CROSSTALK

Sept/Oct 2014 The Journal of Defense Software Engineering Vol. 27 No. 5

ACQUISITION OF Software-Reliant Capabilities

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
OCT 2014		**00-09-2014 to 00-10-2014**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
CrossTalk, The Journal of Defense Software Engineering. Volume 27, Number 5. Sep/Oct 2014	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **CrossTalk / 517 SMXS MXDED,6022 Fir Ave Bldg. 1238,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

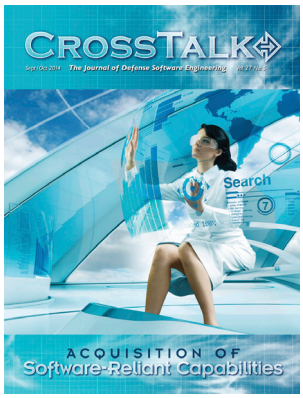
14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**46**	

TABLE OF CONTENTS

Departments



Cover Design by Kent Bingham

Acquisition of Software-Reliant Capabilities

CrossTalk

NAVAIR Jeff Schwalb
DHS Joe Jarzombek
309 SMXG Karl Rogers

Publisher Justin T. Hill
Article Coordinator Heather Giacalone
Managing Director David Erickson
Technical Program Lead Thayne M. Hill
Managing Editor Brandon Ellis
Associate Editor Colin Kelly
Art Director Kevin Kiernan

Phone 801-777-9828
E-mail Crosstalk.Articles@hill.af.mil
CrossTalk Online www.crosstalkonline.org

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Security (DHS). USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: Office of Cybersecurity and Communications in the National Protection and Programs Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of **CrossTalk** providing both editorial oversight and technical review of the journal. **CrossTalk's** mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

Subscriptions: Visit <www.crosstalkonline.org/subscribe> to receive an e-mail notification when each new issue is published online or to subscribe to an RSS notification feed.

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the **CrossTalk** editorial board prior to publication. Please follow the Author Guidelines, available at <www.crosstalkonline.org/submission-guidelines>. **CrossTalk** does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the authors and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with **CrossTalk.**

Trademarks and Endorsements: **CrossTalk** is an authorized publication for members of the DoD. Contents of **CrossTalk** are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CrossTalk Online Services:
For questions or concerns about crosstalkonline.org web content or functionality contact the **CrossTalk** webmaster at 801-417-3000 or webmaster@luminpublishing.com.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

CrossTalk is published six times a year by the U.S. Air Force STSC in concert with Lumin Publishing <luminpublishing.com>.
ISSN 2160-1577 (print); ISSN 2160-1593 (online)

CrossTalk would like to thank DHS for sponsoring this issue.

When organizations purchase software or software-reliant capabilities and services with inadequate consideration for built-in cybersecurity and software assurance, the residual risks of exploitability persist throughout the life of the purchased capability. The lasting effect of inadequate software assurance in procured items is part of what makes acquisition reform so important to achieving mission resiliency, and could provide a more enabling role for achieving more comprehensive cybersecurity. Meanwhile, due to the growing sophistication and complexity of software that controls and enables Information and Communications Technology (ICT), military and federal agency information systems are increasingly at risk of compromise, and organizations need guidance to help manage software supply chain risks. Requirements specification, acquisition and procurement activities must incorporate software assurance criteria as part what constitutes "technically acceptable" and "fit-for-use" in higher assurance mission environments.

The Federal Acquisition Regulation (FAR) continues to change; focusing on the fact that supply chain introduces risks to essential information and services. What progress have we made, and what do we still need to do to better safeguard society and missions from supply chain risks? What should acquisition managers include, as part of their due-diligence, in considering potential suppliers of ICT/software products and services? In the 2014 report "Improving Cybersecurity and Resilience through Acquisition " <http://gsa.gov/portal/content/176547>, in response to Section 8(e) of the President's Executive Order 13636, GSA and DoD, in coordination with DHS, made recommendations regarding the feasibility, security benefits, and relative merits of incorporating security standards into acquisition planning and contract administration; specifically recommending acquisition reforms. More explicit measures are now being advocated because our ICT/software assets are under constant attack. Thwarting the active attacker is not something most designers, engineers, developers, or project managers normally consider or have been trained to address. Yet encouraging resilience criteria in every stage of development and supply of ICT must continue to be the forward-leaning focus of the Software and Supply Chain Assurance efforts within government and industry. Attacks against our supply chains unite acquirers and suppliers in the search of scalable means for sharing information about ICT/software risks that arise through malice or negligence. Suppliers and acquirers need standardized means for conveying information about common issues, especially regarding non-conforming products that might contain counterfeit, tainted, or defective components that can enable exploitation or cause subsequent harm.

Unfortunately, partially because of the lack of adequate due-diligence in acquisition, vulnerabilities are proliferating rapidly thus stretching our mission capabilities and resources. As we seek to discover and mitigate the root causes of these vulnerabilities, sharing the knowledge we have of them helps to mitigate their impact. In order to keep pace with the threat, we must facilitate the automated exchange of information. To achieve that, DHS sponsors "free for use" standards, such as the Common Weakness Enumeration (CWE), which provides standardized means for identifying and mitigating architectural, design, and coding flaws introduced during development and prior to use, and the Common Attack Pattern Enumeration and Classification (CAPEC), which enables developers and defenders to discern attacks, build software resistant to them, and determine the sufficiency of test regimes. These open specifications for interoperable security automation enable secure, machine-to-machine communication of actionable indicators between organizations that want to share this information. The components have been developed collaboratively between Federal Government and industry partners working toward information sharing mechanisms and solutions to reduce the risk of tainted ICT/software components. These standardized means for sharing information are already being used, and they contribute to our efforts to enable all stakeholders to secure their part of cyberspace.

DHS, DoD, NIST, and GSA co-sponsor the Software & Supply Chain Assurance (SSCA) Forum in which Federal, academic, and industry stakeholders discuss risks and mitigation methods. SSCA Forums are free and open to the public -- see details at <http://gsa.gov/portal/content/194963>, and resources are available on the SSCA Community Resources and Information Clearinghouse (CRIC) at <https://buildsecurityin.us-cert.gov/swa>.

Venues such as the SSCA Forum are critical to our understanding of how suppliers incorporate security-aware practices into the production of software. An appreciation of how acquisition and procurement can exercise proper due-diligence can inform risk-based decisions when purchasing software or contracting for software-reliant systems or services. This issue of CrossTalk includes articles focused on advancing software assurance as part of acquisition of software-reliant capabilities that can provide valuable insights into techniques, practices, methods, and models that target and mitigate vulnerabilities in the supply chain.

Justin Hill
CrossTalk Publisher

Using Concept Maps to Introduce Software Security Assurance Cases

Dallas Snider, University of West Florida
John Coffey, University of West Florida
Thomas Reichherzer, University of West Florida
Norman Wilde, University of West Florida
Chris Terry, Comnet Marketing Group
Joe Vandeville, Northrop Grumman
Allison Heinen, Northrop Grumman
Sarah Pramanik, Northrop Grumman

Abstract. To improve the security of software systems, we need to improve the software development processes used to produce them. Software security assurance cases have been proposed as a way of establishing security properties of software at different phases of the software development lifecycle; however, these assurance cases are difficult to write, communicate and introduce into an already burdened software development process. We evaluated a team-based, knowledge engineering approach to introduce software security assurance cases to neophytes through the utilization of concept maps. This approach allowed the study's participants to engage in conversations with security experts about security requirements for their software and with knowledge engineers to construct concept maps demonstrating how their software met the requirements. Our survey results and feedback show great promise for our method to be effective and efficient for disseminating knowledge about software security to new hires and students which in turn would make them cognizant of the security requirements in their organization.

Introduction

Enhancing software security is, in part, about software process improvement. To improve the security of software systems we need to improve the software development processes used to produce them. Software security assurance cases have been proposed as a way of establishing security properties of software at several security touchpoints in the development process [1]. These touchpoints are defined as "lightweight software security best practice activities that are applied to various software artifacts such as requirements and code" [2]. Assurance cases are structured and reviewable artifacts to demonstrate that a system possesses required security properties. However assurance cases are not easy to write, communicate, or introduce into an already burdened software development process.

So how can we introduce software security assurance cases to a development organization or to a new hire in such an organization? In this paper we propose a team-based, knowledge engineering approach that introduces assurance cases through the use of concept maps [3]. A concept map provides a simple visual representation for the capture and communication of technical knowledge about a particular domain.

The knowledge engineering approach we employed combines initial skeleton concept maps that describe known types of software vulnerabilities, concept maps of the software structure derived by parsing code or interface documents from a particular system, an interview process in which a knowledge engineer facilitates a conversation between a security expert and a programmer to develop the concept map-based assurance case for that system, and a review process in which the resulting assurance case is presented to stakeholders.

If successful, this approach could have several advantages. First, the knowledge engineering approach could facilitate the introduction of assurance case touchpoints into the development process. Second, participation in the interviews could improve programmer sensitivity to potential software vulnerabilities and expedite knowledge transfer. Furthermore, the concept maps supporting the assurance cases could be linked to other documentation such as design rationale to provide an integrated view of the software structure. Finally, the visual nature of assurance cases enhanced with concept maps could facilitate communication with diverse project stakeholders.

In this paper we provide a discussion of literature pertaining to software security assurance cases and concept mapping. We then discuss two small-scale case studies that were conducted to gain feedback on the practicality of a knowledge engineering approach to the construction of assurance cases containing concept maps. We summarize results of these studies including the results of questionnaires that address the utility of this approach and conclude with a discussion of lessons learned.

Software Security Assurance Case Development

Security assurance is a multidimensional concern. Evaluation of security requires evaluating targets (systems that may be attacked), processes (used to develop the targets), and remediation (corrective action to mitigate vulnerabilities) [4]. Software security assurance cases fall at the juncture of these three concerns; they are developed for a specific target as part of its software engineering process, and they may identify vulnerabilities for remediation.

An assurance case is a body of evidence that is analogous to a case presented in legal proceedings and is basically an argument assuring that some claim about a system holds. Assurance cases are structured and reviewable artifacts used to document that a system possesses required properties such as security, safety and reliability.

Software security assurance cases are prepared at defined security touchpoints during the software development lifecycle. Throughout this lifecycle, the software security assurance case evolves as it is reviewed by stakeholders with varying areas of expertise. Besides security experts, these stakeholders can include developers, testers, installers, system administrators and customers [5].

A proven software security assurance case is reusable for new versions of an existing system and may be adaptable for a completely new system. The software security assurance case creates a structure for the analysis of changes to a system and helps to ensure that changes do not have adverse effects on security by introducing new vulnerabilities. A properly created

assurance case will take into consideration the personnel, processes and technology involved in the development and delivery of software. Arguments based on personnel quality might be based on an employee's training and experience. Process arguments could be based on established testing procedures, bug tracking, code management and release schedules. Technology arguments could cover properties of hardware, operating systems and integrated development environments [6].

One of the challenges in creating a software security assurance case is the large number and diverse nature of the security requirements. These requirements are often introduced by citing large external checklists and standards such as the U.S. DoD's Application Security and Development Checklist [7] and 8500.1 Information Assurance Implementation [8] along with the U.S. Department of Commerce's SP800-53 Recommended Security Controls for Federal Information Systems and Organizations [9]. Not following these requirements correctly or in a timely manner can cost organizations millions of dollars in extra costs, damages or lost business. In addition to being lengthy, these requirements documents are subject to interpretation, which might require the opinion from someone in an organization's legal department. Requirements documents are sometimes conflicting which leads to the question of which document takes precedence. The result is a lengthy security assurance case document to address this myriad of requirements. These assurance cases are expensive to produce and are also subject to interpretation [10].

Another challenge is that a software security assurance case often requires a combination of scarce knowledge from different experts. A development organization may have security experts with deep knowledge of relevant kinds of security vulnerabilities. It will also have software developers with deep knowledge of a particular system. But these will not normally be the same people. Further exacerbating this problem is the fact that security requirements are constantly changing due to new threats and exposed vulnerabilities. Also contributing to the problem are personnel changes in information security departments and software development teams resulting in scarce knowledge of security and source code walking out the door.

One of the goals of this work is to determine if techniques from the discipline of knowledge engineering could help, at least to "break the ice" when an organization is adopting assurance cases for the first time, or when a new employee is being introduced to security assurance.

Knowledge Capture With Concept Maps

Concept maps provide a simple visual representation of the concepts in a particular domain and of the relationships between them. (See Figure 1 for a concept map pertaining to knowledge management.) Concept mapping has been used extensively to capture domain knowledge for purposes of knowledge preservation, consensus finding, and subsequent construction of formal knowledge representations for automated reasoning. Concept maps are well established in a wide variety of domains and experience levels and they are excellent for communicating structured knowledge [11].

Concept maps can be created by individuals or groups at levels of capability that extend from elementary school children to subject matter experts (SMEs). Concept mapping for knowledge capture

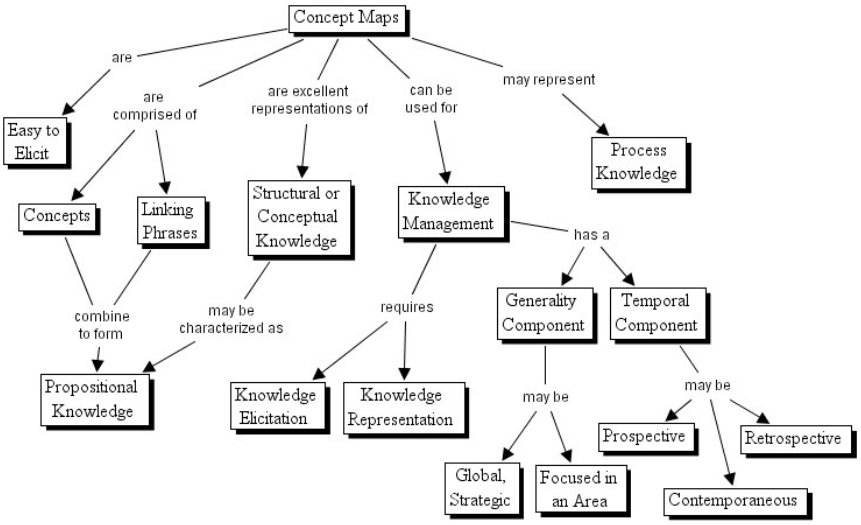


Figure 1. A concept map about concept maps used for knowledge management.

often involves more than one participant: one who is an expert in the knowledge domain, and another who is skilled at concept mapping. The person who is skilled at eliciting and representing knowledge from experts, the knowledge engineer, creates the concept map during an interview process with the SME. Mature, open-source applications for concept map development such as the Institute for Human and Machine Cognition's CmapTools are available for use [12]. CmapTools allows the knowledge engineer to populate hierarchical groups of maps and to provide drill-through links to other electronic resources such as source code and configuration files [13].

For software security assurance cases, there are typically at least two kinds of knowledge that must be brought to bear: expertise about security vulnerabilities and expertise about the specific software system under consideration. Accordingly, we suggest a team of three for the interview process as shown in Figure 2. This approach to the development of an assurance case involves the security expert interviewing the developer with questions pertaining to specific security issues, while the knowledge engineer captures the key ideas in the discussion in concept maps. The interview process captures the specialized knowledge of the security expert and allows this knowledge to be transferred to new hires verbally while creating a document that will capture this specialized knowledge for posterity. The concept maps can be augmented with other resources such as program code or other documentation that lend support to the emerging case such as testing protocols, test results and snippets of source code. Issues that need to be addressed can be represented along with the evidence for issues that have been properly addressed.

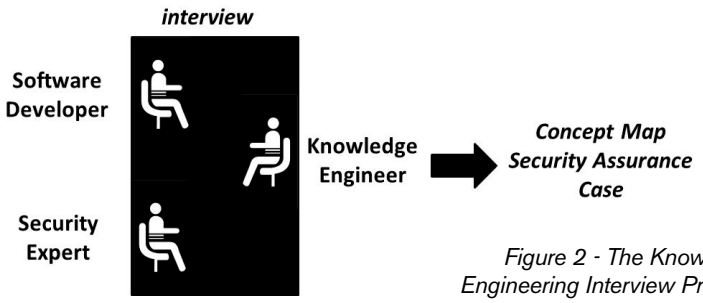


Figure 2 - The Knowledge Engineering Interview Process

Case Studies

In this section, we describe two case studies that were performed to test the potential of the approach to developing software security assurance cases. The two studies were both small-scale exercises to provide a sort of "sanity check" on using concept maps for this purpose. We wanted to see if early-career developers could create and review such cases. Further we wished to gather feedback from participants on how well such cases communicated the basic idea of an assurance case, relevant knowledge pertaining to the case, and any difficulties in the process.

The first study was academic and performed in the context of a project course for Masters of Science students in Software Engineering at the University of West Florida. Participants in the study were mature students, with some professional experience either in programming or system administration. However none had any particular background in software security. All activities were conducted online since the students were distributed geographically. The main theme of the course was the development or maintenance of services to enhance an existing Services Oriented Architecture application. Their development process included steps for:

1. **Preliminary design**
2. **Design Phase Software Security Assurance Case (touchpoint)**
3. **Design Phase Software Security Assurance Case Review, moderated by a student Software Quality Assurance (SQA) leader**
4. **Implementation and Deployment**
5. **Deployment Phase Software Security Assurance Case (touchpoint)**
6. **Deployment Phase Assurance Case Review, moderated by the student SQA leader**

These steps are shown in Figure 3.

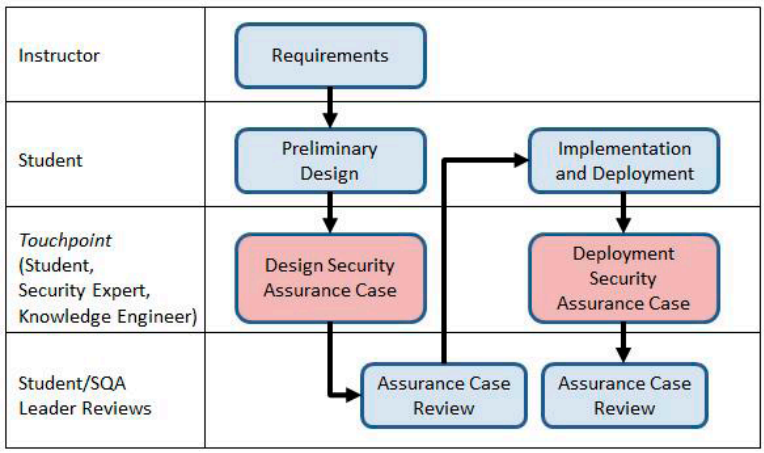


Figure 3 – Student Software Security Assurance Case Process

To develop the assurance cases for their services, the students were given four security requirements drawn from the literature [4]:

1. **Service is not subject to error handling vulnerabilities**
2. **Service supports the creation of transaction logs for access and changes to data**
3. **Service does not contain embedded authentication data**
4. **Service executes with no more privileges than necessary for proper operation.**

Participants were given an initial skeleton concept map presenting the four software security vulnerabilities as shown in Figure 4 to jump-start the knowledge elicitation process. Two knowledge engineering sessions were conducted, one at each touchpoint. During both of these sessions, the participant was interviewed by a security expert responsible for asking questions related to the security requirements and a knowledge engineer responsible for capturing knowledge through the construction of the concept map. For the post-design interview, the knowledge engineer started with the skeleton concept map and added concepts and notes for follow-up items for identified deficiencies in the design. During the post-implementation interview, the knowledge engineer added concepts, code samples and interface documents that would help to make the case that the code was secure. Following both interviews, the participants presented their concept map assurance cases for review by their peers.

Figure 5 is a screen shot showing a portion of one of the resulting assurance cases. This figure shows the analysis for the transaction logging security requirement mentioned earlier. This student did a PHP implementation of his service. He has chosen to use a combination of the Apache web server log, a text file log and a MySQL log to meet the requirement. The icon in the "logEvent()" box indicates that there is a sample of the code that can be inspected to confirm that all service operations actually call the logging function.

The second case study was done in a one-day on-site visit at a Northrop Grumman facility in Melbourne, Florida. The objective of the study was to see if participants could learn and apply concept mapping for the development of assurance cases in this short period. A total of eight participants took part in this study, four interns and four recent hires. Participants in the study had backgrounds in computer science and software development. They were familiar with graphical representations of conceptual knowledge such as UML class diagrams. Participants were provided with an introduction to concept mapping, knowledge modeling, and CmapTools. Following this introduction, the participants created concept maps on topics of their choosing to gain experience with concept mapping and with the CmapTools software.

Participants were then presented with information regarding how concept mapping and knowledge modeling might be used to build assurance cases. Participants then worked in teams to build concept maps pertaining to assurance cases. After each round of concept mapping, participants showed the maps they had created to the group with the goal of assessing strengths and weaknesses in the maps, and developing suggestions for their improvement. At the end of both case studies, each participant was asked to complete a survey with the five Likert statements shown in Table 1 and to provide comments to support their responses to the Likert statements.

Results and Discussion

The Likert statements were based on a 6-point scale with 1 representing strongly agree and 6 representing strongly disagree. Table 1 combines the results for the two groups.

For the first statement, some of the comments from the participants focused on how concept maps made the security

requirements and the design goals easier to understand. Other comments mentioned how the process helped the participants to learn the importance of secure software and to identify weaknesses in their designs along with understanding the design goal.

With the second statement, the feedback provided mentioned the process giving the participants an additional understanding of the vulnerabilities when looking for ways and methods to develop the evidence needed to prove the vulnerability had been addressed. Another comment that reflected the lower mean score for the second statement mentioned the knowledge engineering process increased their awareness, but did not provide an in-depth understanding.

For the third statement, participants commented that the CmapTools application was a great visual aid and the concept maps made their peer review presentation easier. Furthermore, the ability to drill down to the source code, configuration files and design documents was beneficial.

For the fourth statement, the participants mentioned that the collaboration during the interview sessions helped in the sharing of ideas and broadened the scope of available knowledge. Regarding the concept maps, the participants stated that the visual primitives on the maps allowed for comprehension of the vulnerabilities and the maps ensure that all the components of the cases are addressed and the relationships between the components are documented as well.

Out of all of the statements, the final statement had the least amount of agreement and the greatest dispersion of responses. Multiple responses focused on the need to continue to seek out assistance to ensure the system followed security requirements. Other comments stated how the process helped to instill confidence in the participants' ability to build secure software. While our process might not have provided the neophyte with the in-depth knowledge required to build a complete assurance case, the process is an important first step in becoming cognizant of an organizations software security requirements.

Conclusions

This paper has concentrated on the use of knowledge engineering in "first time" experiences with software security assurance cases. In the two studies described here, a total of 12 participants were introduced to concept mapping and its application to the development and presentation of software security assurance cases. Though the sample size is relatively small, we believe that the results are encouraging. Results from surveys administered to all participants indicate that knowledge engineering and concept maps could have a useful role in "breaking the ice" for software security assurance case practices and raising the level of security competence.

