Force on Department of Defense Policies and Procedures for the Acquisition of Information Technology [15]. The DSB report concluded, "The conventional DOD acquisition process is too long and too cumbersome to fit the needs of the many IT systems that require continuous changes and upgrades" [3].

The report noted that an agile acquisition approach would increase IT capability and program predictability, reduce cost, and decrease cycle time.

In addition to the emerging Agile IT Acquisition Lifecycle, the DoD developed an agile requirements process for IT Systems called the "IT Box" [16]. The Joint Requirements Oversight Council Memorandum 008-08 stated, "IT programs are dynamic in nature and have, on average, produced improvements in performance every 12-18 months" [17]. Recognizing the need for performance improvements, the IT Box allows IT programs the flexibility to incorporate evolving technologies.

Lack of evidence implies the system aspect does not have similar agile initiatives. There are several systems engineering guides and standards available such as the Defense Acquisition Guidebook (DAG) Chapter 4, EIA-632, IEEE std 1220-2005, ISO/IEC 15288, and ISO/EIC 26702 [18,19,20,21,22]. In practice, no single systems engineering standard is used, but instead a combination of standards. For example, the Air Force produced Instruction 63-1201, Life Cycle Systems Engineering, which references numerous systems engineering standards and is to be used in the development of all AF systems [23]. These guides and standards provide the overall structure of the systems engineering process as well as identify characteristics required during the process.

IEEE Std 1220-2005 defines a systems engineering process (Figure 3) as, "a generic problem-solving process, which provides the mechanisms for identifying and evolving the product and process definitions of a system." It further notes that the SEP should be applied throughout the system lifecycle for development and further identifies the lifecycle stages (System definition stage, Preliminary design stage, Detailed design stage, Fabrication, assembly, integration, and test stage, Production and customer support stages). However, it does not detail how the SEP should be applied from an agile project management perspective.

In contrast to IEEE Std 1220-2005, the DAG, Chapter 4, divides the SEP into two categories: Technical Management Processes and Technical Processes [18]. At a high level, the generic Technical Processes frame the steps necessary to develop a system whereas the Technical Management Processes are used to manage the technical development (Figure 4).

In addition to further describing key activities in each process area, the DAG contains some systems engineering best practices such as employing a modular design and designating key interfaces [18].

Current systems engineering guides and standards provide a waterfall-like structure and key systems engineering characteristics that are imperative for successful system development. However, they do not provide a framework for planning and managing projects that allow systems engineers to rapidly respond to the changes. The design and implementation of such a framework is left to the systems engineers who are provided little guidance. The structure and characteristics provided need to remain intact while their application needs to be framed such that it allows for an agile implementation.

Similar to the system aspect, the software aspect has a number of standards available such as ISO 12207, ISO 9001 and the Capability Maturity Model Integrated (CMMI®) [24,25,26]. The CMMI was a collaborative effort by the U.S. government, industry and Carnegie Mellon [27] that contains a process improvement model consisting of best practices addressing activities throughout the products lifecycle [24].

ISO 12207 "contains processes, activities and tasks that are to be applied during the acquisition of a system that contains software" [26]. A limitation identified within ISO 12207 is that it does not specify details on how to implement the identified activities or tasks [26].

As with the system aspect, the software aspect guides and standards only provide the characteristics required; however, the software aspect has agile frameworks built on top of these standards, that allow software to be developed in an atmosphere where requirements are changing. One such agile framework is called Scrum. Scrum was formalized by Ken Schwaber at the Object-Oriented Programming, Systems, Languages and Applications conference in 1995 [28]. Since Scrum has been in existence for 15 years, it has a large collection of lessons learned, as well as success stories, which have contributed to its current state. These additional frameworks allow the Software Aspect increased agility during the development process.

Conclusion

The rapid technology refresh rate coupled with the need to respond to changing requirements requires a complete agile development process; one where the business, system, and software areas contain an agile framework and work in unison to create a successful SIS. A deficit in any of the three areas will cripple the overall process. The increase in software within today's systems only increases the need for an agile systems engineering process.