The approach of introducing a new developer to the idea of software security assurance cases through collaborations with a security expert and a knowledge engineer who captured the proceedings in real time, worked well. The primary additional cost of this approach is the time of the knowledge engineer in the interviews. Each interview lasted only an hour, or roughly 15 minutes for each security checklist item. The resulting concept maps were evaluated by participants to be an excellent communication tool for reviews and other subsequent uses. The

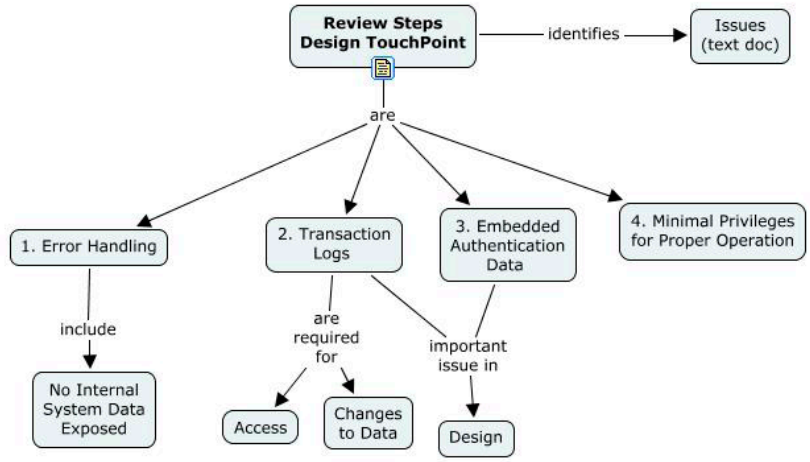


Figure 4. The initial skeleton concept map showing the four security vulnerabilities.

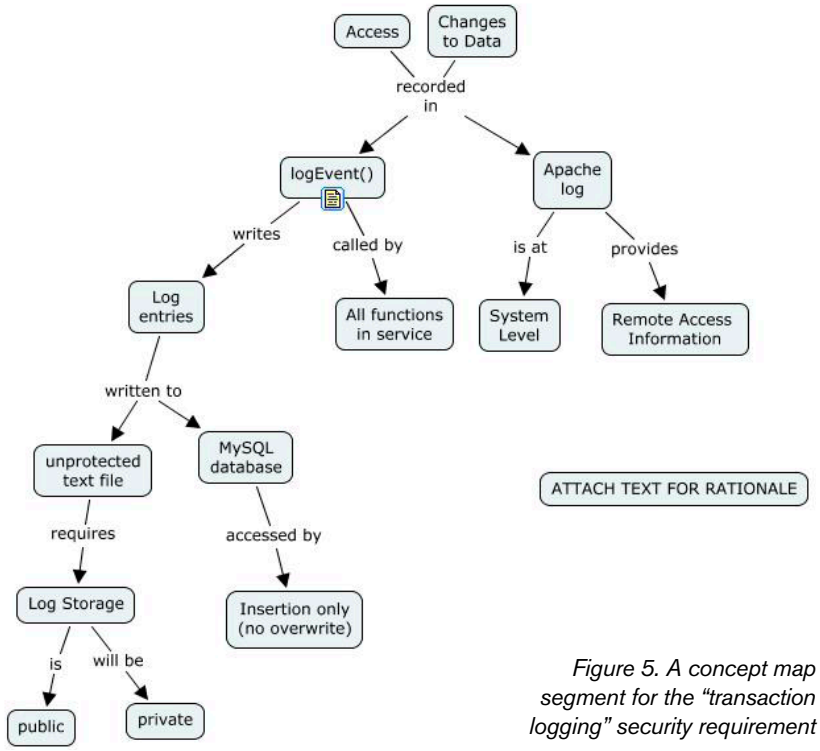


Figure 5. A concept map segment for the "transaction logging" security requirement

Statement Number	Statement 1 = *strongly agree*, 6 = *strongly disagree*	μ	σ
1.	On finishing the team-based process in this course, I feel that I understand how security assurance cases may be prepared and used as part of the software development process.	1.67	0.62
2.	On finishing the team-based process in this course, I feel that I understand the four software vulnerabilities and how to mitigate them.	2.00	1.08
3	The concept map security assurance case was easy to present in the peer review session.	1.50	0.76
4.	If in the future I need to prepare security assurance cases again, I would like to use this same team-based process.	1.42	0.49
5.	If for some future system I need to prepare a concept map security assurance case for these same vulnerabilities, I am confident I could do so with less assistance from the security expert and knowledge engineer.	2.58	1.26

Table 1 - Sample mean and standard deviation results from post-survey (N=12)

knowledge represented in the concept maps provided clarity in understanding software vulnerabilities and how to design and implement a system to handle these vulnerabilities. We found the participants adept at understanding the graphical primitives that constituted the concept map. One observation from our study at Northrop Grumman was the participants quick mastering of the creation of concept maps and the understanding of the graphical primitives because of their familiarity with linked graphical representations such as UML class diagrams. Furthermore, the concept maps helped the participants to share knowledge among their peers and supervisors.

In discussions with our collaborators at Northrop Grumman, we determined several additional potential benefits of using our knowledge engineering process to create concept maps for software security assurance cases. First, our process could help streamline the creation of testing abuse cases and unit tests for security assertions. Second, the process can assist with getting stakeholders on the same page in regards to interpreting security requirements. Furthermore, concept maps could help to communicate to all levels of management the cost implications of security requirements which would help to strike a balance between security and affordability, thus contributing to reducing overall program costs.

Acknowledgments

Funding for this project was provided by Northrop Grumman through the Security and Software Engineering Research Center and by the University of West Florida Foundation under the Nystul Eminent Scholar Endowment. We would also like to thank the students and interns who participated in the study, particularly Ms. Heather Dennison the SQA leader.

REFERENCES

1. Allen, J. H., et al. Software Security Engineering: A Guide for Project Managers. Addison-Wesley, 2008.
2. McGraw, G. Software Security: Building Security In. Boston: Addison Wesley, 2006.
3. Novak, J. D. and D. B. Godwin. Learning How to Learn. Cambridge University Press, 1984.
4. Bayuk, J. and A. Mostashari. "Measuring Systems Security." Systems Engineering 16.1 (2013): 1-14.
5. Vivas, J., I. Agudo and J. Lopez. "A Methodology for Security Assurance-Driven System Development." Requirements Engineering 16.1 (2010): 55-73.
6. Weinstock, C. B., H. F. Lipson and J. Goodenough. Arguing Security - Creating Security Assurance Cases. Jul. 2013. Aug. 2013 <https://buildsecurityin.us-cert.gov/articles/knowledge/assurance-cases/arguing-security---creating-security-assurance-cases>.
7. U.S. Department of Defense. Application and Security and Development Checklist, Version 3, Release 8 Jul. 2014.
8. U.S. Department of Defense. Instruction 8500.1 - Cybersecurity Implementation Mar. 2014.
9. U.S. Department of Commerce, National Institute of Standards and Technology. NIST Special Publication 800-53, Recommended Security Controls for Federal Information Systems and Organizations, revision 4. Jan 2014.
10. Pramanik, S. "Increasing Security for Regulatory Constrained Systems, Through New Systems and Architecture Methodologies." Ph.D. Dissertation, C.S., U.C.C.S., Colorado Springs, CO, 2013.
11. Coffey, J. W. and T. Eskridge. "Case Studies of Knowledge Modeling for Knowledge Preservation and Sharing in the U.S. Nuclear Power Industry ." Journal of Information and Knowledge Management 7.3 (2008): 173-185.
12. Institute for Human and Machine Cognition. CmapTools Home Page. n.d. 31 Oct. 2013 <http://cmap.ihmc.us>.
13. Cañas, A. J., et al. "CMAPTOOLS: A Knowledge Modeling and Sharing Environment." Proc. of the First Int. Conf. on Concept Mapping. Pamplona, 2004.

ABOUT THE AUTHORS

Dallas Snider is an Assistant Professor of Computer Science at the University of West Florida. He received his Ph.D. in Integrated Computing and M.S. in Instrumental Sciences from the University of Arkansas at Little Rock. He received a B.A. in Physics from Hendrix College. Dr. Snider's teaching and research interests include data mining, data warehousing and cybersecurity.

Phone: 850-473-7348
E-mail: dsnider@uwf.edu

John Coffey holds an Ed.D. with an emphasis in Computer Science from the University of West Florida (UWF). He was one of the first members and ongoing collaborator with the Institute for Human and Machine Cognition. He is currently a professor in the UWF Computer Science Department and has over 100 publications. His research interests include advanced education technology, knowledge elicitation and representation, student modeling, web services and Service Oriented Architecture.

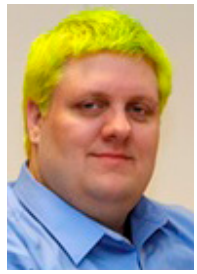
Phone: 850-474-3183
E-mail: jcoffey@uwf.edu

Dr. Thomas Reichherzer is an Assistant Professor in the Computer Science Department at the University of West Florida. He received a Ph.D. in Computer Science at Indiana University in 2009. Dr. Reichherzer's interest include knowledge representation, the Semantic Web, information retrieval, management, and visualization. More recently, he focused on sentiment analysis of unstructured text and AI approaches to smart home environments.

Phone: 850-474-2548
E-mail: treichherzer@uwf.edu

Norman Wilde received his Ph.D. from MIT in 1971. He spent a number of years working in developing countries overseas: in academic positions, with the World Health Organization, and as an independent systems consultant. Dr. Wilde is currently Nystul Chair and Professor of Computer Science at the University of West Florida. He has worked extensively with the Security and Software Engineering Research Center (S2ERC) on research projects in Software Maintenance and Software Security.

Phone: 850-474-2548
E-mail: nwilde@uwf.edu



Chris Terry has 14 years of experience working in the Information Technology industry and is currently the Senior Systems Engineer with Comnet Marketing Group, a national marketing and fundraising firm specializing in nonprofit organizations. As a software engineering student at the University of West Florida he won national awards for competitive programming and security analysis. He is currently overseeing a companywide virtualization and services orientated migration project and supervises day-to-day IT security operations.

Phone: 541-864-1470
E-mail: chris.terry@comnetmarketing.com

Joe Vandeville is an Embedded Software Engineer at Northrop Grumman Aerospace Systems, working in the area of system architecture. He has been associated with the development of software intensive systems for over 40 years, and has been involved in hardware and software development in both commercial and military systems. He has Masters Degrees in both Computer Science and Systems Engineering and is currently pursuing a PhD in Systems Engineering.

Phone: 321-951-5287
E-mail: joseph.vandeville@ngc.com

Allison Heinen is a member of the Northrop Grumman Aerospace Systems Software Products organization located in Melbourne, Florida. She has over 30 years of experience in Software Engineering and has held many leadership roles managing functional organizations, technical projects and process related initiatives. She is currently the Software Lead for process and tool related initiatives. Ms. Heinen holds a Bachelor of Science degree as well as two Master of Science degrees.

Phone: 321-726-7657
E-mail: allison.heinen@ngc.com

Sarah Pramanik is the lead for software information assurance (IA) on her current program. She is responsible for leading the development, integration and execution of IA activities. Dr. Pramanik has worked in multiple areas of cybersecurity, including vulnerability and penetration testing, information system security engineering, cybersecurity training and security architecting. She holds a Bachelors and Masters degree in Computer Science and a Ph.D. of Engineering in Security from The University of Colorado, Colorado Springs.

Phone: 516-346-7737
E-mail: Sarah.Pramanik@ngc.com

Evaluating Software Assurance Knowledge and Competency of Acquisition Professionals

Dan Shoemaker, University of Detroit Mercy
Nancy R. Mead, SEI

Abstract. As the potential for highly destructive cyberattacks grows, organizations must ensure that their procurement agents acquire high quality, secure software. ISO 12207 and the Software Assurance Competency Model, when used together, provide a clear view of the activities, knowledge, and competencies required to procure secure software.

The Benefit of Standardized Acquisition

Gartner forecasts that the worldwide dollar-valued IT spending forecast will grow 3.1% in 2014, reaching $3.8 trillion [1]. Considering the magnitude of this investment, organizations should work hard to ensure the effective acquisition of systems and software. This task is a complicated one; the success or failure of any acquisition effort depends on the capability of the individuals who do the work, and those individuals' capability depends on knowledge and experience. While an experienced and knowledgeable procurement agent may deliver the desired result, one who is inexperienced or incapable may bring about a disaster. Establishing capability requirements for every person involved in the acquisition process is vital for organizations to preserve their investment in technology.

It is essential to use standard criteria to judge the performance of any task; standard criteria allow actions to be judged objectively. Having a standard set of criteria also ensures coordinated management of the process. Though the benefits of coordinated management are manifold, the primary advantage is that defining a process enables repeatability. Looking back, the entire decade of the 1990s seems to have been devoted to detailing the benefits of repeatable processes. That thinking was probably best expressed in A Discipline for Software Engineering [2]. The justification for a well-defined, documented, and systematically executed process is that it can be more effectively managed and continuously improved [2]. A single, comprehensive set of standard criteria to guide the work also ensures efficient communication between participants, which, in turn, ensures a more suitable final product.

Repeatability requires consistent execution of the fundamental activities of a process. According to conventional wisdom within the software industry, standards convey those requisite activities. Standards define the fundamental requirements for the performance of a given process. A properly written and administered standard will ensure that every participant in the process knows and follows principles and practices that have track records of success. Concerning this discussion, there are several official and quasi-official standards for acquisition. The standards for acquisition include IEEE 1062-1998, an eight-page collection of high-level recommendations for ensuring quality in software acquisition [3]. There are guidelines that provide recommendations for the security testing of government off-the-shelf (GOTS) and commercial off-the-shelf (COTS) products [4]. However, these recommendations in no way constitute a complete process. The Common Body of Knowledge to Produce Acquire and Sustain Secure Software itemizes a complete set of principles and practices for secure acquisition [5]. However, this white paper does not provide general guidance [6].

The almost total absence of comprehensive lifecycle recommendations for acquisition might be explained by the dominant role of ISO 12207-2008, both internationally and in the United States [7]. That standard documents a comprehensive set of activities and supporting tasks to establish effective lifecycle acquisition of system and software products. The standard dictates a complete set of highly interdependent lifecycle activities for proper execution of the supply and reuse process, in addition to explicit acquisition recommendations. The standard also provides comprehensive advice about to how to carry out the ancillary activities that are necessary to support those processes, such as documentation, software quality assurance, and configuration management.

Factoring the 21st Century into the Equation

All of the existing standards for acquisition could serve as a basis for structuring a repeatable lifecycle acquisition function. However, with the exception of the Common Body of Knowledge to Produce Acquire and Sustain Secure Software, they are all oriented toward assurance of product quality. Though most of the standard activities associated with product quality (e.g., planning, testing, reviews, audits) still have currency in this discussion, the ever-increasing threats in cyberspace have added a new dimension to the requirements for a capable and successful procurement process. Thus, it is critical that acquirers adopt and follow assurance practices to ensure that products not only operate as intended, but also have sufficient integrity to withstand attack.

The need for secure products makes the problems associated with ensuring the quality of the purchased product almost nostalgically simple. A recent report summarizes the security issues facing all acquirers; the report uses five categories—each with a different implication for acquirers—to classify these concerns [8]:

- **installation of malicious logic on hardware or software**
- **installation of counterfeit hardware or software**
- **failure or disruption in the production or distribution of critical products or services**
- **reliance upon a malicious or unqualified service provider for the performance of technical service**
- **installation of unintentional vulnerabilities on software or hardware**

These categories highlight a central question: "Do acquisition personnel have the capability to ensure that purchased system and software products are free of these threats?"

Though the past decade has produced a number of acceptable methods for assuring the security of the product [9, 10], ensuring the ability of the individual worker to apply these approaches

is difficult. A new model from the Carnegie Mellon University Software Engineering Institute (SEI), the Software Assurance Competency Model, establishes a foundation for assessing the capability of software assurance professionals [11]. The model can be used by individuals to assess their own capabilities and professional goals, and by organizations to assist in staffing and building teams with appropriate competencies. At present, there is not a competency exam associated with the model, which is intended to be instantiated by organizations for their own use.

This model, which has been endorsed by the IEEE Computer Society, portrays the requisite competencies for software assurance work across a range of knowledge areas [11]. The competency areas captured in this model are 1) Assurance Across the Lifecycle, 2) Risk Management, 3) Assurance Management, 4) Assurance Assessment, 5) System Security Assurance, 6) System Functionality Assurance, and 7) System Operational Assurance. The model is further decomposed into individual units based on knowledge and skills. Those knowledge and skill units can be ranked at competency levels 1 through 5 [9]. The Software Assurance Competency Model provides a common definition of the activities required to ensure a secure product, and it uses a competency-based evaluation scheme. The model's knowledge and competency stipulations can be combined with the acquisition process recommendations from ISO 12207 to define a set of standard, competency-based acquisition processes for any organization. This amalgamation can then be used to judge whether a given acquisition process is being performed at a sufficient level of capability.

A Competency-Based Model for Secure Acquisition Practice

The SEI Software Assurance Competency Model comprises seven competency areas, which are decomposed into 20 knowledge units. Some of these knowledge units are devoted to elements of software work that do not involve acquisition. However, 13 of those 20 units can apply to ensuring a secure acquisition: Software Lifecycle Processes, Software Assurance Processes and Practices, Risk Management Concepts, Risk Management Processes, Software Assurance Risk Management, Assurance Assessment Concepts, Measurement for Assessing Assurance, Making the Business Case for Assurance, Managing Assurance Compliance Considerations, Assurance Ethics and Integrity in Creation, Acquisition, and Operation of Software, Systems Assurance Technology, Assurance in Acquisition, Operational Monitoring, System Control, and Operational Procedures [11].

Each of these knowledge units is tied to a staged set of competencies. Table 1 provides the general definition of these requisite abilities [11].

Integrating Standard Acquisition Practices with Competency Requirements

The areas in the SEI Software Assurance Competency Model cover the entire software and system assurance process. Though the SEI model does not specifically designate competencies for acquisition, ISO 12207 does specify an end-to-end set of acquisition practices. These practices have been standardized since 1995 [7]. Table 2 summarizes required practices for ISO 12207.

L1 – Technician	Possesses technical knowledge and skills, typically gained through a certificate or an associate degree program, or equivalent knowledge and experience
L2 – Professional Entry Level	Possesses application-based knowledge and skills and entry-level professional effectiveness; may also manage a small internal project, supervise and assign L1 personnel, supervise and assess system operations, and implement accepted assurance practices
L3 – Practitioner	Possesses breadth and depth of knowledge, skills, and effectiveness; may also set plans, tasks, and schedules for in-house projects and may define and manage such projects and supervise teams on the enterprise level
L4 – Senior Practitioner	Possesses breadth and depth of knowledge, skills, and effectiveness and a variety of work experiences; has 5 to 10 years of experience and professional development; identifies and explores software assurance practices, manages large projects, interacts with clients
L5 – Expert	Advances the field by developing, modifying, and creating methods, practices, and principles at the organizational level or higher; has peer/industry recognition

Table 1: Staged Competencies for Each Knowledge Unit

1. Initiation • Prepare a concept or a need to acquire, develop, or enhance a product or service • Prepare a set of requirements, including relevant design, testing, and compliance standards • Prepare a risk and cost-benefit analysis for acquisition • Prepare a set of acceptance criteria and criteria for evaluation Prepare acquisition plan based on requirements, analyses, and criteria defined in prior steps
2. Request for Proposals • Document acquisition requirements depending on acquisition option selected • Define contract milestones • Specifically delegate implementation of requirements to responsible organizational entity
3. Contract Preparation and Update • Establish plans for supplier selection • Institute and carry out a negotiation process including contract preparation • Institute a process for change control
4. Supplier Monitoring • Prepare a plan for supplier review • Systematically review supplier during product preparation period
5. Acceptance and Completion • Perform acceptance reviews and testing • Institute systematic configuration management

Table 2: Standard Acquisition Steps for ISO 12207

Together, ISO 12207 and the SEI Software Assurance Competency Model describe the skills and competencies required to execute a software acquisition process. The complete set of acquisition practices specified in ISO 12207 can be combined with the knowledge units and competencies from the SEI Software Assurance Competency Model to provide an assurance knowledge and competency-based description for the standard activities of software and system acquisition.

Table 3 presents a suggested amalgamation of the ISO 12207 acquisition process requirements with the standard knowledge units of the SEI Software Assurance Competency Model (note: 12207 practices are in bold and SEI SwA Competency practices are in italics). The associated SwA Competency levels can be added to each of the individual SEI knowledge units based on the needs of the situation.

ISO 12207 Practice	SEI Software Assurance Competency Model Practice
Prepare a concept or a need to acquire, develop, or enhance a product or service	○ *Software Lifecycle Processes* ○ *Making the Business Case for Assurance* ○ *Ethics and Integrity in Creation, Acquisition, and Operation*
Prepare a set of requirements including relevant design, testing and compliance standards	○ *Software Assurance Processes and Practices* ○ *Risk Management Concepts* ○ *Risk Management Processes* ○ *Software Assurance* ○ *Risk Management Assurance Assessment Concepts* ○ *Measurement for Assessing Assurance*
Prepare a risk and cost-benefit analysis for acquisition	○ *Risk Management Concepts* ○ *Risk Management Assurance Assessment Concepts* ○ *Making the Business Case for Assurance* ○ *Assurance in Acquisition*
Prepare a set of acceptance criteria and criteria for evaluation Software Lifecycle Processes	○ *Software Assurance Processes and Practices* ○ *Risk Measurement for Assessing Assurance* ○ *Assurance in Acquisition* ○ *Operational Monitoring*
Prepare acquisition plan based on requirements, analyses, and criteria defined in prior steps	○ *Software Lifecycle Processes* ○ *Risk Management Concepts* ○ *Risk Management Processes* ○ *Software Assurance* ○ *Managing Assurance Compliance Considerations* ○ *Assurance in Acquisition* ○ *Operational Monitoring*
Document acquisition requirements depending on acquisition option selected	○ *Software Assurance Processes and Practices* ○ *Risk Management Processes* ○ *Software Assurance* ○ *Risk Management Assurance Assessment Concepts* ○ *Measurement for Assessing Assurance*
Define contract milestones	○ *Software Lifecycle Processes* ○ *Software Assurance Processes and Practices* ○ *Managing Assurance Compliance Considerations*
Specifically delegate implementation of requirements to responsible organizational entity	○ *Software Lifecycle Processes* ○ *Management Concepts* ○ *Risk Management Processes* ○ *Software Assurance* ○ *Risk Management Assurance Assessment Concepts*
Establish plans for supplier selection	○ *Making the Business Case for Assurance* ○ *Managing Assurance Compliance Considerations* ○ *Ethics and Integrity in Creation, Acquisition, and Operation* ○ *Assurance in Acquisition*
Institute and carry out a negotiation process including contract preparation	○ *Risk Management Processes* ○ *Software Assurance* ○ *Assurance in Acquisition*
Institute a process for change control	○ *Software Lifecycle Processes* ○ *System Control* ○ *Operational Procedures*
Prepare a plan for supplier review	○ *Software Assurance Processes and Practices* ○ *Risk Management Concepts* ○ *Risk Management Processes* ○ *Software Assurance* ○ *Risk Management Assurance Assessment Concepts* ○ *Measurement for Assessing Assurance* ○ *Systems Assurance Technology* ○ *Assurance in Acquisition* ○ *Operational Monitoring* ○ *System Control*
Systematically review supplier during product preparation period	○ *Management Processes Software Assurance* ○ *Measurement for Assessing Assurance* ○ *Managing Assurance Compliance Considerations* ○ *Systems Assurance Technology* ○ *Assurance in Acquisition* ○ *Operational Monitoring* ○ *System Control* ○ *Operational Procedures*
Perform acceptance reviews and testing	○ *Measurement for Assessing Assurance* ○ *Systems Assurance Technology* ○ *Assurance in Acquisition* ○ *System Control*
Institute systematic configuration management	○ *Systems Assurance Technology* ○ *Operational Monitoring* ○ *System Control* ○ *Operational Procedures*

Table 3: Creating a Competency-Based Model of Secure Acquisition Practice

Conclusion

The ability to guarantee a secure acquisition is far too important to the well-being of any organization to base its activities on individual virtuosity. Therefore, there is justification for a well-defined model of practice. ISO 12207 provides a commonly accepted statement of the complete set of practices necessary to conduct system and software acquisition. The acquisition activities and tasks specified in this standard have been accepted as correct for almost two decades [7]. The Software Engineering Institute has provided a model of the knowledge and competency levels needed to assure software and systems. Combining ISO 12207 and the Software Assurance Competency model to form a single description of the activities, knowledge, and competencies required to procure secure software and systems benefits the community as a whole.

The potential for highly destructive attacks directed through acquired software and system products is a reality in cyberspace. Whether the adversary is a nation state or a single hacker, it is presently far too easy to cause serious harm through the insertion of malicious and counterfeit objects into purchased software and systems. The inclusion of such tainted products in our national infrastructure could potentially threaten our way of life. Given the swiftness of technological change, it is excusable that organizations might not recognize the emerging importance of purchased software and systems. It is inexcusable, however, to know that threats exist and to stand idly by without doing anything about the situation. This paper suggests one approach organizations can take to better ensure the security of the products they buy.

Acknowledgments/Disclaimers:

Copyright 2014 Carnegie Mellon University

This material is based upon work funded and supported by the DoD under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

No warranty. This Carnegie Mellon University and Software Engineering Institute material is furnished on an "as-is" basis. Carnegie Mellon University makes no warranties of any kind, either expressed or implied, as to any matter including, but not limited to, warranty of fitness for purpose or merchantability, exclusivity, or results obtained from use of the material. Carnegie Mellon University does not make any warranty of any kind with respect to freedom from patent, trademark, or copyright infringement.

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

DM-0001096

ABOUT THE AUTHORS

Daniel P. Shoemaker, Ph.D., is Principal Investigator and Senior Research Scientist at UDM's Center for Cyber Security and Intelligence Studies. Dan is also a full time Professor and former Department Chair at University of Detroit Mercy.

Phone: 313-680-1434
E-mail: dan.shoemaker@att.net

Nancy R. Mead is a Fellow and Principal Researcher at the Software Engineering Institute (SEI). Mead is also an Adjunct Professor of Software Engineering at Carnegie Mellon University. She is currently involved in the study of security requirements engineering and the development of software assurance curricula.

Mead has more than 150 publications and invited presentations, and has a biographical citation in Who's Who in America. She is a Fellow of the Institute of Electrical and Electronic Engineers, Inc. (IEEE) and a Distinguished Member of the Association for Computing Machinery (ACM). Dr. Mead received her PhD in mathematics from the Polytechnic Institute of New York, and received a BA and an MS in mathematics from New York University.

Phone: 412-818-3454
E-mail: nrm@sei.cmu.edu

REFERENCES

1. Gartner Worldwide IT Spending Forecast <http://www.gartner.com/technology/research/it-spending- forecast/>
2. Humphrey, Watts. A Discipline for Software Engineering. Reading, MA: Addison-Wesley, 1995.
3. Institute of Electrical and Electronic Engineers. IEEE Recommended Practice for Software Acquisition, (IEEE Std 1062, 1998 Edition [R2002]). New York: IEEE, 1998.
4. Roback, Edward A. NIST Special Publication 800-23: Guidelines to Federal Organizations on Security Assurance and Acquisition/Use of Tested/Evaluated Products. Gaithersburg, MD: National Institute of Standards and Technology, 2000.
5. Redwine, Sam. Software Assurance: A Curriculum Guide to the Common Body of Knowledge to Produce, Acquire and Sustain Secure Software. U.S. Department of Homeland Security, 2007.
6. Duncan, Scott. IEEE Software and Systems Standards Committee (S2ESC) Meeting Report. San Diego, CA, February 13-15, 2014. Web. 10 Mar. 2014. <http://asq.org/software/about/chairfeb06-software.html>
7. International Standards Organization. ISO/IEC 12207:2008, Systems and software engineering -- Software life cycle processes. Geneva: ISO, 2008.
8. United States Government Accountability Office. IT Supply Chain: National Security-Related Agencies Need to Better Address Risks (GAO Report to Congressional Requesters). United States Government Accountability Office, 2012.
9. Woody, Carol and Ellison, Robert J. Improving Software Assurance. Published 1 April, 2010, revised 5 July 2013. Web. 10 Mar. 2014. <https://buildsecurityin.us-cert.gov/articles/knowledge/assurance-cases/improving-software-assurance>
10. Davis, Noopur. Secure Software Development Life Cycle Processes. Published 5 July 2006, revised 31 July 2013. Web. 10 Mar. 2014 <https://buildsecurityin.us-cert.gov/articles/knowledge/sdlc-process/secure-software-development-life-cycle-processes>
11. Hilburn, Thomas; Ardis, Mark; Johnson, Glenn; Kornecki, Andrew; and Mead, Nancy R. Software Assurance Competency Model, Software Engineering Institute (CMU/SEI-2013-TN-004). Pittsburgh: Carnegie Mellon Software Engineering Institute, 2013.

Evaluating Security Risks Using Mission Threads

Carol Woody, Ph.D., SEI
Christopher Alberts, SEI

Abstract. Mission threads describe operational process steps required to perform organizational functions. Researchers from the Carnegie Mellon Software Engineering Institute (SEI) explored the use of mission threads to connect desired operational capability to the underlying technology for analysis of system and software qualities such as security. The SEI has successfully applied this approach to develop an integration strategy for alert originators who plan to use the Wireless Emergency Alerts (WEA) system, a public alerting service offered by the Department of Homeland Security. This paper describes the approach using the WEA example and the value gained in applying mission thread analysis for security.

Importance of Systems of Systems

Everything we do these days involves system and software technology: cars, planes, banks, restaurants, stores, telephones, appliances, and entertainment rely extensively on technology. Much of this capability is supported by systems of systems—independent heterogeneous systems that work together to address desired functionality through complex network, data, and software interactions. The Wireless Emergency Alerts (WEA) service is a good example of a system of systems.

WEA enables local, tribal, state, territorial, and federal public safety officials to send geographically targeted text alerts to the public. The U.S. Department of Homeland Security Science and Technology (DHS S&T) Directorate partners with the Federal Emergency Management Agency (FEMA), the Federal Communication Commission, and commercial mobile service providers (CMSPs) to enhance public safety through the deployment of WEA, which permits emergency management organizations nationwide to submit alerts for public distribution by mobile carriers [1]. Alert originators can send three types of messages:

- Presidential Alert issued by the president of the United States to reach any region of the nation, or the nation as a whole
- Imminent Threat Alerts
- AMBER (America's Missing: Broadcast Emergency Response) Alerts

CMSPs relay these alerts from FEMA's Integrated Public Alert and Warning System (IPAWS) to mobile phones using cell broadcast technology, which does not get backlogged during times of emergency, unlike wireless voice and data services. Customers who own WEA-capable mobile phones will automatically receive these alerts during an emergency if they are located in the affected geographic area.

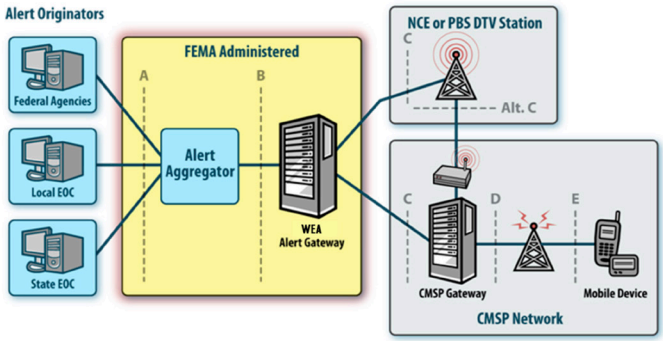


Figure 1: WEA system of systems

Alert originators already have extensive alert dissemination capability through the Emergency Alert System, highway signage systems, Internet websites, and telephone dialing systems, just to mention a few widely used alerting channels. The WEA system, which is diagramed in Fig. 1, would expand these options to mobile devices. FEMA established the message structure and the approvals needed to have the Alert Aggregator system accept messages for dissemination to mobile devices. Many alert originators plan to integrate this capability with systems already in place for other dissemination channels.

The Systems Engineering Handbook describes the following challenges for the development (and sustainment) of systems of systems [2]:

- Each participating system operates independently.
- Each participating system has its own update, enhancement, and replacement cycle.
- Overall performance of the desired functionality depends on how the various participating systems can interact, which is not always known in advance.
- Missing or conflicting standards can make the design of data exchanges among the participating systems complex and difficult to sustain.
- Each participating system has its own management, and the coordination of requirements, budget constraints, schedules, interfaces, and upgrades can have a major impact on the expected capability of the system of systems.
- Fuzzy boundaries can cause confusion and error; no one really owns the interface, but one of the participants needs to take leadership to ensure some level of shared understanding.
- The system of systems is never finished because as each system grows, expands, and ages, there is a constant opportunity and need for adjustment.

Public safety officials and alert recipients want to be able to rely on WEA capabilities and need to have confidence that the alerts are accurate and timely. Effective security is required to support this confidence. The risk that an attacker could create false alerts or cause valid alerts to be delayed, destroyed, or modified was identified as a critical issue. This could place the alert-originating organization's mission, and the lives and property of the citizens it serves, at risk.

DHS S&T asked a team of security experts at the SEI to research this problem and identify a means for evaluating WEA alert originator security concerns. The team selected mission thread analysis as a means for developing a view of the system of systems that could be used for evaluating security risks.

An analysis approach was needed to prepare alert originators to address the following critical security questions [3]:

- What do alert originators need to protect? Why does it need to be protected? What happens if it is not protected?
- What potential adverse consequences do alert originators need to prevent? At what cost? How much disruption can they stand before they take action?
- How do alert originators determine and effectively manage the residual risk?

In addition, alert originators needed to consider the local, state, and federal compliance standards that the organization must address to ensure that the planned choices for security also meet other mandated standards.

Preparing for Mission Thread Analysis

Drawing on SEI security expertise, initial questions were assembled to assist the alert originator in gathering information about the current environment and preparing for WEA (or any new technology capability). The alert-originating organization should compose answers to the following questions:

- What WEA capability do we plan to implement (types of alerts to issue, geographic regions to cover)?
- Can we expand existing capabilities to add WEA, or do we need new capabilities?
- Are good security practices in place for the current operational environment? Is there any history of security problems that can inform our planning?
- Will we use current resources (technology and people), or do we need to add resources?

Responses to these questions begin to frame the target operational context and the critical functionality that organizations must evaluate for operational security. Each organization will have a different mix of acquired technology and services, in-house development components, and existing operational capability into which the WEA capability will be woven. With the use of mission threads, responses to these questions can be described in a visually compelling form that management, system architects, system and software engineers, and stakeholders can share and refine.

A mission thread is an end-to-end set of steps that illustrate the technology and people resources needed to deliver expected behavior under a set of conditions and provide a basis for identifying and analyzing potential problems that could represent risks. For each mission step, the expected actions, outcomes, and assets are assembled. Confirmation that the components appropriately respond to expected operational use increases confidence that the system will function as intended even in the event of an attack [4].

Mission threads provide a means to identify and evaluate the ways, intentional or unintentional, that component system failures could occur and how these would impact the mission. A WEA example is provided to demonstrate how SEI used a mission thread to analyze security.

WEA Mission Thread Example

Mission thread analysis begins with the development of an operational mission thread. For WEA, typically 25 steps take place from the determination of the need for an alert to the receipt by cell phone owners:

1. First responder contacts local alerting authority via an approved device (cell phone, email, radio, etc.) to state that an event meets criteria for using WEA to issue, cancel, or update an alert and provides information for a message.
2. Local alerting authority (person) determines that a call or email from a first responder is legitimate.
3. Local alerting authority instructs Alert Origination System (AOS) operator to issue, cancel, or update an alert using information provided by a first responder.[1]
4. AOS operator logs on to the AOS.
5. AOS logon process activates auditing of the operator's session.
6. AOS operator enters alert, cancel, or update message.
7. AOS converts message to a format compliant with the Common Alerting Protocol (CAP, a WEA input standard).
8. CAP-compliant message is signed by a second person for local confirmation.
9. AOS transmits message to the IPAWS Open Platform for Emergency Networks (OPEN) Gateway.
10. IPAWS-OPEN Gateway verifies[2] message and returns status message to AOS.
11. AOS operator reads status message and responds as needed.
12. If the message was verified, IPAWS-OPEN Gateway sends message to WEA Alert Aggregator.
13. WEA Alert Aggregator verifies message and returns status to IPAWS-OPEN Gateway.
14. IPAWS-OPEN Gateway processes status and responds as needed.
15. WEA Alert Aggregator performs additional message processing as needed.
16. If the message was verified, WEA Alert Aggregator transmits alert to Federal Alert Gateway.
17. Federal Alert Gateway verifies message and returns status to WEA Alert Aggregator.
18. WEA Alert Aggregator processes status and responds as needed.
19. If the message was verified, Federal Alert Gateway converts message to CMAC (Commercial Mobile Alert for Interface C) format.
20. Federal Alert Gateway transmits message to CMSP gateway.
21. CMSP Gateway returns status to Federal Alert Gateway.
22. Federal Alert Gateway processes status and responds as needed.
23. CMSP Gateway sends message to CMSP Infrastructure.
24. CMSP Infrastructure sends message via broadcast to mobile devices in the designated area(s).
25. Mobile device users (recipients) receive the message.

Although many of the steps do not involve technology, they can still represent security risks to the mission. Mission thread analysis, unlike other techniques such as Failure Mode and Effect Analysis [5] allows consideration of the people and their interactions with technology in addition to the functioning of a system itself. Also, most security evaluations consider only individual system execution. However, effective operational execution of a mission must cross organizational and system boundaries to be complete. The use of mission thread analysis

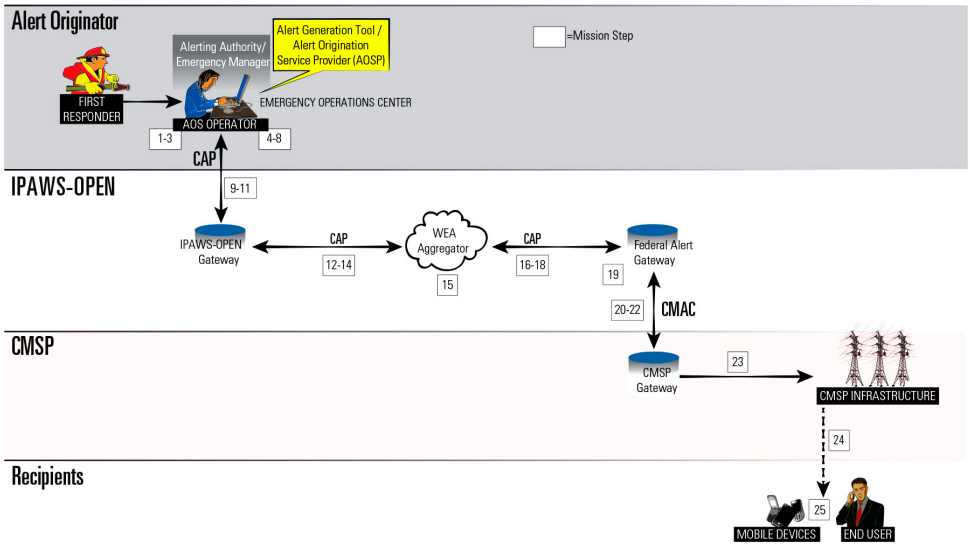


Figure 2: WEA mission thread diagram

Step #	Step	Assets	STRIDE Threat Identification Examples
4	AOS operator attempts to log on to the alert origination system.	• One person • Server (valid accounts/authentication information) • Logon procedure • Logon application • Username/password data in database • Communications between logon software, server, and AOS	S: Unidentified individual attempts to log on with AOS operator's information R: AOS operator denies having logged on I: Capture of logon info using key logger or packet sniffer D: AOS operator's account not registered or servers are down E: Successful logon by an unidentified and unauthorized individual
5	AOS logon activates auditing of the operator's session.	• Auditing application • Auditing procedure • Communications from accounts to auditing application • Local or remote storage	T: Logged entries added, deleted, or modified inappropriately I: Logged entries containing credential data are compromised D: Log full or server unavailable
6	AOS operator enters alert, cancel, or update message.	• One person • Alert scripts • Procedures for building scripts • GUI application • Communications between GUI application and alert generation software (including server and application)	T: Formatting errors produce incorrect message D: Scripts are unavailable or corrupted
7	AOS converts message to CAP-compliant format required by IPAWS.	• Conversion application	T: Data are changed between the AOS and the server D: Server is down
8	CAP-compliant message is signed by two people.	• Signature entry application • Signature validation application • Public/private key pair for every user	S: Digital signature is falsified R: User claims not to have signed D: Server goes down so keys cannot be distributed, or keys have expired and message cannot be sent
9	AOS transmits message to the IPAWS-OPEN Gateway.	• Application that securely connects to IPAWS • Information used to authenticate AOS and IPAWS	S: Falsified AOS CAP message or IPAWS gateway attacked and site is redirected T: Data within message are modified I: Message is not encrypted and credentials are visible D: IPAWS-OPEN Gateway is down

S: spoofing; **T**: tampering with data; **R**: repudiation; **I**: information disclosure; **D**: denial of service; **E**: elevation of privilege.

Table 1: STRIDE analysis for selected WEA mission thread steps

for security provides a way of confirming that each participating system is secure and does not represent a risk to all others involved in mission execution.

Fig. 2 provides a picture of the WEA mission thread and includes step numbers from the list to link each step to the appropriate system area. Successful completion requires flawless execution of four major system areas—alert originator, FEMA IPAWS system, CMSPs, and cell phone recipients—each shown in a row of the figure. Each area operates independently, connected only through the transmission of an alert.

WEA Security Analysis

Using the mission thread illustrated in Fig. 2, potential security concerns can be identified through possible security threats. For the WEA example, SEI selected the STRIDE Threat Method for threat evaluation. STRIDE, developed by Microsoft, considers six typical categories of security concerns: spoofing, tampering with data, repudiations, information disclosure, denial of service, and elevation of privilege [6]. The name of this threat method is derived from the first letter of each security concern [7]. As an illustration of how STRIDE can be applied, focus on Steps 4–9 of the mission thread, which represent the transition across two major system areas from the alert originator to the FEMA system and provide an opportunity for mission failure if interaction between the system areas is not secure. Table 1 shows the result of the STRIDE analysis on the selected steps.

For each step, the technology assets critical to step execution were analyzed to determine ways that STRIDE threats can compromise each asset used in that step [7]. Security and software experts as well as individuals familiar with the operational mission need to participate in this portion of the analysis. The security and software experts have an understanding of what can go wrong and the potential impact of each possible failure on the analysis. Those knowledgeable about the operational execution can ensure that the scenarios are realistic and valid. Available documentation can provide a start for the development of the mission threads, but there is a tendency to document the desired operational environment and not the real one. Effective security risk analysis requires access to realistic operational information.

Based on this input, security experts (individuals with operational security training and experience) identified at least two security risks that could lead to mission failure:

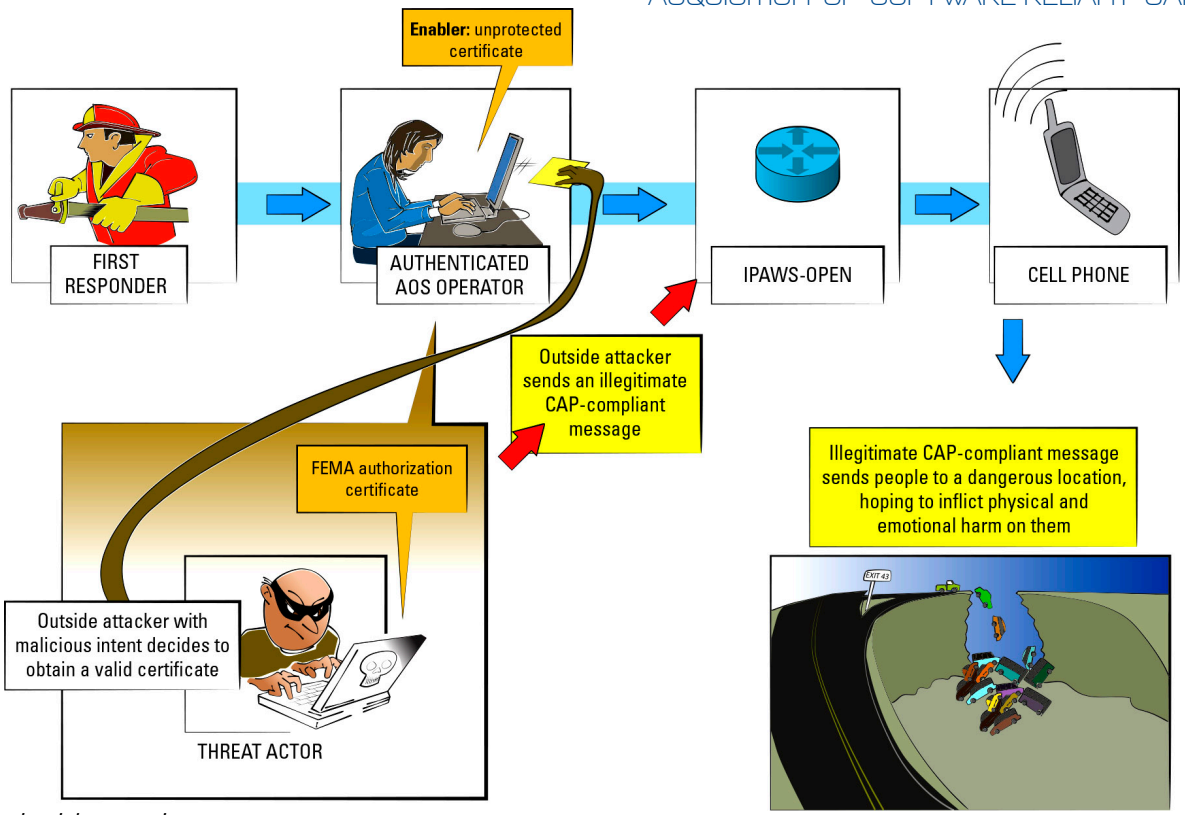


Figure 3: Security risk scenario

1. authentication of the individual using the AOS in Step 4
2. validation and protection of the digital signatures applied to the alert approved for submission to the Alert Aggregator in Step 8

To analyze these risks in greater detail and help alert originators understand how a security risk could materialize, mission threads for each specific risk were assembled. Fig. 3 provides a picture of the risk scenario that describes the second security risk (validity of the digital signature) noted from the analysis of the WEA operational mission thread. The following paragraphs provide a dialogue that describes ways in which the security threat could materialize and why the alert origination organization should consider possible mitigations.

An outside attacker with malicious intent decides to obtain a valid certificate and use it to send an illegitimate CAP-compliant message. The attacker's goal is to send people to a dangerous location, hoping to inflict physical and emotional harm on them. The key to this attack is capturing a valid certificate from an alert originator. The attacker develops two strategies for capturing a valid certificate. The first strategy targets an alert originator directly. The second strategy focuses on AOS vendors. Targeting a vendor could be a particularly fruitful strategy for the attacker. The number of vendors that provide AOS software is small. As a result, each vendor controls a large number of certificates. A compromised vendor could provide an attacker with many potential organizations to target.

No matter which strategy is pursued, the attacker looks for vulnerabilities (i.e., weaknesses) in technologies or procedures that can be exploited. For example, the attacker will try to find vulnerabilities that expose certificates to exploit, such as

- unmonitored access to certificates
- lack of encryption controls for certificates during transit and storage
- lack of role-based access to certificates

The attacker might also explore social engineering techniques to obtain a certificate. Here, the attacker attempts to manipulate someone from the alert originator or vendor organization into providing access to a legitimate certificate or to get information that will be useful in the attacker's quest to get a certificate.