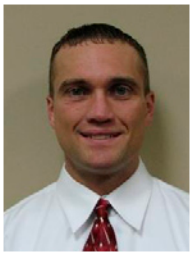
The emerging DoD Agile IT acquisition lifecycle and IT Box provide the foundation for the business area's transformation to agility. Currently, nothing is being done to address the lack of responsiveness within the system area. The system area provides the critical link between the business and software areas; as such, lack of agility in the system area can have a debilitating effect on the overall development process. This increases the risk of negating both the improvements being made in the business area and the existing agile processes in the software area.

The development of an agile system engineering framework is required to enhance the overall effectiveness of the SIS development process. Key interfaces also need to be identified from the system area to the business and software areas enabling seamless communication between adjacent areas.

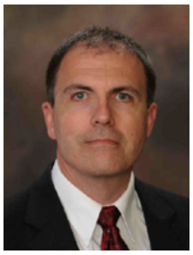
Disclaimer:

®CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

ABOUT THE AUTHOR



Matthew R. Kennedy is a professor of software engineering at DAU. He served in the U.S. Air Force as a network intelligence analyst and he has more than 10 years of experience in IT. He has a bachelor's and master's degree in computer science.



David A. Umphress, Ph.D., is an associate professor of computer science and software engineering at Auburn University, where he specializes in software development processes. He has worked over the past 30 years in various software and system engineering capacities in military, industry, and academia settings. He is an IEEE certified software development professional.

REFERENCES

1. Defense Acquisition University. "Glossary." 2009.
2. The Standish Group. CHAOS Summary 2009. Boston, 2009.
3. Force, Defense Science Board Task. Department of Defense Policies and Procedures for the Acquisition of Information Technology. Washington: Office of the Under Secretary of Defense, 2009.
4. Ferguson, Jack. "Crouching Dragon, Hidden Software: Software in DoD Weapon Systems." IEEE Software (2001): 105-107.
5. FORCE, DEPARTMENT OF THE AIR. "Guidelines for Successful Acquisition and Management of Software-Intensive Systems." 2003.
6. Information Technology Equipment Life-cycle. Michigan, 2004.
7. Daniels, Jody. Review of Acquisition for Transformation, Modernization, and Recapitalization. Carlisle: U.S. Army War College,Carlisle Barracks, 2006.
8. Statistics. 23 10 2010. 23 10 2010 <http://web.nvd.nist.gov/view/vuln/statistics-results?cid=4>.
9. The Standish Group. CHAOS Summary 2009. Boston, 2009.
10. Dominguez, Jorge. "The Curious Case of the CHAOS report 2009." 2009.
11. About GAO. 11 09 2010 <http://www.gao.gov/about/index.html>.
12. Office, United States General Accounting. "Changing Conditions Drive Need for New F/A-22 Business Case." 2004.
13. Charette , Robert N. "This Car Runs on Code." IEEE Spectrum 2009.
14. "Electronics: Driving Automotive Innovation." Pictures of the Future 2005, Fall ed.
15. America, One Hundred Eleventh Congress of the United States of. "National Defense Authorization Act for Fiscal Year 2010." 2010.
16. Wells, Charles (LTC). "Information Technology Requirements Oversight and Managment (The "IT Box"." 2009.
17. JROC. "Joint Requirements Oversight Council." 2009.
18. "Defense Acquisition Guidebook." 2010.
19. ISO/IEC. "Systems and software engineering - System life cycle processes." 2008.
20. "Systems engineering – Application and management of the systems engineering process." 2005.
21. IEEE. "IEEE Standard for Application and Management of the Systems Engineering Process." 2005.
22. ANSI/EIA. "Processes for Engineering a System." 1999.
23. Force, Secretary Of The Air. Life Cycle Systems Engineering. 2007
24. CMMI® for Development, Version 1.2. Pittsburgh: Carnegie Mellon University, 2006.
25. "ISO 9001." Quality Management Systems. 2008.
26. Standardization, International Organization for. "ISO 12207." Software Life Cycle Processes. 2008.
27. Software Engineering Institute - Carnegie Mellon. "Brief History of CMMI." n.d.
28. Sutherland, Jeff and Ken Schwaber. "The Scrum Papers: Nut, Bolts, and Origins of an Agile Framework." 2010.