Obtaining a certificate is not a simple endeavor. The attacker has to be sufficiently motivated and skilled to achieve this interim goal. However, once this part of the scenario is complete, the attacker is well positioned to send an illegitimate CAP-compliant message. The attacker has easy access to publicly documented information defining how to construct CAP-compliant messages.

The attacker's goal in this risk is to send people to a location that will put them in harm's way. To maximize the impact, the attacker takes advantage of an impending event (e.g., weather event, natural disaster). Because people likely will verify WEA messages through other channels, synchronizing the attack with an impending event makes it more likely that people will follow the attacker's instructions. This scenario could produce catastrophic consequences, depending on the severity of the event with which the attack is linked.

Through the use of mission thread analysis, security expertise can be integrated with operational execution to fully describe and analyze operational security risk situations. While there may be many variations of operational execution, an exhaustive study of all options is not necessary. Building a representative example that provides a detailed view of a real operational mission from start to finish has proven to be of value for security risk identification.

Conclusion

The process of developing a well-articulated mission thread that operational and security experts can share and analyze provides an opportunity to uncover missing or incomplete requirements as well as differences in understanding, faulty assumptions, and interactions across system and software boundaries that could contribute to security concerns and potential failure [4].

The mission thread analysis connects each mission step with the technology and human assets needed to execute that step and provides a framework to link potential security threats directly to mission execution. Mission thread diagrams and tables assemble information in a structure that can be readily reviewed and validated by operational and technology experts from various disciplines including acquisition, development, and operational support. Mission thread security analysis can be an effective tool for improved identification of security risks to increase confidence that the system of systems will function with appropriate operational security.

Acknowledgment

This material is based upon work funded and supported by Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013 and 252.227-7013 Alternate I.

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 Department of Homeland Security or the United States Department of Defense.

ABOUT THE AUTHORS

Dr. Carol Woody has been a senior member of the technical staff at the Software Engineering Institute, Carnegie Mellon University since 2001. Currently she is the technical lead of the cyber security engineering team whose research focuses on building capabilities in defining, acquiring, developing, measuring, managing, and sustaining secure software for highly complex networked systems as well as systems of systems.

Carnegie Mellon University
Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213
Phone: 412-268-9137
E-mail: cwoody@cert.org

Christopher Alberts is a Principal Engineer in the CERT® Program at the Software Engineering Institute, where he leads applied research projects in software assurance and cyber security. His research interests include risk analysis, measurement, and assessment. He has published two books and over 35 technical reports and articles. Alberts has BS and ME degrees in engineering from Carnegie Mellon University. Prior to the SEI, he worked at Carnegie Mellon Research Institute and AT&T Bell Laboratories.

Carnegie Mellon University
Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213
Phone: 412-268-3045
E-mail: cja@cert.org

REFERENCES

1. United States. Federal Emergency Management Agency. "Wireless Emergency Alerts." Federal Emergency Management Agency. FEMA, 20 June 2013. Web. 7 June 2013. <http://www.fema.gov/wireless-emergency-alerts>.
2. Haskins, C., ed. Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities. Version 3.2. Revised by K. Forsberg, M. Krueger, and R. Hamelin. San Diego: International Council on Systems Engineering (INCOSE), 2010. Print.
3. Allen, J., S. Barnum, R. Ellison, G. McGraw, and N. Mead. Software Security Engineering: A Guide for Project Managers. Upper Saddle River: Addison-Wesley, 2008. Print.
4. Ellison, R., J. Goodenough, C. Weinstock, and C. Woody. Survivability Assurance for System of Systems (CMU/SEI-2008-TR-008). Pittsburgh: Software Engineering Institute, Carnegie Mellon University, May 2008. 7 June 2013. <http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=8693>. Web.
5. Stamatis, D. H. Failure Mode Effect Analysis: FEMA from Theory to Execution. 2nd ed. Milwaukee: ASQ Quality Press, 2003. Print.
6. Microsoft. "The STRIDE Threat Model." Microsoft Developer Network. Microsoft, 2005. 7 June 2013. <http://msdn.microsoft.com/en-US/library/ee823878%28v=cs.20%29.aspx>. Web.
7. Howard, M., and S. Lipner. The Security Development Life Cycle. Redmond: Microsoft Press, 2006. Print.

NOTES

1. In some cases, the alerting authority and the AOS operator may be the same person.
2. In this list of steps, message verification includes authentication and ensuring that the message is correctly formatted.

The Necessity of Intended Use Specification for Successful Modeling and Simulation of a System-of-Systems

Charles H. Piersall III, P.E., Missile Defense Agency
Franklin E. Grange, Ph.D., ISSAC Corporation

Abstract. Informal and casual consideration of Intended Use in Modeling and Simulation practice can pose programmatic risks in acquisition, especially in system-of-system contexts, such as the Ballistic Missile Defense System. Leveraging lessons learned from intense reliance on system-of-system simulations, the Missile Defense Agency is formalizing specification of Intended Uses to mitigate those risks and to improve effectiveness and affordability of the Agency's diverse simulations.

Introduction

The objective of this article is to present an approach in development by the Missile Defense Agency (MDA) for specifying the Intended Use (IU) in Modeling and Simulation (M&S) applied in a system-of-systems (SoS) context. The MDA's mission is to develop, test, and field an integrated, layered, Ballistic Missile Defense System (BMDS) to defend the United States, its deployed forces, allies, and friends against all ranges of enemy ballistic missiles in all phases of flight. The BMDS is a SoS.

Many practical and legal constraints force SoS acquisition organizations like MDA to depend on M&S for diverse needs, such as assessment, training, exercise and concept development [1]. Typically, SoS M&S relies heavily on the compositions (sometimes called "federations") of legacy M&S independently developed for the constituent systems of the SoS. Disparities among the original IUs of the constituent systems' simulations complicate or even confound a SoS M&S IU, which itself usually differs from the constituent simulations' original IUs.

Suitability of BMDS M&S for an IU is at the core of an *Accreditation* process [2] incorporating formal analysis of risks to acquisition and warfighting from using a specific simulation supporting essential decisions, such as deployment. Thus, MDA developed a SoS M&S IU approach to improve outcomes of its BMDS M&S systems engineering and accreditation processes. MDA's SoS M&S IU approach has features and implications useful for SoS M&S engineering in other M&S-reliant domains, such as Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR) and space exploration.

In the following sections, we consider first what M&S IU really means, and how widespread usage of the term without any common definition creates practical difficulties—especially for SoS M&S. We present our lessons learned about programmatic risks that M&S IU misspecification (or outright ignorance) can create. From those lessons emerged MDA's new practice for BMDS M&S IU specification, which focuses on the needs of the stakeholder analyst as a consumer of M&S. A subsequent case study shows the implications of ignoring a SoS M&S IU specification. Finally, we present our vision for migrating MDA's BMDS M&S IU specification to a model-centric systems engineering environment that can increase the reliability and timeliness of BMDS M&S development and use.

"Everybody Knows What M&S 'Intended Use' Means…Right?"

Numerous M&S authorities describe the *importance* of specifying an M&S IU but neglect to *define* "IU" explicitly (e.g., [3], [4]). For example, [5] presents the following circular definition: "An M&S application's Intended Use refers to the explicitly and clearly defined purpose for which the application is intended for use." Representative of DoD interest in M&S accreditation for affordable reuse, MIL-STD 3022, "Documentation of Verification, Validation and Accreditation (VV&A) for Models and Simulations," also does not define "IU" but does require documentation of "the problem to be addressed by the M&S and its associated data, including the system or process being represented and the role it plays in the overall program" [6].

Notwithstanding arguments about inadequacies of M&S requirements engineering, we observe that effective practitioners invariably develop *new* M&S with one or more definite purposes in mind. Thus, for new M&S development, an unambiguous, testable definition of M&S "IU" might seem unnecessary. Many authorities thus imply that M&S "IU" is an important, essential but undefinable concept akin to "point," "line," or "plane" in geometry.[1]

Working in a SoS M&S enterprise emphasizing affordability and simulation asset reuse, we drew from our "lessons learned" the conclusion that M&S "IU" *must* be formally defined so that IU specifications may be compared and tested for clarity, completeness and compatibility. The following review of those lessons learned and programmatic risks we encountered motivates our recommendation for formalizing SoS M&S IU definition and specification in a SoS domain.

"Wait a minute…we're not talking about the same IU?"

Applying M&S to the BMDS, a SoS, we recognized that BMDS M&S is itself a SoS engineering endeavor. In 2006, the MDA established the BMDS M&S Program including a goal of assessing BMDS capabilities with end-to-end, constructive simulations.[2] High-resolution, "engineering-grade" M&S previously and independently developed for Element program acquisitions (i.e., the Elements' overarching IUs) comprise the BMDS simulations of Element interceptors, sensors and battle management interoperating as a BMD SoS. At first, we expected that composing an acquisition-focused BMD SoS M&S from acquisition-focused Element M&S constituents should be hard but straightforward and predictable work. We envisioned that disciplined, collaborative requirements engineering across both the BMDS M&S and the Element M&S was necessary and sufficient for successful implementation of the BMD SoS M&S. What we encountered were conflicts among the Element M&S IUs, as well as Element IU conflicts with the BMDS M&S IU.

- The mission contexts of Element M&S IUs vary substantially; some, such as the BMDS Command, Control, Battle Management and Communications (C2BMC) Element, exist solely for

a BMDS mission, while others, such as Aegis BMD with fleet defense, have standalone or additional non-BMD missions

- Element M&S IUs are rarely "Legos™," as each Element's acquisition engineering M&S IU may be very specific to an independent Element program; e.g., enemy missile threats to which an Element is engineered might be very specific, and the M&S IU will also be as specific
- In most cases the Element programs, before developing their M&S, did not receive M&S composability requirements to enable future BMDS M&S IU; in result, Element M&S architectures complicate integration and interoperability; an example is organic, tightly integrated Radar Cross Section (RCS) modeling of an enemy missile threat by two Elements that the BMDS M&S IU requires to have a common, consistent RCS measurement (e.g., adjusted, of course, for sensor characteristics, viewing angles, and threat orientation).

These kinds of M&S IU conflicts embodied risks that too often we saw become programmatic issues. Preparing for BMDS M&S integration, the Element M&S developers undertook reengineering their simulations for interoperation with others' simulations; in their original, standalone versions, the Element simulations represented the effects of interactions happening, but not the full interactive process (e.g., a BMDS battle manager function assigning a missile intercept to one tracking Element versus another). Discovery of the interoperability and integration requirements was an unexpectedly prolonged trial-and-error process, and integration lead times to prepare BMDS assessments were neither predictable nor affordably short to sustain BMDS analysts' desired work tempos.

In the early BMDS M&S IU, the BMDS architecture was far more loosely coupled than at present, and omitting some Element interactions or common, consistent threat or environment modeling did not necessarily thwart the overall BMDS M&S IU. Nevertheless, root causes of errors discovered during runs-for-record often traced at least in part to previously unrecognized IU conflicts among the Element M&S and the BMDS M&S. As the near-future BMDS architecture becomes more tightly coupled, these kinds of IU-based errors become much harder and costlier to correct.

Like C4ISR, planetary space missions, and many other SoS', comprehensive testing of the BMDS is generally so unaffordable, impractical, unsafe or illegal (by treaty) that M&S is the only means to provide estimates of the capabilities and limitations of the SoS. However, while nearly always safe and legal, M&S must be affordable, practical and timely. Clarifying, understanding, and aligning M&S IUs mitigates at least *some* cost, schedule and scope risks of BMDS M&S.

Emerging MDA Practice for Specifying M&S IU in BMDS M&S

An engineering process of selecting a legacy M&S application/tool or developing a new M&S application/tool begins with creating a clear statement of the M&S IU and a description of the associated M&S Capability Needs *from the perspectives of stakeholder analysts* (see Section 2 of [9]).[3] For the inferences stakeholders want to draw about the BMDS, stakeholder analysts need particular kinds of M&S outputs, and typically require results from particular kinds of M&S runs or experimental conditions. These M&S Capability Needs encompass the set of *analysis capabilities* that M&S should *enable* in order to adequately achieve its IU (e.g., "The BMDS M&S shall provide output X under experimental conditions Y to enable the stakeholder analyst to calculate BMDS metric Z").

The position of the IU within the Systems Engineering Requirements Process is shown in Figure 1. The depicted process pertains to both unitary systems and SoS. For both legacy and new M&S, the stakeholders are responsible to articulate a set of M&S Capability Needs that comprise essential parts of the M&S IU. If the stakeholder who utilizes results of an M&S tool misunderstands what IU drove its development, the stakeholder analyst is at risk of misconstruing the M&S results.

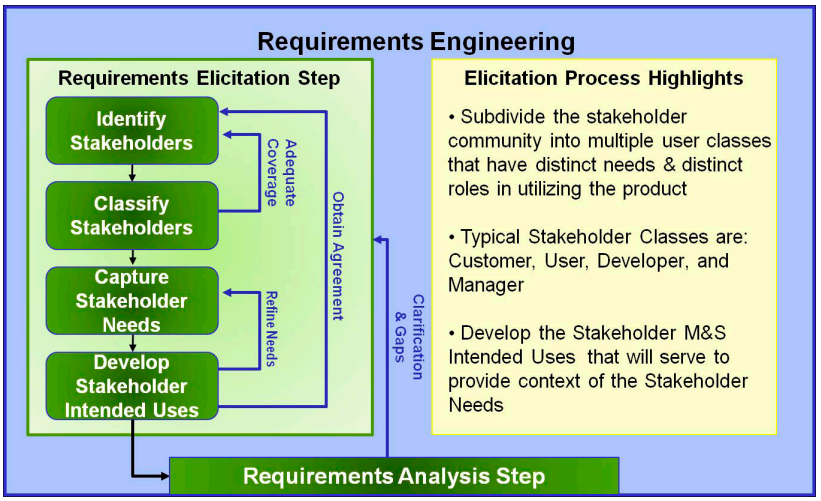


Figure 1: Role of Intended Use Specification within M&S Requirements Engineering

MDA currently requires several parts for a well-defined M&S IU. From the stakeholder analyst's perspective, an M&S IU is not a capability, solution or implementation, but part of a clear, complete analysis problem statement. MDA is establishing a standard IU specification that contains the following components:

1. Title—For ease of communication and reference among stakeholders and developers
2. Description—Five essential elements:

- Narrative identification of the analysis problem
- Narrative description of what stakeholder analyst tasks or processes the IU supports
- Narrative enumeration of specific simulation outputs the stakeholder analyst will use in the analyst's processes; should identify products or documents that the IU has inputs to or creates directly
- Narrative description of special conditions or experimental designs under which the simulation should run to produce the analyst's needed specific simulation outputs; should include any needed simulation calibration methods and associated data
- Narrative description of "controls or governances" identifying documents or processes to which the IU conforms, such as stakeholder test objectives memorandum or a test concepts of operations (CONOPS)

3. Key Attributes—For a specific M&S domain, standard aspects and values that stakeholders and developers agree clarify understanding about the M&S IU; for MDA's BMDS M&S IU, these include the following:[4]
   - Focus: From narrow consideration of a component (e.g., specific radar) to broad scope of the end-to-end BMDS
   - Epoch: Current or future version of the BMDS Architecture represented
   - Simulation Type: Constructive, Virtual, Live [8]
   - Fidelity:[5] Degree of simulation faithfulness
   - Uncertainty Quantification: Methods for representing uncertainties and propagating them to simulation outcomes
   - Tactical Interoperability: Accuracy by which the M&S environment recreates tactical Element or Component external interfaces
   - Stimulation Interface: Level of detail contained within stimulation data distributed through the system; includes threat, modeled communication networks, environment, lethality (interceptor-target interaction physics)
   - Operator Screens: Degree to which user interfaces can provide an Operator-in-the-loop with a realistic training or exercise experience
4. Identified Stakeholders—Listing of stakeholders who either "own" the M&S IU or make decisions with the simulation results

A Case Study of Catastrophic Risk from Ignoring SoS M&S IUs

The Crandall Canyon Mine case study reveals how fatal consequences may result from inadequate consideration of M&S IUs in safety engineering.

In August 2007, the Crandall Canyon Mine in Utah collapsed and trapped six workers. Three more workers died in an additional collapse during rescue operations that ultimately failed to recover the original six victims' bodies. Investigating the disaster, the US Mine Safety and Health Administration (MSHA) reviewed not only mine conditions and operations practices, but also the engineering design of the mine [10]. MSHA determined that improper engineering analysis with two stress simulations was one of the root causes of the collapses killing both the miners and the rescuers.

Mine engineering analysis considers both geologic conditions and mining methods. The Crandall Canyon Mine employed methods of "longwall mining" and "retreat mining" not only to maximize coal recovery, but also to ensure mine structural integrity for the safety of workers and equipment. As longwall mining proceeds through coal beds, pillars of valuable coal left by mining operations support the roof in the mined space. When advance through the space completes, retreat mining involves partial or complete removal of some pillars as workers withdraw equipment back from the mined space. While pillar recovery may trigger some roof collapse, proper retreat mining ensures that roof collapse occurs safely and only in the immediate area of a pillar being extracted. Mine design analysis considers such factors as the mechanical properties of the pillar material, the depth of the mining space and the condition of the prospective roof to establish the needed size and spacing of pillars for productive but safe mining.

MSHA found that undersized pillars contributed significantly to the Crandall Canyon Mine collapse. As retreat mining proceeded, excessive rising stress on remaining pillars finally triggered one pillar failure that started a ripple of load shocks overstressing and collapsing other pillars. MSHA focused specifically on the use of two mine engineering simulations, "Stress and Displacement Calculations" (LaModel) and "Analysis of Retreat Mining Pillar Stability" (ARMPS). While MSHA agreed with the suitability of both models for the mine engineering IU, MSHA found that the Crandall Canyon Mine's engineering contractor failed to apply LaModel and ARMPS *together* (i.e., effectively as a SoS M&S) in accordance with their M&S IUs, which required their initial calibration with the same real-world geologic data from the Crandall Canyon Mine.

This case study provides a sobering lesson that *the associated data and stakeholder analyst processes are an essential part of a SoS M&S IU, and they must be applied consistently to the M&S components involved in the SoS M&S.* For large, complex SoS' like C4ISR systems or the BMDS, document-centric M&S IU analysis and specification rely strongly on systems engineers' broad understanding of the entire SoS scope. The case study shows how engineers may fail to specify and apply an M&S IU for even a much narrower scope than the BMDS. The following section outlines MDA's M&S engineering initiatives to increase the reliability and timeliness of SoS M&S IU analysis and specification through automation.

A Hopeful Future: Formalized M&S IUs for Model-Based Systems Engineering

MDA's BMDS M&S Program plans to update and incorporate an M&S IU specification into Model-Based Systems Engineering (MBSE) [12] of MDA's BMDS M&S. Implementation of MBSE for BMDS M&S represents a transition from document-based systems engineering to an integrated Systems Modeling Language (SysML) models [13] of requirements, structure, behavior and parametrics (algorithms, quantities-of-interest, units-of-measure, etc.). MDA is currently developing the MBSE infrastructure of engineering processes, methods, standards, training, staffs and tools for Model-Based *SoS* Engineering (MBSoSE) of BMDS M&S. Clear, complete M&S IUs are essential to sustained MBSoSE of BMDS M&S to support decisions about the best M&S components to use in standalone or compositions to fulfill stakeholder needs.

Some benefits MDA anticipates from MBSoSE including M&S IUs are the following:

- A repository of M&S IU information updated as M&S constituents evolve
- Automated traceability and allocation of M&S IU information to stakeholder requirements; M&S logical standalone or composition design; M&S behavior; and M&S parametrics
- Automated design verification
- Automated detection and analysis of M&S IU errors, contradictions or gaps in addressing stakeholder needs, derived requirements and logical design

- Clear, consistent, traceable, measurable, testable requirements allocated to the Element and overall BMDS M&S developer organizations (e.g., to put on contract)

Complementing MDA's transition to MBSoSE with M&S IUs are changes in BMDS M&S governance and culture that are not yet fully defined. Realizing some of the above benefits also requires evolution in MBSE tool technology and the SysML language.[6] We anticipate those technical changes will occur concurrently with MDA's BMDS M&S governance and culture evolution. In the future we hope to report on best practices and lessons learned from MDA's transition to MBSoSE of BMDS M&S.

Summary

MDA has formalized specification of BMDS M&S IU to mitigate programmatic risks and to improve affordability and outcomes of M&S-based acquisition activities. Lessons learned in BMDS M&S informed MDA's specification of a BMDS M&S IU standard focused on stakeholder analysts as the consumer of M&S results. Disciplined use of MDA's SoS M&S IU approach helps avoid costly or even catastrophic risks of misapplying M&S. Though the original SoS M&S IU specification is document-centric, MDA anticipates its transition to future model-centric processes with MBSoSE automation increasing both affordability and the tempo of M&S-based acquisition activities.

ABOUT THE AUTHORS

Charles H. Piersall III, P.E. is the Deputy Associate Director for Modeling & Simulation Operations at the Missile Defense Agency (MDA). He earned a Bachelor's Degree in Marine Engineering from the US Naval Academy and a Master's Degree in Applied Science from the Naval Postgraduate School. He is a licensed Mechanical Engineer and qualified Navy Nuclear Engineer. He is currently serving as the President of the National Society of Professional Engineers (NSPE) Colorado.

Mr. Piersall has extensive experience in leading the development of complex Ballistic Missile Defense (BMD) models and simulations. He pioneered the use of Agile Software Development methods for multiple models and simulations within the MDA. He also led the development and implementation of a complex end-to-end simulation of the Ballistic Missile Defense System that was used on Capitol Hill to inform US Law Makers of the complexities of Missile Defense.

Phone: 719-721-7438
E-mail: chuck.piersall@mda.mil

ABOUT THE AUTHORS

Dr. Franklin E. Grange is a Scientist at ISSAC Corporation in Colorado Springs, CO. His involvement with BMDS M&S systems engineering, development and application spans 29 years. His career in simulation and analytics has addressed problems in defense, intelligence, high-tech, natural resources, banking, telecommunications, manufacturing and supply chain. He holds PhD and MS degrees in Mineral Economics/Operations Research from the Colorado School of Mines (CSM), and a BS in Petroleum Refining Engineering also from CSM.

Phone: 719-721-2228
E-mail: franklin.grange.ctr@mda.mil

REFERENCES

1. Department of Defense. Ballistic Missile Defense Review Report. Washington, DC, 2010.
2. Department of Defense (DoD) Instruction 5000.61, DoD M&S Verification, Validation, and Accreditation (VV&A). 9 Dec 2009.
3. Turnitsa, Charles, Jose J. Padilla, and Andreas Tolk. "Ontology for Modeling and Simulation." Proceedings of the 2010 Winter Simulation Conference. Baltimore. IEEE, 2010. 643-51.
4. Cook, David A., and James M. Skinner. "How to Perform Credible Verification, Validation, and Accreditation for Modeling and Simulation." Crosstalk, the Journal of Defense Software Engineering (2005): 20-24.
5. Balci, Osman. "How to Successfully Conduct Large-Scale Modeling and Simulation Projects." Proceedings of the 2011 Winter Simulation Conference. Phoenix. IEEE, 2011. 176-182.
6. Department of Defense (DoD) Standard Practice. MIL-STD-3022 with Change 1, Documentation of Verification, Validation, and Accreditation (VV&A) for Models & Simulations. 5 April 2012.
7. "Primitive Notion." Wikipedia. Wikimedia Foundation, 23 Feb. 2014. Web. 20 May 2014.
8. Page, Ernest H., and Roger Smith. "Introduction to Military Training Simulation: A Guide for Discrete Event Simulationists." Proceedings of the 1998 Winter Simulation Conference. Washington, DC, IEEE, 1998. 53-60.
9. Caine, Lisa, Bobby Hartway, Danny Thomas, and Joe Hale. "Incremental Quantification of VV&A "Levels" for Integrated Management of Modeling & Simulation Tools." Proceedings of the Huntsville Simulation Conference HSC 2006. The Society for Modeling and Simulation International. 17-19 Oct. 2006.
10. U.S. Mine Safety and Health Administration, "Report of Investigation: Underground Coal Mine Fatal Underground Coal Burst Accidents, August 6 and 16, 2007, Crandall Canyon Mine, Genwal Resources Inc, Huntington, Emery County, Utah," ID No. 42-01715, July 10, 2008.
11. Gross, David C. Report from the Fidelity Implementation Study Group. Rep. no. SISO-REF-002-1999. Orlando: Simulation Interoperability Standards Organization, 1999.
12. Weilkiens, Tim. Systems Engineering with SysML/UML: Modeling, Analysis, Design. Amsterdam: Morgan Kaufmann OMG/Elsevier, 2007.
13. Friedenthal, Sanford, Alan Moore, and Rick Steiner. A Practical Guide to SysML: The Systems Modeling Language. Waltham, MA: Morgan Kaufmann, 2012.
14. Zeigler, Bernard P., and Phillip E. Hammonds. Modeling & Simulation-based Data Engineering: Introducing Pragmatics into Ontologies for Net-centric Information Exchange. Oxford: Academic, 2007.
15. Tolk, Andreas, Saikou Y. Diallo, and Jose J. Padilla. "Semiotics, Entropy and Interoperability of Simulation Systems–Mathematical Foundations of M&S Standardization." Proceedings of the 2012 Winter Simulation Conference. Berlin, IEEE, 2012.

NOTES

1. "Point," "line" and "plane" are examples of a primitive notion, a concept undefined by previously defined concepts, and often motivated by intuition, common sense or everyday experience [7]. We find M&S practitioners inclined to specify IUs as "primitive notions" about which all M&S stakeholders should agree. In System-of-Systems (SoS) M&S engineering, diverse stakeholders using the same words with different meanings can disagree about an M&S IU specification without recognizing any disagreement. Thus, for success of M&S SoS engineering, we argue that M&S "IU" in SoS cannot be a primitive notion like those basic ideas of geometry.
2. A "constructive simulation" is a pure software implementation of a model of the system of interest of "real" (or envisioned) people, hardware, software, other facilities, inputs, outputs and processes [8]. While real simulation operators stimulate (i.e., make inputs, initiate runs) to such simulations, the operators are not otherwise involved in determining the simulation outputs (e.g., operators do not interact as "players" or "trainees" with the simulation during a run). Constructive simulations typically support acquisition engineering activities.
3. In DoD practice, stakeholder analysts are typically the users of the results of simulation runs (experiments). Analysts are often not M&S developers and maintainers. In the experience of one of the present authors, commercial practice tends to combine the roles of stakeholder analyst and M&S developer. Commercial practice is also infrequently concerned with simulation reuse or simulation compositions for SoS engineering.
4. MDA has not publicly released Key Attribute values for M&S IU.
5. "The degree to which a model or simulation reproduces the state and behavior of a real world object or the perception of a real world object, feature, condition, or chosen standard in a measurable or perceivable manner; a measure of the realism of a model or simulation; faithfulness" [11].
6. "Pragmatics" is a term used in formal logics of Ontologies and the Semantic Web. Both Zeigler, et al [14], and Tolk, et al [15], provide a formal mathematical definition of "IU" not as a "primitive notion," but as a pragmatic in terms of other concepts from Ontologies. The benefit of defining an IU as a mathematical pragmatic is machine readability (e.g., in a future, more formal SysML grammar, or with the Web Ontology Language, OWL) and machine decidability about the congruence of an M&S IU to stakeholder needs (e.g., in SysML design verification of an M&S composition).

The future MBSE capabilities will slash the lead time for SoS M&S systems engineering and integration, and increase the tempo of M&S-based analyses of large, complex SoSs.

Software Assurance, Trustworthiness, and Rigor

Don O'Neill, Independent Consultant

Abstract. Trustworthiness requires a commitment to rigor in both software production and its verification. No soft skill, rigor has a hard edge. In an environment of neglect and unmet need, the introduction of rigor would be a game changer, one that requires both cultural change and engineering know how with pass/fail training certification.

Introduction

Software Assurance only has meaning in the context of trustworthiness, that is, worthy of being trusted to fulfill the critical requirements needed for a particular software component, system, or system of systems [1]. Software Assurance demands two capabilities associated with trustworthiness, the capability to produce trustworthy software products and the capability to verify that software products are trustworthy. Each depends on engineering and technology rigorously applied.

The kernel of the layered defense approach to Software Assurance is Build Security In and Structured Programming with its rigorous and provably correct use of zero and one predicate prime programs along with proper programs composed of multiple prime programs limited to single entry and single exit [2].

The layered defense approach described here includes Defense in Depth, Build Security In, Risk Management Calculation, and Resilience processes, engineering, and tool automation dependent on an elevated state of software engineering rigor and precision [3]. There is much room for improvement in the Software Assurance methods and practices that assure such rigor.

Software Assurance Definition

From DOD ASD (R&E) the term "software assurance", or "SwA", means the level of confidence that software functions as intended, and only as intended, and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software, throughout the lifecycle. According to the Department of Homeland Security, software assurance spans:

1. **Trustworthiness -** No exploitable vulnerabilities exist, either maliciously or unintentionally inserted.
2. **Predictable Execution -** Justifiable confidence that software, when executed, functions as intended.
3. **Conformance -** Planned and systematic set of multi-disciplinary activities that ensure software processes and products conform to requirements, standards, and procedures.

Accordingly then, the objectives of this article are to be better able to pinpoint the factors and sources of risk involved in achieving Software Assurance, to identify concrete Software Assurance objectives involved and consequential outcomes sought in meeting each of the goals associated with software production and its verification, and to visualize the operations of Software Assurance through the use of assurance assertions and indicators. Concrete Software Assurance objectives with consequential outcomes include components that are provably correct; demonstrably free of weaknesses and Cyber Security vulnerabilities; and demonstrably free of improper proprietary information, copyrighted material, and trade secrets.

Background

On July 31, 2007, the Software Assurance State of the Art Report [4] described the emerging activities, approaches, and technologies thought to be the foundation of the discipline of software security. While the report mentioned Inspections, Correctness by Construction, Static Analysis, Function Extraction for Malicious Code (FX/MC), Common Weakness Enumeration (CWE), and Standard of Excellence Product Checklists, it did not include Structured Programming, Clean Room Software Engineering, or Statistical Testing. Build Security In depends on the adoption of all of these methods. Also not mentioned in SOAR were Technical Debt and Build Security In except for reference to a portal name. Importantly, SOAR identified the need for security extensions to the CMMI®, so far only discussed and not acted upon. Resilience was mentioned in the context of resistance to and tolerance of attack, fault tolerance, and system failure recovery but not in the context of anticipating, avoiding, withstanding, mitigating, and recovering from the effects of adversity whether natural or manmade under all circumstances of use including both systems and systems of systems.

Situation is Dire

The current state of Software Assurance is dire and stems from a combination of neglect and unmet need [1,5]. The future state of Software Assurance needs to feature smart and trusted methods and metrics including Software Assurance risk calculation that is precise, accurate, timely, and complete.

Some unmet need is due to lack of adoption and some, lack of technology. Lack of adoption falls on both industry practitioners and government acquisition specialists. Neglect is on glaring display in the intentional practice of Technical Debt spurred on by a misguided culture of acceptance [6, 7]. Furthermore, the shortfall in STEM workforce yields a persistent level of unintentional neglect based on ignorance. Beyond Software Assurance adoption, the focus areas most in need of attention are Build Security In, Resiliency, Advanced Technology, and renovation of the CMMI to address Software Assurance.

The improper use of proprietary software involving proprietary information, copyrighted material, and trade secrets increasingly goes undetected. Uncovering such use and detecting specific instances is a technical challenge, one that usually requires full and ready access to the Dirty System source code for best results as well as the wherewithal and means to express the proprietary information, copyrighted material, and trade secrets in a precise, rigorous, and trusted abstract manner suitable for computer searching and comparison [8].

Focus Areas	Known Practice	Underutilized Practice	Underutilized Technology	Advanced Technology Needed
Technical Debt	Discontinue this practice since it intentionally fosters vulnerabilities and neglect			
CMMI	Excellent platform for achieving widespread adoption yet incomplete with respect to Software Assurance, Cyber Security		Software Assurance, Cyber Security	
Defense In Depth	Encryption, identity management, access control, authorization management, accountability management, incident management, configuration management, and security assurance operations	Encryption, authorization management, security assurance operations	Three factor authentication: what you are, what you have, what you know	Trusted encryption advances needed
Build Security In	Structured Programming with emphasis on proof of correctness and correctness by construction	Emphasis on proof of correctness and correctness by construction		
	Cleanroom Software Engineering and Statistical Testing	Cleanroom Software Engineering	Statistical Testing	
	Software Inspections Process with well defined Standard of Excellence for SDLC software artifacts	Well defined Standard of Excellence: completeness, correctness, style, rules of construction, multiple views		
	Static Analysis		Approximate Matching to disambiguate multi-vendor tool results	Static Analysis commonality and interoperable tool vendor results post processing
	Function Extraction	Function Extraction	Function Extraction applied to large programs and systems	All languages and instruction set architectures
	Common Weakness and Known Vulnerability Evaluation	Common Weakness and Known Vulnerability Evaluation		Integration of Common Weaknesses and Known Vulnerabilities with Static Analysis tool automation
	Proprietary information, copyrighted material, and trade secret detection	Rigorous and systematic abstraction, filtration, comparison	Approximate Matching using text strings to detect fragments Function Extraction for abstracting intended function	Hypernion using Behavior Specification Units (BSU's) for detecting intended function
Risk Management Calculation	Goal identification and actually calculating risks based on systematically distinguishing factors, sources of risk, and problems using levels of confidence	Actually calculating risks based on systematically distinguishing factors, sources of risk, and problems using levels of confidence	Counterfeit and tainted component detection confidence level Trusted chain of custody confidence level	Cognitive Systems, that is, IBM Watson, with emphasis on levels of confidence used in calculating risk
Resilience	Defense in Depth, Business Continuity, Survivability and RMA Engineering, and Resiliency featuring: coordinated recovery time objectives, digital situation awareness, distributed supervisory control, selective availability, operation sensing and monitoring, information and data recovery, and interoperability of data and information exchange	Coordinated recovery time objectives, digital situation awareness, selective availability, data and information exchange		Cognitive Systems, that is, IBM Watson, with emphasis on anticipation, avoidance, and digital situation awareness

Table 1. Software Assurance Practice and Technology Assessment

Software Assurance Layered Defensive Approach

The kernel of the layered defense approach to Software Assurance is Structured Programming and the rigorous and provably correct use of zero and one predicate prime programs along with proper programs composed of multiple prime programs limited to single entry and single exit [2]. The layered defense approach recommended here includes Defense in Depth, Build Security In, Risk Management Calculation, and Resilience.

1. Defense in Depth is composed of encryption, identity management, access control, authorization management, accountability management, incident management, configuration management, and security assurance operations.
2. Build Security In is composed of Structured Programming with emphasis on correctness [2, 9], Cleanroom Software Engineering and Statistical Testing [10, 11], Software Inspections [12], Static Analysis tool use [13], Function Extraction tool use [14], common weakness [15] and known vulnerability evaluation [16].
3. Risk Management calculation includes goal identification and actually calculating risks based on systematically distinguishing factors, sources of risk, and problems using levels of confidence.
4. Resiliency prerequisites are Defense in Depth, Business Continuity, and Survivability, and features of Resiliency include coordinated recovery time objectives, digital situation awareness, distributed supervisory control, selective availability, operation sensing and monitoring, information and data recovery, and interoperability of data and information exchange [17, 18, 19].

These layered defense tactics are assessed in Table 1 in terms of Software Assurance practice and technology distinguished by known practice, underutilized practice, underutilized technology, and advanced technology needed. A framework of cyber tactics including anticipation, detection, attribution, and counter measures is needed to reason about tools, tiers, and threat categories [20]. Cause and effect chain elements spanning goals, weaknesses, attributes of building security in, attack outcomes, bad actors, and consequences must be identified and traced for selected Cyber Tactics. Consequences must be traced for selected goals and attack outcomes. In the spirit of continuous improvement and consistent with the erratic and rapid pace of both technology and threat emergence, all methods and means including tools need to be assessed and evaluated continuously. Refreshment and update is not so much periodic; instead refreshment is on demand in accordance with the "check everything every time" principle [21].

Obstacles and Underutilized Technology

The practice of Technical Debt fosters permissible and persistent neglect and needs to be discontinued and eliminated [7]. Technical Debt is the organizational, project, or engineering neglect of known good practice that can result in persistent public, user, customer, staff, reputation, or financial cost [6].

While the CMMI provides an excellent platform for achieving widespread adoption of software process maturity, it is incomplete with respect to Software Assurance and Cyber Security process. The importance of underutilized technology lies in the potential of the CMMI and its widespread infrastructure of usage to serve as the platform for Software Assurance adoption and dissemination [4].

With respect to Defense in Depth, encryption, authorization management, and security assurance operations are underutilized practices. In addition three factor authentication based on what you are, what you have, and what you know is an underutilized technology. Finally, trusted encryption advances are needed.

Build Security In

Quite specifically, Structured Programming with the rigorous and provably correct use of zero and one predicate prime programs along with proper programs composed of multiple prime programs limited to single entry and single exit is the kernel of trustworthy Software Assurance [2].

With respect to Build Security In, there are numerous underutilized practices including Structured Programming with insufficient emphasis on correctness and proof of correctness and Inspections with well defined standard of excellence spanning completeness, correctness, style, rules of construction, and multiple views [22, 23]; Cleanroom Software Engineering with its underutilized practice and Statistical Testing and its underutilized technology; Function Extraction and its underutilized practice and technology and its need for advanced technology; the underutilized practice of common weaknesses and known vulnerabilities and the need for additional advanced technology; and the advanced use case of detecting proprietary information, copyrighted material, and trade secrets using Carnegie Mellon University's (CMU) Function Extraction [14], NIST's Approximate Matching [24], and Oak Ridge National Laboratory's (ORNL) Hydernion [25].

Promising Areas for Research

Defense in Depth awaits trusted encryption advances. Static Analysis needs commonality and interoperable tool vendor results post processing advances to disambiguate multi-vendor results and decimate the number of false positives. Function Extraction needs to be able to process large programs and systems such as, those found in legacy software systems, COTS, reuse libraries, and commercial software releases. In addition, Function Extraction needs to be able to process source code from a wide range of languages and object code from various instruction set architectures. Common weakness and known vulnerabilities need to be integrated with Static Analysis tool automation.

Proprietary information, copyrighted material, and trade secret detection can potentially be determined using NIST's Approximate Matching using text strings to detect fragments. More advanced, CMU's Function Extraction for abstracting intended function and ORNL's Hypernion using Behavior Specification Units (BSU's) for detecting intended function offer promise. Proprietary software is licensed under exclusive legal right of the copyright holder with the intent that the licensee is given the right to use the software only under certain conditions, and restricted from other uses. In the legal community, the Abstraction-Filtration-Comparison (AFC) test[1] is a three-step process for determining substantial similarity of the non-literal elements of a computer program. Abstraction distinguishes which aspects of the program constitute its expression and which are the ideas. Filtration removes from consideration aspects of the program that are not legally protectable by copyright, such as, elements associated with efficiency, external factors, and the public domain. Comparison considers whether just those aspects of the program that constitute its expression and not those aspects not legally protected by copyright are present in the Clean System.

Risk Management Calculation needs to apply Cognitive Systems, that is, IBM Watson, with its emphasis on levels of confidence useful in calculating risk. More specifically, with respect to Risk Management Calculation, while goal identification is commonplace, actually calculating risks based on systematically distinguishing factors, sources of risk, and problems using levels of confidence is an underutilized practice. Beyond that, determining the level of confidence in assigning a failed state to a factor is assisted by evidence-based assurance assessment questions.

Resilience needs to apply Cognitive Systems, that is, IBM Watson, with its potential for anticipation, avoidance, and digital situation awareness. More specifically, with respect to Resilience, coordinated recovery time objectives, digital situation awareness, selective availability, and data and information exchange represent underutilized practices and features.

Line of Sight for Adoption

The line of sight for a target organization adopting Software Assurance is guided by a five level maturity value map including Defense in Depth, Build Security In, Risk Management Calculation, and Resilience.

* **Level 2:** Defense in Depth including standards for existing methods and practice for assuring security focuses on protection from vulnerabilities and threats through security in depth. Protective measures include encryption, identity management, access control, authorization management, accountability management, incident management, configuration management, and security assurance operations.
* **Level 3:** Build Security In including standards for existing methods and practice for trustworthiness focuses on building security in by constructing software components, systems, and systems of system to an engineered standard of excellence based on completeness, correctness, and rules of construction. Beginning with discontinuing the practice of Technical Debt, Build Security In practices include structured programming with an emphasis on correctness, Clean Room software engineering and statistical testing, software inspections process with well defined standard of excellence, the application of static analysis tools, the use of common weaknesses and known vulnerabilities, and the application of Function Extraction (FX).
* **Level 4:** Risk Management Calculation including standards for existing methods and practice for calculating software assurance uncertainty focuses on factors, sources of risk and problems evaluated for each goal, a count of factors that might

serve as sources of problems in goal achievement, and a count of factors that represent objectives that are in a failed state. Determining the level of confidence in assigning a failed state is assisted by evidence-based assurance assessment questions. For each goal, the calculated risk is the number of problems divided by factors evaluated expressed as a percent.

* **Level 5:** Resilience of Systems of Systems [18,19] and Supply Chains [26, 27] including standards for existing methods and practice for assuring resiliency under stress focuses on anticipating, avoiding, withstanding, mitigating, and recovering from the effects of adversity whether natural or manmade under all circumstances of use. Features of resiliency engineering include coordinated recovery time objectives, digital situation awareness, distributed supervisory control, selective availability, operation sensing and monitoring, information and data recovery, and interoperability of data and information exchange.

Software Assurance maturity seeks some of the same consequential outcomes as traditional organizational maturity, such as, more efficiency, more predictability, less risk, and less chaos. Accordingly efforts would be directed at incorporating Software Assurance and Cyber Security into the CMMI in order to utilize its existing effective dissemination and adoption infrastructure [28, 4]. While Software Assurance certification programs and university degree programs may successfully impact and meet the developmental needs of individuals, the organizational capability and its systematic appraisal and assessment would be tied to a Software Assurance-enabled CMMI.

Summary and Conclusion

The layered defense approach recommended here includes the Defense in Depth, Build Security In, Risk Management Calculation, and Resilience engineering and processes dependent on an elevated state of software engineering rigor. There is much room for improvement in the rigor of Software Assurance methods and practices capable of delivering consequential outcomes including components that are provably correct; demonstrably free of weaknesses and Cyber Security vulnerabilities; and demonstrably free of improper proprietary information, copyrighted material, and trade secrets.

Fueled by STEM shortfall, Software Assurance is impeded by the acceptance of Technical Debt, a practice that breeds neglect and should be discontinued. Advancing Software Assurance adoption would be stimulated by renovating the CMMI to include Software Assurance and Cyber Security process capabilities.

Strengthening Defense in Depth encryption, authorization management, and security assurance operations practices is needed. Build Security In emphasis on correctness proofs, correctness by construction, and the defined standard of excellence for completeness, correctness, style, rules of construction, and multiple views is the required foundation for the increased rigor being sought. The full utilization of statistical testing, Function Extraction, and static analysis automation would verify and calibrate the improvement in rigor in a cost effective and schedule efficient manner enabling the Next Generation Software Engineering goal of doing more with less… fast [29]. Much needs to be done on Resilience including the widespread implementation of Resiliency functions and features.

ABOUT THE AUTHOR

Don O'Neill served as the President of the Center for National Software Studies (CNSS) from 2005 to 2008. Following twenty-seven years with IBM's Federal Systems Division (FSD), he completed a three-year residency at Carnegie Mellon University's Software Engineering Institute (SEI) under IBM's Technical Academic Career Program and has served as an SEI Visiting Scientist. A seasoned software engineering manager, technologist, independent consultant, and expert witness, he has a Bachelor of Science degree in mathematics from Dickinson College in Carlisle, Pennsylvania. His current research is directed at public policy strategies for deploying resiliency in the nation's critical infrastructure; disruptive game changing fixed price contracting tactics to achieve DOD austerity; smart and trusted tactics and practices in Supply Chain Risk Management Assurance; and a defined Software Clean Room Method for transforming a proprietary system into a Clean System devoid of proprietary information, copyrighted material, and trade secrets and confirming, verifying, and validating the results.

Phone: 301-990-0377
E-mail: oneilldon@aol.com

NOTES

1. Wikipedia, "Abstraction-Filtration-Comparison test", <http://en.wikipedia.org/wiki/Abstraction-Filtration-Comparison_test>

REFERENCES

1. "Software 2015: A National Software Strategy to Ensure U.S. Security and Competitiveness," Center for National Software Studies, May 2005 <http://www.cnsoftware.org/nss2report/NSS2FinalReport04-29-05PDF.pdf>
2. Linger, R.C., H.D. Mills, B.I. Witt, "Structured Programming: Theory and Practice", Addison-Wesley Publishing Company, Inc., 1979
3. O'Neill, Don, "Software Assurance Trustworthiness and Rigor", 2013, <http://youtu.be/N2IT84hEhkU>
4. Goertzel, Karen Mercedes, Theodore Winograd, Holly Lynne McKinley, Lyndon Oh, Michael Colon, Thomas McGibbon, Elaine Fedchak, Robert Vienneau, "State of the Art Report: Software Security Assurance", IATAC and DACS , July 31, 2007 <http://iac.dtic.mil/csiac/download/security.pdf>
5. O'Neill, Don "Software 2015: Situation Dire YouTube", 2013, <http://youtu.be/qKBKbhH_J64>
6. O'Neill, Don, "Technical Debt in the Code: Cost to Software Planning", Defense AT&L Magazine, March-April 2013 <http://www.dau.mil/pubscats/ATL%20Docs/Mar_Apr_2013/O%27Neill.pdf>
7. O'Neill, Don, "Technical Debt YouTube Presentations", 2013, <http://youtu.be/1z6LPnRL4wU>
8. O'Neill, Don, "A Defined Software Clean Room Method for Transforming a Dirty System into a Clean System", <http://youtu.be/UTfHD1ORjOs, 2014>
9. Kourie, D.G., B.W. Watson, "The Correctness-by-Construction Approach to Programming", Springer, ISBN: 978-3-642-27918-8, 264 pages, 2012
10. Linger, Richard C. and Carmen J. Trammell, "Cleanroom Software Engineering Reference Model", Version 1.0, Technical Report CMU/SEI-96-TR-022, Carnegie Mellon University Software Engineering Institute, November 1996 <http://resources.sei.cmu.edu/asset_files/TechnicalReport/1996_005_001_16502.pdf>
11. Trammell, Carmen J., Richard C. Linger and Jesse H. Poore Stacy J. Prowell "Cleanroom Software Engineering: Technology and Process", SEI Series in Software Engineering, Addison-Wesley, March 19, 1999, 390 pages
12. O'Neill, Don, Software Inspections Situations Video, <http://youtu.be/BvIvh5RD4Bk>, 15:22 minutes
13. Okun Vadim, Aurelien Delaitre, Paul E. Black, "Report on the Static Analysis Tool Exposition SATE IV, NIST Special Publication 500-297, January 2013 <http://samate.nist.gov/docs/NIST_Special_Publication_500-297.pdf >
14. Hevner, Alan R., Richard C. Linger, Rosann W. Collins, Mark G. Pleszkoch, Stacy J. Prowell, Gwendolyn H. Walton, "The Impact of Function Extraction Technology on Next Generation Software Engineering", July 2005, CMU/SEI-2005-TR-015 <http://www.sei.cmu.edu/reports/05tr015.pdf>
15. "Common Weakness Enumeration" MITRE, 2013, <http://cwe.mitre.org>
16. "Common Vulnerabilities & Exposures Database". 2013, <http://cve.mitre.org>
17. Resilient Military Systems and the Advanced Cyber Threat, Department of Defense Defense Science Board Task Force Report, January 2013
18. O'Neill, Don, "Maturity Framework for Assuring Resiliency Under Stress", Build Security In web site, Operated by Software Engineering Institute and Sponsored by Department of Homeland Security, July 2008
19. O'Neill, Don, "Meeting the Challenge of Assuring Resiliency Under Stress", CrossTalk, The Journal of Defense Software Engineering, September/October 2009 <http://www.crosstalkonline.org/storage/issue-archives/2009/200909/200909-ONeill.pdf>
20. O'Neill, Don, "Cyber Strategy, Analytics, and Tradeoffs: A Cyber Tactics Study", CrossTalk, The Journal of Defense Software Engineering, September/October 2011 <http://www.crosstalkonline.org/storage/issue-archives/2011/201109/201109ONeill.pdf>
21. Mead, Nancy R. and Carol A. Sledge, "Foundations of Software Lecture", Software Assurance for Executives, MSWA Curriculum, CERT, Software Engineering Institute, Carnegie Mellon University, 2013 <http://www.cert.org/mswa/SAE-course-material-list.html>
22. O'Neill, Don, "Software Inspections Tutorial", Software Engineering Institute, Technical Review, 1988
23. O'Neill, Don, "Peer Reviews", Encyclopedia of Software Engineering- Volume 2, Second Edition, Edited by John Marciniak, John Wiley & Sons, Inc., January 2002, pp. 929-945
24. "Approximate Matching: Definition and Terminology", Draft NIST Special Publication 800-168, January 2014, <http://csrc.nist.gov/publications/drafts/800-168/sp800_168_draft.pdf>
25. "The Hyperion System: Computing the Behavior of Software", Cyber and Information Security Research Group, Oak Ridge National Laboratory, 2014
26. Croll, Paul R., "Supply Chain Risk Management: Understanding Vulnerabilities in Code you Buy, Build, or Integrate", CrossTalk, The Journal of Defense Software Engineering, March/April 2012 <http://www.crosstalkonline.org/storage/issue-archives/2012/201203/201203-Croll.pdf>
27. O'Neill, Don, "Software and Supply Chain Risk Management (SSCRM) Assurance Framework", 2013, <http://youtu.be/1uUnG6HY1_c>
28. O'Neill, Don, "Cyber Security and CMMI", 2013, <http://youtu.be/evkdv0j4ZRo>
29. O'Neill, Don, "Preparing the Ground for Next Generation Software Engineering", IEEE Reliability Society, Annual Technology Report 2008, pp. 148-151, June 2009
30. O'Neill, Don, "Calculating Security Return on Investment", Build Security In web site, Operated by Software Engineering Institute and Sponsored by Department of Homeland Security, February 2007
31. Software Analysis and Forensic Engineering (SAFE), <http://www.safe-corp.biz/products_codesuite.htm>
32. O'Neill, Don "IBM Watson Experimental Prototype Challenge: Supply Chain Risk Management Assurance", 2013, <http://youtu.be/QycPEI0RY20>

N-Version Architectural Framework for Application Security Automation (NVASA)

Majid Malaika, Southern Methodist University
Suku Nair, Southern Methodist University
Frank Coyle, Southern Methodist University

Abstract. In this paper, we expose application security issues by presenting the usage of N-Version programming methodology to produce a new architectural framework to automate and enhance application security. Web applications and cloud computing are dominating the digital world; therefore, our goal is to build resilient systems that can detect and prevent both known and zero-day application attacks. Automated process flow not only reduces security efforts during the Software Development Life Cycle (SDLC), but also enhances the overall application security. In addition, we propose compartmentalizing the application into separate components and applying the N-Version methodology to the critical ones to reduce the additional overhead introduced by the N-Version methodology.

Introduction

With the World Wide Web, online applications have become vital and more compelling than ever. Many financial transactions are made from home/office without the need to physically be present at the bank to finalize these transactions. Governments around the world are digitizing most of their services to make the process more convenient for their citizens. It is estimated that more than 60% of Internet users interact with government websites to perform tasks such as completing applications, renewals or inquiring for information. In fact, this number is increasing every year [1]. Online shopping has become more prevalent and convenient to customers than ever. In 2011, it is estimated that more than a trillion U.S. dollars were spent on online merchandise in the USA alone [2]. Rapid growth and huge improvements in information technology have raised many challenges. One challenge is application security.

The DoD relies heavily on software to deliver instantly accessible data to its users [3]. However, this makes remote malicious attacks a serious threat to DoD systems and users. Methods investigated and presented in this paper will complement DoD efforts in detecting and preventing these attacks through an application security framework (NVASA) that uses the N-Version programming methodology.

The Security Problem

Today, the main problem with system security is that it is viewed by enterprises as a commodity, where the usage of password patterns and the integration of anti-virus applications and firewalls promote a false sense of security. This is because most cyber-attacks target the application layer rather than the physical or network layer. Most annual security reports demonstrate insufficient application security measures taken by both enterprises and individuals [4, 5]. Some of the challenges in providing application security [6, 7] include: 1) System Complexity, applications today have the capability to interact automatically with users and/or other systems in a very complex fashion, therefore increasing the possibility of error injection during the SDLC. 2) Ubiquitous networking, more systems are connected to the Internet without appropriate security, thus becoming available online. 3) Built-in Extensibility, This is a desired feature in software engineering because it would enable the flexibility to add new components to the existing system in the future. Therefore, making it possible to inject malicious code in to the system. 4) Common platforms, While common platforms reduce cost and time when implementing new technologies or building an application, they also increase a malicious user's chances of exploiting more systems.

Most active attacks are carried out by exploiting existing vulnerabilities in the system. These vulnerabilities could be: 1) architectural design, 2) implementation, or 3) operational and platform vulnerabilities [8]. Human faults made during the design phase result in architectural design errors into the model's structure. Consequently, human faults made during the writing of the code would result in implementation vulnerabilities. Finally, faults in configuration files are operational and are considered platform vulnerabilities. The most dangerous vulnerabilities of all categories are the ones leading to immediate unauthorized access of the application [9, 10, 11, 12]. Providing high levels of application security is paramount in ensuring network, systems, and data security.

The remainder of the paper is organized as follows: Section II presents related work in the field of application security followed by a description of the methodology of N-Version programming. Section III presents the building blocks of our proposed N-Version Architectural Framework for Application Security Automation. In section IV, we present a prototype of the proposed NVASA framework and experimental results. In section V we introduce compartmentalization to reduce overhead. Finally, section VI concludes with a look at future research.

Related Work

Application security is often one step behind the latest cyber-attack schemes for reasons discussed in the previous section. The current emphasis on application security is to fix existing implementation errors that could be exploited by publicly known attacks such as Buffer Overflow, SQL Injection and Cross Site Scripting [9, 11, 12]. Other related work emphasizes extensive revisions to eliminate or reduce the injection of errors during the application's lifecycle.

Application Security

Most current related work in the field of application security focuses on enforcing extensive security guidelines during the SDLC [13, 14]. These guidelines focus on providing the regulations needed to promote the development of secure applications through the SDLC. The initial approach to application security espoused manual audits to the source code [15]. This approach consists of reading the source code line by line to detect and fix existing vulnerabilities. Another method is "fuzzing" testing, which

is followed by inputting a distorted or illegal input to the application and monitoring the behavior of the application. Sulley and SPIKE frameworks [16] are two examples of "fuzzing" testing.

Runtime checking is another method for detecting and preventing the exploitation of existing vulnerabilities. This method of testing is followed by adding special checks within the source code to ensure the program behaves as desired. ProPolice framework [17] is a GNU Compiler Collection (GCC) extension developed by IBM to protect against stack smashing that uses runtime checking. Mudflap framework [18] is another GCC extension for pointer debugging that uses runtime checking to detect and prevent exploitation of vulnerabilities.

These solutions are limited and have major disadvantages. First, actual security vulnerabilities are triggered by a certain specific set of circumstances which makes it extremely hard to strike using random "fuzzing" testing. A second limitation is latent security vulnerabilities that are present within critical systems but concealed from the system's stakeholders. These latent vulnerabilities can damage an organization's reputation and could lead to financial loss. Further, it takes an average of 14 weeks to patch or fix an existing vulnerability after discovery, which opens a window for additional attacks leading to more damages [5]. A third limitation is the high cost of implementing a secure application because the enforcement of security training as well as hiring special security testers and purchasing specific security tools adds enormously to the total cost. A fourth limitation is the increased time-to-market since these solutions engage in extensive training and thorough testing of the code. Finally, following these extensive guidelines often results in the detection of known existing vulnerabilities. However, these extensive guidelines have no scheme for detecting or preventing latent vulnerabilities from being exploited.

N-Version Programming

The concept N-Version Programming was introduced in the late 1970's by Liming Chen and Algirdas Avizienis. It is defined as, "The independent generation of N≥2 functionally equivalent programs from the same initial specification" [19].

The aim of the N-Version programming methodology is to improve software reliability. It is introduced by the following proposal: "The independence of programming efforts will greatly reduce the probability of identical software faults occurring in two or more versions of the program" [19].

Therefore, to build a pure N-Version model as shown in Fig. 1, two policies must be adopted: 1) All versions must share the exact same initial specification. The purpose of the initial specification is to state the functional requirements that stakeholders want the application to perform. They must be clear and detailed oriented to eliminate any confusion during the development process. 2) Versions must be independently generated. This is achieved by choosing different algorithms and programming languages for each version, as well as the independent processes for generating the versions, which should be carried by N independent individuals or groups that have no interaction with each other. This isolation of design and process between groups, coupled with the diversity of choosing programming languages and algorithms, greatly reduces the probability of producing identical software faults in two or more versions [19, 20].

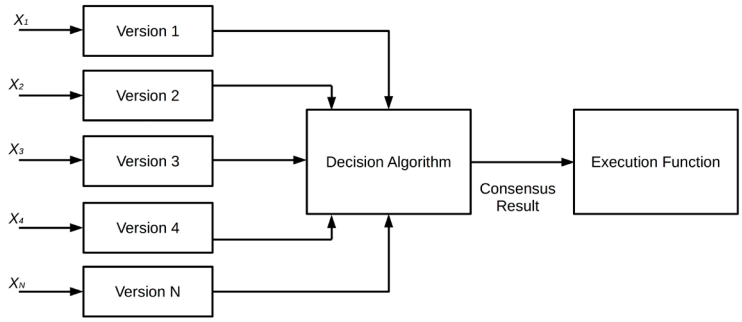


Figure 1. N-Version Model [21]

Building N-Versions Architectural Framework for Application Security Automation (NVASA)

To build the NVASA framework, we first collect the initial specifications and requirements from stakeholders. Second, we hand the specifications to N-Different programmers or groups of programmers. Each group develops a separate version using a different language and algorithm than the rest of the groups to satisfy the diversity of the N-Version methodology. Depending on the language and specifications outlined by stakeholders, one of the groups will develop the NVASA framework's layers mentioned below [21].

As shown in Fig. 2, the NVASA framework is constructed of four layers. The first layer is the N-Version routing layer, the second layer is the N-Version environment layer, the third layer is the N-Version decision layer, and the fourth layer is the backend application server layer.

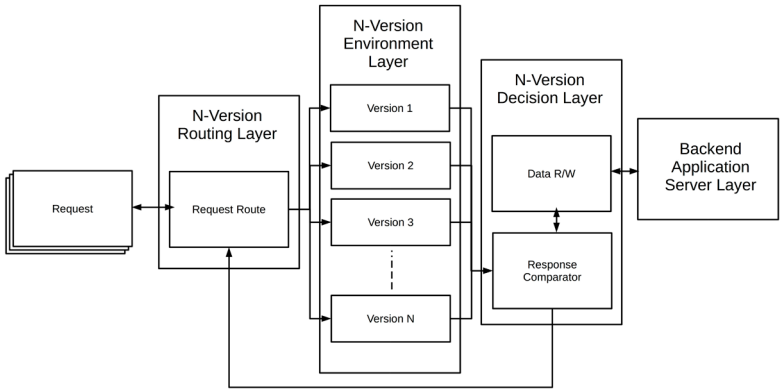


Figure 2. NVASA Framework Four Layers [21]

N-Version Routing Layer

The N-Version routing layer connects the framework with users/applications over the network. It is responsible for receiving requests from external users and routing the input to the N-Versions in the environment layer to be executed. The routing layer is also responsible for replying to requesters with the appropriate response after executing the request.

N-Version Environment Layer

The N-Version environment layer contains the N-Version applications where each version is designed and developed by an independent individual or group using a unique algorithm and/or programming language as mentioned in Section 2. Each version receives its execution command along with the identical input of

all N-Versions from the routing layer. All the versions then execute concurrently, which eliminates any reduction in performance. The N-Versions can be either loosely coupled running on different platforms in different physical locations or tightly coupled running on the same physical machine; this is decided based on the initial design specification of the NVASA framework.

Loosely coupled versions have many advantages: 1) The time overhead is reduced compared to tightly coupled versions running on one machine. 2) There is no single point of failure within the N-Version environment layer compared to tightly coupled versions. 3) Loosely coupled systems scale better than tightly-coupled systems. On the other hand, loosely coupled versions add more complexity to the development and testing phases of an application since messaging schemes must be implemented to connect layers and components. Additionally, the communication channels need to be protected through transport or network layer security protocols. Depending on the type of applications, communication between various layers and versions may cause additional time overhead.

N-Version Decision Layer

The N-Version decision layer is composed of two components: 1) The Response Comparator component, and 2) The Data Read and Write (R/W) Component. The response comparator component receives the N-Version outputs where a decision algorithm applies generic consensus rules to determine a consensus output. The role of the decision algorithm is to determine exploited versions by identifying conflicting output compared to the majority of the versions and removing the exploited and breached versions from the pool, thereby eliminating any malicious attempt to exploit the application. If necessary the response comparator component then passes the consensus output to the data read and write (R/W) component to generate the R/W command from the consensus output to be applied to the backend application server layer. Finally, the data read and write component generates the output or confirmation based on the initial request and passes it to the response comparator to be then sent to the request route component to reply to the requester.

Backend Application Server Layer

For applications that read or write data to or from a server or database, the backend application server layer is essential. It receives the consensus output from the data R/W component in the decision layer, preventing any direct communications between the backend application server layer and the N-Versions in the environment layer. This significant design requirement prevents any successful exploits from modifying or leaking the information by a malicious request.

Implementation and Results

A prototype NVASA framework was implemented and testing conducted to validate the protection provided by leveraging an N-Version service implementation in a distributed or web application architecture. In order to develop a practical prototype that can produce reliable results, we searched for pre-developed AES implementations online to be used as our N-Versions. These N-Versions must be written in different languages and by different developers in order to satisfy the diversity required in the N-Version programming methodology mentioned earlier. Three AES versions were located and used, the three were similar in structure and produced the same output. The first version was written in Java by Neal R. Wagner [22].The second version was written in C# by James McCaffrey [23]. And the third and final version was written in C/C++ by Niyaz PK [24].

As part of the AES framework implementation we exposed the C# and C/C++ versions as web services to be able to integrate them with the NVASA Framework. Each web service would accept a request with a 192-bit key and 128-bit clear text block and reply with the cipher text if the encryption was successful. Since the interface was developed using the Java language, a simple function call was sufficient to connect to the Java version which will also accept a request with a 192-bit key and a 128-bit clear text block. Fig. 3 shows the structure of our AES NVASA Framework implementation. We minimized our involvement in writing or modifying any code as much as possible to satisfy the diversity required in the N-Version programming methodology. Therefore, we used pre-developed implementations that matched each version's language to integrate each with the NVASA framework.

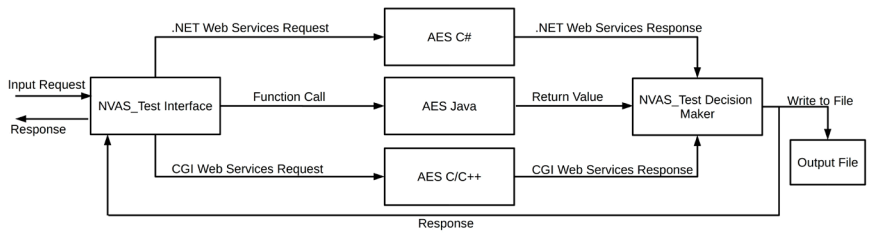


Figure 3. NVASA Framework AES Implementation

For this prototype we wrote a java class to act as the Routing and Decision Layer (Layer 1 and 3 From Fig. 3). The java class simply accepts the user input then executes all three versions simultaneously. Each version then produces its output which is then fed to the decision algorithm. The decision algorithm then runs to reach a consensus output. If a consensus is reached, the cipher text is written to a file and sent to the requester; otherwise, we have the choice of either dropping the request or replying back to the requester with a "Request Denied" message.

To test our NVASA implementation we developed a benchmark of a 100 test cases, each test case included a key length, key, and a clear text block. We executed each standalone version against each test case first, then executed the same set of test cases against the NVASA AES implementation. Table 1. shows a segment of the failed test cases and a comparison between the output of each version and our NVASA AES implementation.

Looking at table 1. we realize that in the first two test cases more than one version failed from a total of three versions, but providentially the NVASA recovered from such an exploit due to the fact that each version produced a different and unique output; therefore, the decision algorithm couldn't reach a consensus output; thus, dropped the request and never propagated it through the system. This step is essential because the decision algorithm will detect an attack and prevent it from propagating through the system even if it was successful in exploiting most of the versions.

This NVASA prototype shows how the new framework is able to improve Security by detecting and preventing known and potential zero-day attacks using the N-Version Programming methodology which revealed that the diversity of the languages and algorithm used greatly reduces the probability of having identical vulnerabilities in two or more versions. This was achieved without the need to modify the source code within the versions or install any patches.

Overhead Reduction

We have demonstrated that the NVASA framework is effective in improving security by detecting zero-day cyber attacks; however, the NVASA adds overhead to the development phase in the SDLC compared to the single version implementation. Therefore, compartmentalizing the application's architecture and applying the NVASA framework to the critical components in terms of security can reduce the overhead while improving security.

There are standards produced to assist in identifying the critical components within an application. The Common Criteria Information Technology Security Evaluation [25] is an international standard widely used to identify the critical security components. Other work has been done by Young, 1991. Also other work has been done by Andrew Rae, and Colin Fidge [26] to improve Young's approach by making it more efficient.

Rae's approach relies on the category of information manipulated by the component to prioritize and identify the critical components. The assumption here is that the application architecture model consists of components and connections. These connections carry the information from one component to another while the components can generate or manipulate the information to achieve the application's goal.

Based on the carried information, components and connections are categorized into two categories, First, Data Connection/Component where classified information is carried or manipulated by the Data component/connection. Classified data is defined as critical and sensitive data to the application or external world. Second, Control Connection/Component where non-classified information is carried by the Control connection/component. Unclassified information is defined as non-critical data to the application or external world. If a component/connection acts as both 'Data' and 'Control', then we classify it as 'data' to protect the classified information within.

Fig. 4 demonstrates N-Versioning the compartmentalized critical component highlighted in red. The nFork acts as the interface layer, it executes the N-Versions with the incoming parameters while the nJoin acts as the decision layer where it receives all N-Version outputs and executes the decision algorithm to come to a consensus.

			Standalone			
Number	**Test Case**	**Test For**	**C/C++**	**Java**	**C#**	**NVASA**
1	"73204483711"	Integer Over flow in Key Length	×	✓	×	✓
2	"zz...." in the plaintext or key field	Input Injection (Non HEX) in the plaintext and/or key field	×	×	×	✓
3	Null Input (Command Line or File)	Exception Handling with plaintext and/or key	✓	×	✓	✓
4	"Ree" instead of an Integer	Type Injection in the Key length field	×	✓	✓	✓
5	Normal HEX 192 Key and 128 Key	Normal Input	✓	✓	✓	✓
6	Larger key than specified (200 bits instead of 192 bits)	Input Validation Key Field	✓	✓	×	✓

✓ Program Passed Test Case × Program Failed Test Case

Table 1. Results of Testing NVASA Implementation Vs. the Standalone Versions

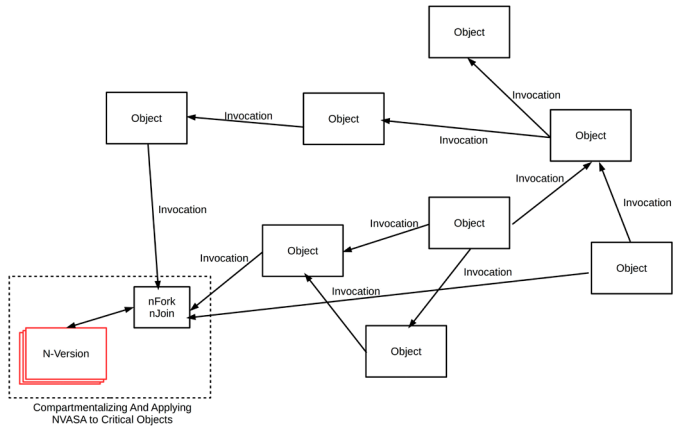


Figure 4. Applying the NVASA framework to the Critical Components

Conclusion and Future Work

The NVASA framework achieves better application security for critical web services, online, and cloud computing applications and improves the overall system security by using the N-Version programming methodology that comes to a consensus based on all the N-Versions outputs before applying any of these commands to the Backend Application Server Layer. This prevents any exploitation of the system which would lead to the destruction, modification or leakage of the confidential information to malicious users.

We showed the effectiveness of the NVASA architecture via the AES implementation, and how the decision layer is not only detecting attacks but also preventing the propagation of the effects of an attack. We showed with regards to some test cases how the decision algorithm was able to identify the irregularity among the N-Versions outputs, this was true even when all the N-Versions failed. This was achieved because each version failed uniquely producing a distinct output. Hence, enabling the decision algorithm to detect and prevent any of the malformed outputs from propagating further through the system.

In addition, we demonstrated compartmentalizing the application and applying the NVASA framework to the critical components as a method to reduce the added overhead associated with the implementation of the N-Version methodology.

Future work involves dealing with attacks targeting the state of the machine, for example backdoors etc. To that end we are further investigating the compartmentalization of components and its effects to real world applications in conjunction with automated code generation.

ABOUT THE AUTHORS

Majid Malaika received his B.S degree in Computer Science from King Abdul Aziz University. He received his M.S degree in Computer Engineering and his Doctorate Degree in Software Engineering from Southern Methodist University. Currently, he is an application security analyst at Amplify, a leading educational technology company in New York, USA. Majid's prior engagements were security consulting for multinational financial firms in New York. His work experience includes architecture risk analysis, risk management, code review, and penetration testing.

E-mail: majid.malaika@gmail.com

Suku Nair received his B.S. in Electronics and Communication Engineering from the University of Kerala, India. He received his M.S. and Ph.D. in Electrical and Computer Engineering from the University of Illinois at Urbana in 1988 and 1990, respectively. Currently, he is a Professor and Chair of the department of Computer Science and Engineering SMU at Dallas where he held a J. Lindsay Embrey Trustee Professorship in Engineering. His research interests include Cyber Security and Software Defined Networks.

E-mail: nair@lyle.smu.edu

Frank Coyle is a Senior Lecturer at SMU. He received his BS degree from Fordham College, his MS from Georgia Tech and his PhD from Southern Methodist University. His areas of interest are software engineering, software security, and engineering education. He maintains a technology blog at www.drczone.com.

E-mail: coyle@lyle.smu.edu

Department of Computer Science and Engineering at Southern Methodist University
P.O.Box 750122 Dallas, TX 75275-0122
Phone: 214-768-3083
Fax: 214-768-1192

REFERENCES

1. Executive office of the President office of Management and Budget Washington, D.C. 20503 "E-Government Strategy" February 27, 2002.
2. U.S. Department of Commerce, Washington, D.C. 20233, "Quarterly Retail E-Commerce Sales 1st Quarter of 2011".
3. Mr. Jeff Hughes, Dr. Martin R. Stytz, Ph.D. "Advancing Software Security– The Software Protection Initiative".
4. IC3, "2008 Internet Crime Report" <http://www.nw3c.org/downloads/2008_IC3_Annual%20Report_3_27_09_small.pdf>
5. Networks and WhiteHat Security Solution, "Vulnerability Assessment Plus Web Application Firewall (VA+WAF)" June 2008. F5
6. John Viega & Gary McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way" (Chapter 1, Pages 9-13) Addison-Wesley Professional Computing Series, Addison-Wesley, New York. 2001
7. Schneier, Bruce "Secrets and Lies" (Chapter 23, Page 358), New York: John Wiley & Sons, 2000.
8. Cowan, C. Wagle, F. Calton Pu Beattie, S. Walpole, J., "Buffer overflows: attacks and defenses for the vulnerability of the Decade" 2000.
9. Guy-Vincent Jourdan, "Command Injection" 2005.
10. Chris Anley, "Advanced SQL Injection In SQL Server Applications" 2002.
11. scut / team teso, "Exploiting Format String Vulnerabilities" 2001.
12. President's Information Technology Advisory Committee. "Cyber Security: A Crisis of Prioritization19": Report to the President. National Coordination Office for Information Technology Research and Development, February 2005.
13. DISA for the DoD, "Application Security and Development" STIG, V2R1.
14. Richard Kissel & Kevin Stine & Matthew Scholl & Hart Rossman & Jim Fahlsing & Jessica Gulick, "Security Considerations in the System Development Life Cycle" NIST Special Publication 800-64 Revision 2, 2008.
15. Alexander Ivanov Sotirov, "Automatic Vulnerability Detection Using Static Source Code Analysis Thesis", 2005.
16. Michael Sutton & Adam Greene & Pedram Amini, "Fuzzing: Brute Force Vulnerability Discovery", 2007.
17. Etoh, Hiroaki, & Kunikazu Yoda, "Protecting from stack-smashing attacks" <http://www.trl.ibm.com/projects/security/ssp/main.html>
18. Frank Eigler, "Mudflap: Pointer use checking for C/C++" 2003.
19. A.A. Avizienis, "The Methodology of N-version Programming", Software Fault Tolerance, edited by M. Lyu, John Wiley & Sons, 1995, pp. 23-46.
20. Michael R. Lyu, Algirdas Aviz˘ienis , "Assuring Design Diversity in N-Version Software: A Design Paradigm for N-Version Programming".
21. Majid Malaika, Suku Nair, and Frank Coyle "Application Security Automation for Cloud Computing" CloudComp 2010.
22. Neal R. Wagner, "The Laws of Cryptography: Java Code for AES Encryption" 2001. <http://www.cs.utsa.edu/~wagner/laws/AESEncryptJava.html>
23. James McCaffrey, "Keep Your Data Secure with the New Advanced Encryption Standard" 2003. <http://msdn.microsoft.com/en-us/magazine/cc164055.aspx>
24. Niyaz PK, " Advanced Encryption Standard (AES) " <http://www.hoozi.com/posts/advanced-encryption-standard-aes-implementation-in-cc-with-comments-part-2-decryption/>
25. "Common Criteria Information Technology Security Evaluation", 1999
26. Andrew Rae, Colin Fidge, "Identifying Critical Components During Information Security Evaluation"

Acquisition Anonymous

Paul Kimmerly, Double Play Process Diagnostics, Inc.

Abstract. Everyone is familiar with the idea of acquisition. Acquisition is something that we all do in everyday activities. It involves obtaining products or services to address a need. In the case of government acquisition, it includes the use of supplier agreements. When basic principles are tried in government acquisition, it often does not work out as intended. Everyone has heard horror stories about cost overruns, incomplete products, botched services, fraud and cancelled projects in government acquisition. Why is that when it seems so simple to acquire products and services in everyday life? Luckily, there is a proven method to address addictive and self-destructive behavior, the twelve step program. This article offers a look at applying such an approach to improving acquisition efforts.

Introduction

Everyone is familiar with the idea of acquisition. Acquisition is something that we all do in everyday activities where it seems to come naturally. It involves obtaining products or services to address a need. In the case of government acquisition, it includes the use of supplier agreements. When basic principles are used in government acquisition, for some reason, it often does not work out as intended. The history of government acquisition provides horror stories about cost overruns, incomplete products, botched services, fraud and cancelled projects. Why is that when it seems so simple to acquire products and services in everyday life?

In the case of government acquisition, organizations often create their own problems by not paying attention to what they should be doing as an acquirer. Organizations often neglect to take care of their own activities because they are distracted by the activities of their suppliers and the impatience of customers. While the acquiring organization needs to be aware of suppliers' activities and customer concerns, they first need to focus on their own responsibilities. Acquisition starts with writing the contractual requirements, selecting the supplier and writing the supplier agreement. Acquirers often fail to include all the details needed to properly monitor the agreement. Once an agreement is reached, the acquiring organization has obligations as part of that agreement just as the supplier does. Instead of concentrating on their own activities, acquirers can become obsessed with overseeing the supplier's activities.

Obsessing over a supplier's activities can become addictive behavior. It leads to missed obligations and self-destructive actions. Luckily, there is a proven method to address addictive and self-destructive behavior, the twelve step program. The twelve steps can be used to address acquisition problems. The following program applies twelve proactive non-religious steps from Proactive Change[1] to acquisition.

Step 1: I get it: What I've been doing is self-destructive. I need to change.

It's become a cliché, but in order to fix a problem, one must acknowledge that there is a problem. Organizations need to realize that they should concentrate on controlling their own acquisition activities rather than spending time concentrating on supplier activities. The acquiring organization must acknowledge that if they get too involved in the supplier's activities they can neglect activities that will make the acquisition process go more smoothly. If acquirers fail to concentrate on their own activities, problems that arise in one acquisition effort are likely to occur in subsequent acquisitions.

Step 2: I see the big picture: The way to stop relapsing into self-destructive behaviors is to build a healthier sense of self.

Without coming to an understanding of their acquisition process and its limitations, organizations can continue counterproductive activities by sticking to the idea that "it's always been done this way." That does not mean that it's been done well. To gain a big picture understanding of the current process, guidelines or standards form a basis for comparison. Two widely accepted acquisition standards are the Federal Acquisition Regulation (FAR) and the CMMI® for Acquisition (CMMI-ACQ). An improvement effort for acquisition starts with a close look at the FAR or the CMMI-ACQ. The CMMI-ACQ focuses on what to do in relation to best practices from the acquisition industry. The FAR outlines the steps to be followed in government acquisition. The FAR delves into details of how activities are to be performed if specific conditions are met. The FAR can be a daunting document, but it contains an abundance of useful information. The CMMI-ACQ provides a framework for taking a big picture, holistic approach to performing acquisition activities. Simply put, the CMMI-ACQ describes what to do and the FAR describes how.

Step 3: I have an action plan: From now on, I am squarely facing everything that is in the way of feeling really satisfied with my life.

Getting a big picture view of the current process as we described in step 2 provides a good start for developing an improvement action plan. Acquisition organizations need to take an honest look at their current process to identify areas for improvement. While a government acquisition organization may claim it is following the FAR, the details of what is really going on may not be known. Before improvements can be identified, an organization should document the current state of the process as it is being performed, not the ideal process. Organizations may not want to admit that their current process differs from the ideal, but until they admit it, they cannot really begin to improve as we said in step 1. Once the current process is understood, a detailed improvement plan should be developed and used to address identified needs or concerns.

Step 4: I honestly look at the effects of my actions on others and myself.

Acquiring organizations need an understanding of how their actions are perceived by their customers and suppliers. They also need to take stock of their own actions. Customer surveys or other interactions help get an external look. For an internal look, organizations should conduct an appraisal of their current performed processes against the CMMI-ACQ or an audit against the FAR. By making an objective evaluation of their current processes, an organization can gain an understanding of processes in need of improvement and effective practices that can be leveraged to improve future acquisition efforts.

Step 5: I take responsibility for my actions.

Acquirers must understand and accept their responsibilities. The acquirer has primary responsibilities at the beginning of the acquisition process when they gather requirements, develop the acquisition plan, solicit bids and select the supplier. At that point, control of the project passes to the supplier for developing the product or service. During the development activities, the acquirer is responsible for monitoring and overseeing the supplier's activities. The acquirer cannot control supplier actions, but the acquirer has influence and provides leadership. After the product is delivered, responsibility shifts back to the acquirer. The acquirer should verify and validate the product before transitioning it to its operational environment. While all acquisition efforts involve several parties, the acquiring organization will be better able to meet their customers' needs and oversee the efforts of their supplier if they have better understanding and control over their own areas of responsibility.

Step 6: I see that my knee-jerk reactions have to do with being in the grip of more or less conscious fears.

Organizations become complacent and comfortable in doing things the way that they have always been done. However, by doing things the same way, the same problems are likely to occur. Organizations are not prepared for situations that arise and instead react from crisis to crisis if they do not have an action plan for each acquisition effort. Such a plan includes acquisition activities, supplier interactions and risk management. By reacting to every crisis without an overall plan of action, organizations can miss opportunities that will help them succeed. An established, documented and used set of processes can help an organization manage their efforts according to their plan without overreacting to each minor problem.

Step 7: I strive to find my motivation in a deeper sense of who I really am, rather than fear and defensiveness.

By identifying and using a set of measures, an organization gets a deeper understanding of process performance that enables management of current efforts and predictions of future performance. Basic measures related to size, effort, cost, schedule and defects should be defined, collected and used. Examples of measurements for acquisition include the amount of time to develop a request for proposal (RFP), the number of pages in a RFP, the number of proposals reviewed, actual versus planned schedule milestones, proposal review time, peer review defects for acquisition documents and the amount of time to create a supplier agreement. The use of measurements brings an understanding of process performance. Without measurement and the understanding it brings, organizations can find themselves reacting rather than managing. Constant crisis management often leads to doing the wrong things and addressing the wrong problems. It can also lead to addressing symptoms rather than the root cause of problems.

Step 8: I stop blaming and feeling blamed, with a willingness to heal the wounds.

Failures often turn into a search for the guilty and may lead to blame for the acquiring organization, suppliers and even customers. Organizations should look beyond blame and accept that problems occur. Rather than focus on problems, an organization should concentrate on how to recover and avoid the problems in future projects. This allows the organization to identify the causes for problems and find ways to address them.

Step 9: I swallow my pride, and sincerely apologize to people I've hurt, except when this would be counterproductive.

Organizations should accept and acknowledge the shortcomings of their processes and begin a sincere improvement program. When a problem occurs an organization needs to acknowledge the problem to all affected stakeholders. All stakeholders, including customers and suppliers should be involved in the discussion of acquisition problems and in the development of solutions to address known problems.

Step 10: I live mindfully, paying attention to the motives and effects of my actions.

As stated in step 7, an organization should establish a measurement program to monitor acquisition activities and project results. Measurements monitor progress and allow the organization to take corrective action when necessary. Objective evaluations should be conducted to ensure that established processes are being followed. Peer reviews provide a useful form of self-evaluation. Peer reviews of acquisition documents like RFPs, supplier agreements and operational plans help an organization stay mindful of the quality of their activities. Measurements and evaluations help identify the effects of the organization's actions and lead to an understanding of what is working and what needs to be improved.

Step 11: I stay in touch with a broader sense of who I really am, and a deeper sense of what I really want.

Acquirers should acknowledge who they are and recognize their specific areas of control. Acquirer responsibilities start with the first interactions with the customer to develop requirements. Customer needs are used to develop customer and contractual requirements. The contractual requirements help identify and select suppliers. When the supplier agreement is developed, the acquiring organization must ensure that all important components are included in the agreement. If it's not in the agreement, there is no guarantee that it will happen. Once the agreement is established, the acquirer needs to ensure that they and the supplier understand the assigned actions for both organizations. The acquirer is responsible for holding up their end of the bargain.

Step 12: A growing sense of wholeness and contentment motivates me to keep at it, and to share this process with others who are struggling.

An organization can learn from measurement results, appraisals, peer reviews and evaluations. Appraisals against standards, evaluations against established processes, peer review results and performance measures of the acquisition efforts can be used to leverage successes to other projects. The organization should use the measurement program to collect historical data for future use. Successes and failures should be shared with other parts of the organization to help with future acquisition efforts.

Conclusion

These twelve steps may seem simple and straightforward. Acquisition seems that way too. In order to improve, an organization must accept that there are problems to address. Improvement means change and change does not come easy. The twelve steps presented here can help an organization address some of the counterproductive behaviors common to a lot of acquisition efforts. By following these steps, an organization accepts their responsibilities, documents current processes as performed, evaluates processes against applicable standards, measures their results, monitors to ensure that processes are being followed and gains greater control over acquisition activities. As a result, acquirers will be better equipped to address customer needs, establish supplier agreements, monitor suppliers and manage their own acquisition activities.

Disclaimer:

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

ABOUT THE AUTHORS

Paul Kimmerly worked 25 years for the different incarnations of the United States Marine Corps Technology Services Organization (USMC TSO). He spent the last 16 years as the SEPG Lead before he retired in July 2011. Paul is a certified CMMI High Maturity Lead Appraiser and Instructor for the CMMI for Development and the CMMI for Acquisition. He works as an independent contractor with the CMMI Institute teaching and observing candidate lead appraisers and instructors. He is also a member of the editorial board for CROSSTALK magazine. He contributed several articles on process improvement to CROSSTALK. The articles cover topics including organizational change, management's role in process improvement and high maturity concepts. Since retiring from government service, Paul continues to work with clients in government and private industry as part of Double Play Process Diagnostics Inc.

Double Play Process Diagnostics, Inc.
PO Box 17015
Pensacola, FL 32522
Phone: 913-220-4499
E-mail: Paul.kimmerly@doubleplayconsulting.com

NOTES

1. Twelve Proactive Steps, Proactive Change is a Registered Service Mark of Serge Prengel, 119 W 57 St, New York City, NY 10019

Forecasting from Defect Signals

Paul Below, QSM

Abstract. On large software development and acquisition Programs, testing phases typically extend over many months. It is important to forecast the quality of the software at that future time when the schedule calls for testing to be complete. Shewhart's Control Charts can be applied to this purpose, in order to detect a signal that indicates a significant change in the state of the software.

Introduction: Shewhart's Control Charts

Every process displays variation. Some processes display controlled variation and others display uncontrolled variation.

In the 1920s, Control Charts were invented by Walter A. Shewhart. In most of the rest of the 20th century the concepts were popularized by people such as W. Edwards Deming. The driver for this development as explained by Shewhart:

"The engineer desires to reduce the variability in quality to an economic minimum. In other words, he wants (a) a rational method of prediction that is subject to minimum error, and (b) a means of minimizing variability in the quality of a given produce at a given cost of production" [1].

Over several decades, we have found that software metrics commonly follow a Rayleigh curve [2]. This results in a very different situation from the typical use of control charts, where the process being measured is expected or desired to have a consistent output each time, every time.

In this paper, I describe the use of control charts during testing phases of software development projects. This use is not to determine if the testing is in control, nor is it in order to improve product quality (although that has also been done [3, 4]), but rather to determine when there has been a shift in quality. This is in order to improve mapping of project progress to forecast curves and thereby improve estimates.

Although some purists may raise objections to this application of control charts, I would quote Shewhart himself:

"…In other words, the fact that the criterion which we happen to use has a fine ancestry of highbrow statistical theorems does not justify its use. Such justification must come from empirical evidence that it works" [5].

Therefore, let us look at a typical example.

Using Control Charts to Detect Signals

In order to improve defect forecasts, I use Individuals and Moving Range charts (XmR). This is a type of control chart that is suitable for most real time situations, including the collection of periodic data such as defects detected in a given time period (such as week or month). The Individuals chart has each value plotted in time order. The Moving Range chart, on the other hand, plots the short-term variation from one period to the next.

While most signals appear on the individuals chart, it is good practice to look at the moving range chart as well, as some signals will only show up on it.

Control limits are based on the long-term average value as well as the average moving range value of one point to the next. It is important to calculate control limits correctly in order to not miss valid signals. The appropriate formulas can be found in select books [6, 7] and are also built into statistical tools such as SPSS, Minitab and SAS.

Control limits provide a signal of sporadic or chronic problems. For tracking defects, however, the signal we are looking for is a change in the underlying quality of the software product. Hopefully, this will be a signal of an improvement and not a signal of a problem!

Rules

There are a number of rules that are used to detect signals. The number of rules used and the definitions of the rules vary slightly from one source to another. However, the traditional use of control charts is best met by keeping the number of rules to a minimum, thereby reducing the chance of obtaining a false signal. In this application, however, it is more important to err on the side of obtaining a false signal rather than missing a true signal. All uses of control charts walk this decision line. Shewhart originally used 3 sigma limits because he wanted to minimize false signals, which would incur the unnecessary cost of researching a problem that didn't exist. In other words, when he saw a signal he wanted to be almost completely certain it was real.

In IBM SPSS 22, for example, there are 11 possible rules that can be turned on or off:

- One point above +3 Sigma, or one point below -3 Sigma
- 2 out of last 3 above +2 Sigma, or 2 of 3 below -2 Sigma
- 4 out of last 5 above +1 Sigma, or 4 out of 5 below -1 Sigma
- 8 points above center line, or 8 below center line
- 6 in a row trending up, or 6 trending down
- 14 in a row alternating up and down

Example Control Charts

In Figure A, weekly defects detected are plotted. All the SPSS rules are turned on. If the defect detection rate has changed significantly, that would show up as a special cause signal in the control chart. In this example, the balance between testing and fixing has not remained constant. Five of the points show up as red, meaning they violated one of the rules (see Table 1).

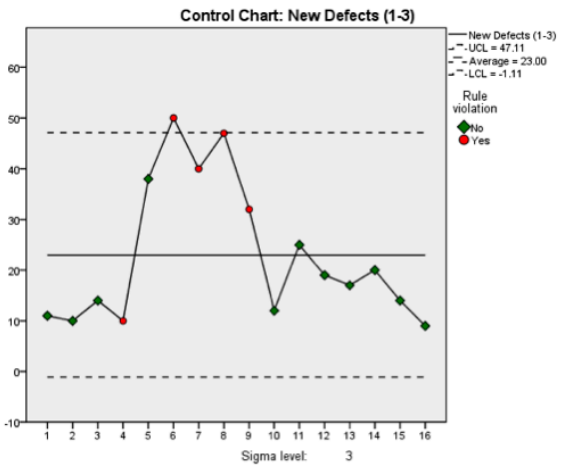


Figure A: Control Chart Example

Point #	Violations for Points
4	4 points out of the last 5 below -1 sigma
6	Greater than +3 sigma
6	2 points out of the last 3 above +2 sigma
7	2 points out of the last 3 above +2 sigma
8	4 points out of the last 5 above +1 sigma
9	4 points out of the last 5 above +1 sigma

Table 1: Rule violations

What can we surmise from this? These violations are not unusual. As mentioned previously, defect metrics commonly follow a Rayleigh distribution. In Figure B, actual defects detected monthly are overlain on a defect forecast based on the current project plan (a parametric SLIM Control forecast based on historical defect rates and project type, size, staff, and duration). We can see the peak detected as a set of rule violations falls in line with the peak of the Rayleigh curve.

One important question is whether the drop in defects in the last few weeks is a signal that a turning point on the project has been reached, as the Rayleigh curve suggests. Control charts can be used to help verify that the signal is real and not random noise.

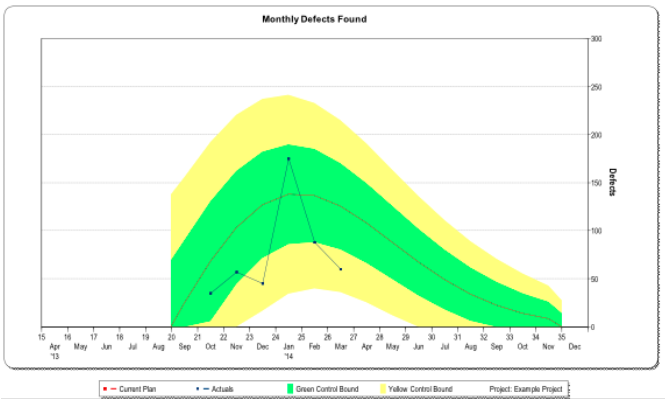
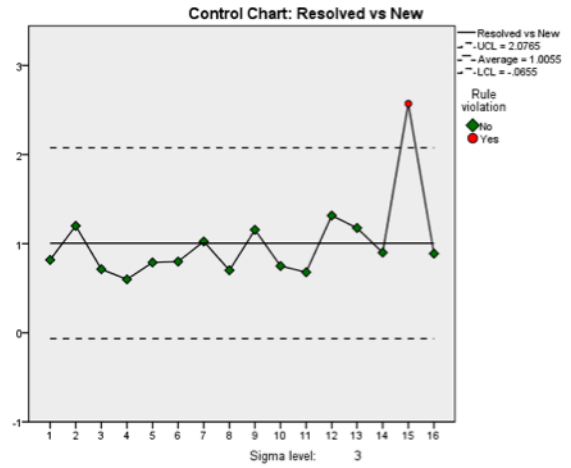
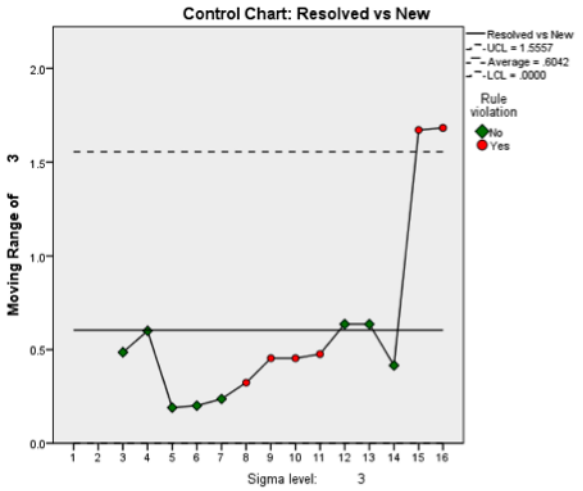


Figure B: Rayleigh Example





Figures C and D: Individual and Moving Range Chart

Figures C and D are Individual and Moving Range charts for the ratio of defects discovered to defects resolved. This ratio measures the balance between defect detection in testing and defect repair. Values in the individuals chart greater than 1 indicate more defects were resolved than were detected during that week. Both charts show rule violations. Point 15 provides evidence that the balance has shifted, supporting the conclusion that the project is truly on the downslope of the Rayleigh Curve.

Defect Prediction

The plan in Figure B was based on parametric estimating. It is possible to create very useful estimates of defects based on only a few key metrics. For example, I created a regression analysis to predict defects based on over 2000 recently completed software projects from the QSM database. This resulted in an adjusted R square of .537 using only the input variables the log of peak staff, the log of ESLOC, and the log of production rate (ESLOC per calendar month). The output variable is log of defects. (Why logs? For the explanation, see [8].) The standardized residuals are plotted on a histogram in Figure E. As can be seen, the residuals have a normal distribution with mean close to zero. The model is not skewed.

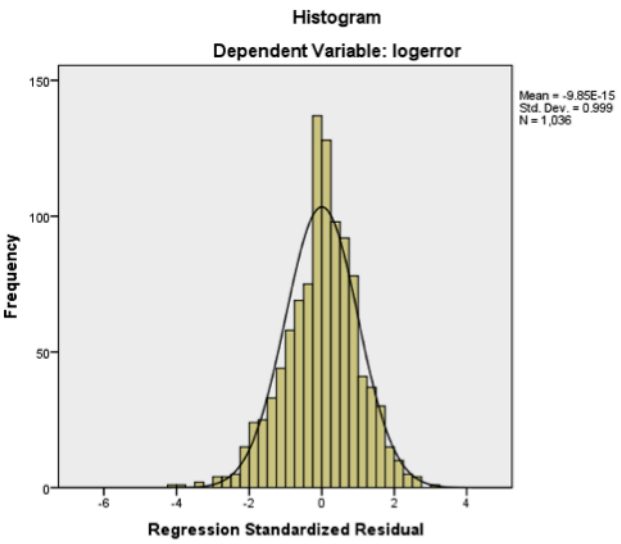


Figure E: Standardized Residuals

Large projects have multiple testing phases. Such models, with multiple control charts, can be used throughout. For example, with one Fortune 500 client, I found that merely using the number of prerelease defects was an excellent predictor of their go live release defects (R Square of over 0.7).

Summary

Control charts can be used to determine whether apparent changes in defect rates are significant. One use for this knowledge is to create and improve forecasts of Program completion, or software quality at key Program milestones. Shewhart gave us this thought regarding updating forecasts:

"…since we can make operationally verifiable predictions only in terms of future observations, it follows that with the acquisition of new data, not only may the magnitudes involved in any prediction change, but also our grounds for belief in it" [9].

ABOUT THE AUTHOR

Paul Below has over 30 years of experience in technology measurement, statistical analysis, estimating, Six Sigma, and data mining. He is a Principal Consultant with Quantitative Software Management, Inc. (QSM) where he provides clients with statistical analysis of operational performance, process improvement and predictability.

This is his second article for Crosstalk, and he is co-author of the IFPUG Guide to IT and Software Measurement (CRC Press, 2012). He has developed courses and been an instructor for estimating, Lean Six Sigma, metrics analysis, function point analysis, and also taught metrics for two years in the Masters of Software Engineering Program at Seattle University. He has presented papers at a dozen industry conferences.

Paul is a Certified SLIM Estimation Professional, and has been a Certified Software Quality Analysis and a Certified Function Point Analyst. He is a Six Sigma Black Belt, and has one US Patent. He is a member of IEEE, the American Statistical Association (ASA) and the National Defense Industrial Association (NDIA).

Quantitative Software Measurement, Inc. (QSM)
<http://www.qsm.com>
Phone: 800-424-6755, 703-790-0055
E-mail: paul.below@qsm.com

REFERENCES

1. Statistical Method from the Viewpoint of Quality Control, Walter A. Shewhart. Dover, 1986 edition, p.9.
2. Five Core Metrics: The Intelligence Behind Successful Software Management, Lawrence H. Putnam and Ware Myers. Dorset House, 2003, Chapter 13.
3. Why CMMI Maturity Level 5?, Michael Comps. Crosstalk, Jan-Feb, 2012, pp 15-18.
4. Do Not Get Out of Control: Achieving Real-time Quality and Performance, Craig Hale and Mike Rowe. Crosstalk, Jan-Feb 2012, pp. 4-8.
5. Economic Control of Quality of Manufactured Product, 50th Anniversary Commemorative Reissue, Walter A. Shewhart. ASQC, 1980, p. 18.
6. Understanding Statistical Process Control, Donald J. Wheeler. SPC Press, 2010.
7. Implementing Six Sigma: Smarter Solutions Using Statistical Methods, Second Edition, Forrest W. Breyfogle III. John Wiley & Sons, 2003.
8. The IFPUG Guide to IT and Software Measurement: A Comprehensive International Guide, IFPUG, ed. CRC Press, 2012. Chapter 17, Paul Below, pp. 319-333.
9. Statistical Method from the Viewpoint of Quality Control, Walter A. Shewhart. Dover, 1986 edition, p.104.

Agile Surveillance Points: An Alternative to Milestone Reviews

Dick Carlson, Agile & Lean Education Associates (ALEA)

Abstract. Since the advent of applying Agile [1] practices on DoD projects, specifically Scrum [2] practices, the burning question asked by many has been, "How does Agile integrate with Waterfall [3] milestones?"

Agile development is by definition incremental, meaning major functionality is completed and demonstrated to the customer, users, and other stakeholders at specified releases and through a series of short, scheduled iterations (or sprints).

Waterfall projects rely on completing all requirements by the Preliminary Design Review (PDR) milestone, completing all design by the Critical Design Review (CDR) milestone, and completing product development by the Test Readiness Review (TRR) milestone. Some success has been made to abate this issue by breaking the typical waterfall-based milestone reviews into multiple interim reviews enabled by the flexibility in the DoD 5000 acquisition regulation requirements. Interim reviews reflect the iterative nature of Agile as they can be held more frequently and with tighter focus than the traditional days-long milestone reviews.

This article presents an approach that uses surveillance points throughout the project lifecycle as the basis for technical interchange meetings (TIMS) each of which includes a demonstration and review of completed functionality that can be deployed as an operational component of the final product. When deployed into the targeted domain, users are encouraged to provide reflections of product usage in the form of feedback to the project team in real time.

Introduction

Agile Software Development is not a fad, methodology, or process. Rather, Agile [2] is a mind-set that applies values and principles to the practices implemented by motivated and trained practitioners. The origination of Agile dates back to February 2001, when 17 industry software development experts got together in search of a common ground of software development techniques. After three arduous days of talking, skiing, and relaxing, they created the Manifesto for Agile Software Development [4]. The declaration proclaimed by the manifesto was aimed at restoring balance and credibility, to provide hope of success in the new economy, and to move aggressively into the era of e-business, e-commerce, and the web. The manifesto represented a beacon of light for companies who were burdened with their Dilbert manifestations of make-work and hidden policies that could change their antiquated and wasteful ways.

NOTE: This paper is based on a presentation given by Dick Carlson at the 2012 Software Technology Conference in Salt Lake City, UT.

Domination of the Waterfall Model

The Waterfall Model has been a well-known approach to software development projects for nearly 60 years. In short, the Waterfall Model is a sequential design process where progress flows steadily downwards (as in a waterfall) through Analysis, Design, Construction, Test, and Production phases of a project. The Waterfall Model is attractive to project management because of its distinct and sequential phases, has an appealing air of simplicity, and the use of easily tracked milestones. The Waterfall Model may have been used to build the pyramids, and it continues to work very well when the problem space is very well defined and few or no changes are anticipated. Ambiguity and changes cause turmoil to Waterfall management.

However, the model has some significant drawbacks. For example, development is done in isolation from other important groups; overhead for technical reviews is costly; it aggravates complexity overload and "analysis paralysis"; it pushes many high-risk and difficult elements toward the end of a project; it does not accommodate uncertainty or changing requirements well; it does not yield a working version of the product until late in the process; and it does not contain intrinsic feedback loops. Waterfall milestones assume stable requirements and design early and focus heavily on documentation and formal reviews. Changes in scope typically result in significant impacts to design and construction.

Milestone Reviews and Surveillance Points

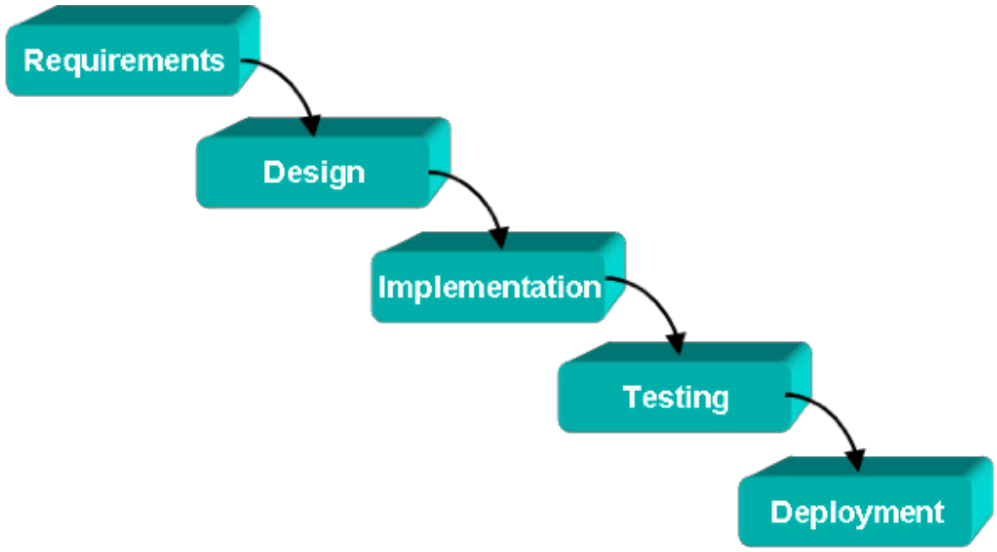
To understand why surveillance points are used, a few words need to be said about milestones. Milestones in projects that use the Waterfall Model establish baselines after the approval of a milestone review (see Figure 1).

Baselines typically consist of the customer's approval of requirements adequacy, design stability, implementation completeness, and test readiness. Approval signifies an endorsement of work completed to that point. Such interchanges are infrequent and costly to both the customer and the contractor. Tremendous effort is expended on preparation of material that involves a significant number of people who are charged with gathering data, and then preparing presentations and other work products that will be reviewed, presented, and discussed during the review. This activity takes time away from doing the work contracted. Milestone reviews can last one day to all week, depending on the importance, complexity, or size of the program.

As a feasible alternative to Waterfall milestones, consider Agile Surveillance Points, which are intervals of the lifecycle where customers, users, and other key stakeholders have opportunities to observe product development progress. On Agile projects, these junctures are the last day of every sprint (a.k.a. iteration) and at the closure of every release, both of which occur frequently throughout a project's lifecycle.

At the end of each sprint, at Sprint Review is conducted where all stakeholders have an opportunity to observe work that the project team has completed. Completed functionality is demonstrated and other completed work products and how they related to the product capability or performance is mentioned.

Figure 1: Waterfall Model



The end of each sprint series is followed by a release where the completed functionality of the product being developed, including other relevant work products such as test cases and documentation, are demonstrated within a controlled or simulated environment.

"Wait, There's More!"

For projects that are required to use the Waterfall model, a project can implement the Agile project management approach, Scrum, to work within the more traditional method. When this approach is used, commonly referred to as a "hybrid" Agile approach, certain release demonstrations and reviews may be planned in advance as formalized releases.

The formal release demonstration/review can be accommodated quite easily through the conduct of technical interchange meetings (TIMs) and used as customer "acceptance points." where the customer evaluates whether the product is meeting their requirements and is being developed according to their expectations. The customer may also evaluate whether the project team is making sufficient progress to the schedule and ultimately determine if the project team should continue working according to plan, or make adjustments or corrections prior to proceeding any further. (See Figure 2.)

The number of formal demonstrations/reviews must be agreed upon by the customer and the project team. Formal demonstrations or reviews targeted to be formal in nature must be documented and scheduled with customer approval. During the project planning phase (otherwise known as Sprint 0, entry criteria for every planned formal demonstration or review must be established and well-understood by all.

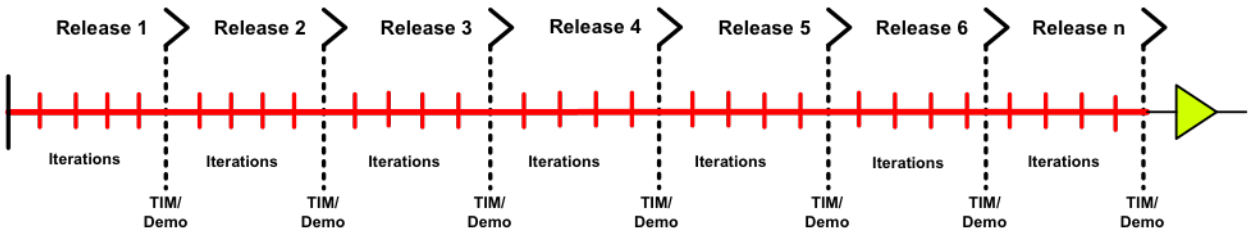
Light at the End of the Tunnel

To put things into perspective, at the start of every release cycle or after approval of the project or next stage of the project, the Product Owner (the project visionary) and others discuss and plan a solid release strategy. From the project's Product Roadmap, features and functionality that address the customer's real needs and value are decomposed into smaller chunks to enable the team to more accurately plan the most important work first. Such items are listed in prioritized order in the Product Backlog. Priority of items must be customer-driven to maximize business value. At this point, the team provides gross or high-level estimates to determine scope and project duration.

From this point onward and upon the start of the project, the stakeholder's surveillance of the product's development and progress is highly encouraged. In a technical paper published by the Software Engineering Institute [5], it was suggested that surveillance in the form of formal reviews such as preliminary and critical design reviews (e.g. PDR and CDR) be reformatted into interim design reviews.

Interim design reviews provide opportunities for key project stakeholders to observe product development progress through more frequent and active participation than that of the traditional PDRs and CDRs that is costly in preparation and travel and involves more people than necessary.

Figure 2: Agile Surveillance Points model



Conclusion

The use of surveillance points on Agile projects enables project stakeholders to observe product development from sprint to sprint and release to release. Milestones in projects that use the Waterfall Model establish baselines after the approval of a milestone review. The Waterfall Model is attractive to project management because of its distinct and sequential phases, has an appealing air of simplicity, and the use of easily tracked milestones. Agile Surveillance Points are intervals of the lifecycle where customers, users, and other key stakeholders have opportunities to observe product development progress.

For projects that are required to use the Waterfall model, a project can implement the Agile project management approach, Scrum, to work within the more traditional method. When this approach, commonly referred to as a "hybrid" Agile approach is used, certain release demonstrations and reviews may be planned in advance as formalized releases. Interim design reviews provide opportunities for key project stakeholders to observe product development progress through more frequent and active participation than that of the traditional PDRs and CDRs that is costly in preparation and can involve more people than necessary.

REFERENCES

1. <http://www.agilealliance.org/>
2. <http://www.scrumalliance.org/>
3. <http://en.wikipedia.org/wiki/Waterfall_model>
4. <http://agilemanifesto.org/>
5. Software Engineering Institute Technical Note, CMU/SEI-2010-TN-002

ABOUT THE AUTHOR

Dick Carlson has an extensive engineering background that includes many years of practical knowledge and hands-on experience in the implementation and deployment of Agile, Lean, and Scrum values and principles in communications-electronics, and software and systems engineering within the aerospace, DoD, IT, and industry domains. He has developed and actively conducted comprehensive training courses for Scrum Teams, Scrum Masters, Product Owners, project/program managers, customers, executives, organizational leaders, and others interested in learning how to implement and deploy Agile, Lean, and Scrum. Mr. Carlson has a Bachelor of Science degree in Business and Management from the University of Maryland, and is a Certified Scrum Professional, Certified Scrum Master, and Certified Scrum Product Owner.

Agile & Lean Education Associates (ALEA)
<http://www.a2zalea.com>
info@a2zalea.com
Phone: 714-350-9946
E-mail: dcarlson@iascar.us

Upcoming Events

Visit <http://www.crosstalkonline.org/events> for an up-to-date list of events.

Association of Software Testing: The Art of Science and Testing
August 11-13, 2014 New York City
http://www.associationforsoftwaretesting.org/conference/cast-2014/

SIGCOMM'14 – ACM SIGCOMM 2014 Conference
17-22 Aug 2014
Chicago, Illinois
http://www.sigcomm.org/events/sigcomm-conference

SEFM- Software Engineering and Formal Methods
1-5 Sept 2014
Grenoble, France
http://sefm2014.inria.fr

Defense Systems Acquisition Management Course
8-12 Sep 2014
Dallas, TX
E-mail: adekleine@ndia.org or call 703-247-2599

APPSEC USA 2014
16-19 Sept 2014
Denver, Colorado
http://2014.appsecusa.org/2014/

International Conference on Software Engineering and Technology
17-18 September 2014
Paris, France
http://www.icste.org/

Fall 2014 Software and Supply Chain Assurance (SSCA) Forum
Theme: Improving Cybersecurity and Resilience Through Acquisition
23 - 25 Sept 2014, Washington, DC
The event is free and co-sponsored by DoD-DHS-NIST-GSA
http://gsa.gov/portal/content/194963

QSIC 2014 Int. Conf. on Quality Software
Dallas, Tx
October 2-3, 2014
http://paris.utdallas.edu/qsic14

APCOSEC 2014
7-9 Oct 2014
Qingdao, Shandong, China
http://www.apcose.org/

Star West Breaking Software Conference
12-17 Oct 2014
Anaheim, CA
http://starwest.techwell.com/

SEDE 2014: The 23rd International Conference on Software Engineering and Data Engineering
New Orleans, Louisiana
13-15 Oct 2014
http://www.wikicfp.com/cfp/servlet/event.showcfp?eventid=33994&copyownerid=9837

System Safety and Cyber Security 2014 Conference
14th to 16th October 2014
Manchester, United Kingdom
http://conferences.theiet.org/system-safety/index.cfm?origin=conference-alerts

International conference on Computer Science and Information Systems (ICSIS'2014)
17-18 Oct 2014 Dubai (UAE)
http://www.iieng.org/2014/10/18/53

World Congress on Engineering and Computer Science 2014 Conference
22nd to 24th October 2014
San Francisco, CA
http://www.iaeng.org/WCECS2014

17th Annual Systems Engineering Conference
27-30 Oct 2014
Springfield, VA
E-mail cbriggs@ndia.org or call 703-247-2570

Better Software Conference East
9-14 Nov 2014
Orlando, Fl
http://bsceast.techwell.com/

2014 T3 Conference - Enterprise Edition
Hilton - Downtown Atlanta, GA
Nov. 11-13, 2014
http://www.t3enterpriseconference.com/

International Conference on Software Quality
March 9-11, 2015
Long Beach, CA
http://asq-icsq.org/index.html

Unfit for Use

(Unless You Perform a Lot of Maintenance!)

I work best under a deadline. Which is why Heather Giacalone, the awesome CrossTalk Article Coordinator, has learned to not let me know that a BackTalk article is due until right before she needs it. Give it to me too early and I end up writing an inferior article, but then beleaguer Heather with numerous "updates," "minor improvements," and frequently, an "ignore earlier submission, here's a different one." Wait until it's due, add the last-minute stress, and out pops a BackTalk column.

You would think, however, that I would sort of know when two months pass on a calendar, and start thinking of what to write. Not this time. I was at a total loss for a topic until the phone rang yesterday while I was driving. It was my pharmacy, letting me know that my prescription was ready for refill. Except the process was less than smooth. First, the automated messaging system asked me to, "Verify my identity by entering my birthday in the format MMYYYY." At least, that's what I think it said. I was busy juggling my cell phone, trying to put it on speaker mode, and bring up the keypad to enter the numbers. Because I started this before the automated message finished, I didn't hear (and couldn't remember) if I was supposed to hit the pound key afterwards. Obviously not, because after I entered my birthday, nothing seemed to happen, so I hit the # key just in time to hear, "Invalid entry. Please enter the date again." I reentered the date, did not hit the # key, and eventually got a message saying, "Your automatically renewed prescriptions are ready for pickup."

Why couldn't the message just have said, "The individuals associated with this phone number have prescriptions ready to pickup"? If I was sitting at home or work, using a landline phone, entering birthdays using the touchtone pad is pretty easy. On a smartphone? Not easy at all, especially if you are driving.

I bet this system worked great about 15 years ago when nobody had a cell phone. Not now. The system has become obsolete. It's "unfit for use." It's not unfit for use due to bad programming, but due to a changing environment and changing user needs. The system has become outdated— just one step away from obsolete.

As most good software developers know, maintenance is the hard part of keeping a system running. It's hard work that is typically unappreciated. It is, however, probably the most important part of any long-lived system.

I teach software engineering, and the maintenance paradigm I teach involves three components:

1. **Adaptive Maintenance** (keeping the system current for changing environments and user needs—hopefully in advance, but usually at the last minute, and in a panic)
2. **Corrective Maintenance** (fixing errors)
3. **Perfective Maintenance** (a mythical time wherein developers have some spare time, and they take a working system and make it perform faster or more reliably)

Let's ignore perfective maintenance (most systems do!) as it's discretionary. Perfective maintenance is for "when you get it working." As in "…let's just use this quick-and-dirty algorithm, and we'll go back and fix it and do it right when we get it working." Nice idea, but it seldom happens. Adaptive and corrective maintenance are more important.

I would estimate that up to 75% of all maintenance is corrective, but the word "corrective" covers a lot of ground. Didn't have time to put in some functionality during development? Mark it as an error, and call it corrective maintenance. It's not right, but it's the way we get software out the door within a reasonable time frame.

Unfortunately, lots of corrective maintenance is just plain fixing errors. I ran across an excellent online article called "The First Rule of Programming: It's Always Your Fault" that states part of being a good programmer is recognizing your mistakes, and fixing them. Unfortunately, just fixing errors does not give you a long-lasting system. For that, you sometimes have to take a working system and change it to meet rapidly evolving environmental changes (think new Operating System or new hardware) or user needs. This requires adaptive maintenance.

Adaptive maintenance—now that's the money-maker. The difference between a system that lasts 1 year and one that lasts 10 years is the ability to modify the system to meet changing user needs and changing environments. Used to run on DOS? Then Windows 3.1? Then 95, 98, ME, 2000, XP, Vista, 7 and now 8? You must have some good maintenance programmers. Has your radar systems survived 10 different airplane bus upgrades? Kudos to you!

The aforementioned pharmacy system is unfit for use due to lack of adaptive maintenance. User needs (and operating conditions) changed, and they did not adapt the system to match. Most people today use a smart phone of some type. You can't assume it's easy to have a keypad available without some fumbling. Voice recognition, while frustrating at times, would work better.

Some systems just stumble along, never really working well, and users sort of pretend to not notice the unfitness of the system. My pharmacy's telephone automated refill system is horrible—but then, I use the online system and avoid the telephone system at all costs.

Still, how hard could it be to put in a simple voice recognition system? (And understand, I have no idea how hard that would be. That's the problem with maintenance. Changes sound so simple to the naïve non-developer, don't they? But that's another column!)

David A. Cook, Ph.D.
Stephen F. Austin State University
(and former maintenance programmer)

CrossTalk / 517 SMXS MXDED
6022 Fir Ave.
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737