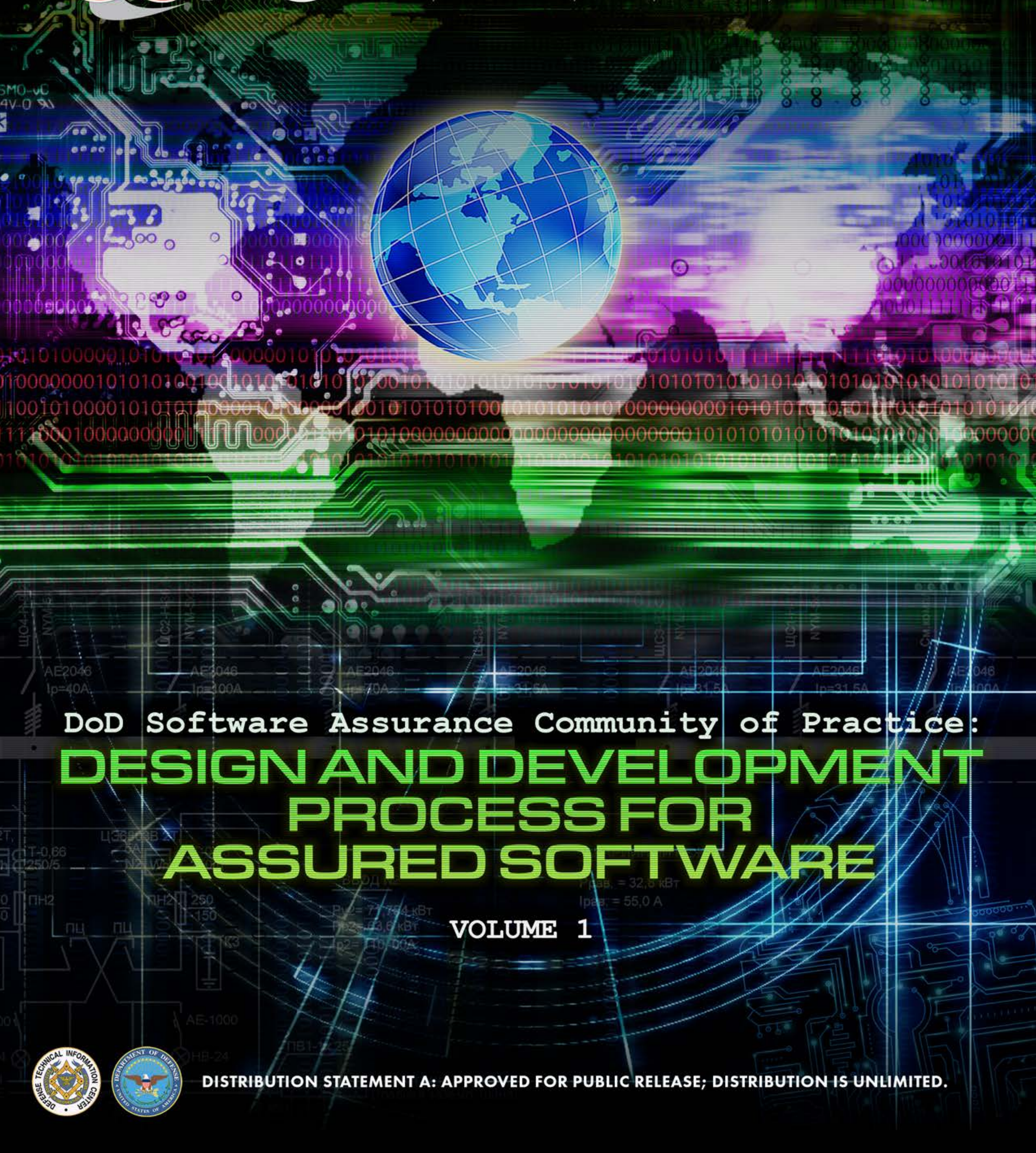
VOLUME 5 NUMBER 2
JULY 2017

CSIAC JOURNAL

A Quarterly Publication of the Cyber Security & Information Systems Information Analysis Center



DoD Software Assurance Community of Practice:

DESIGN AND DEVELOPMENT PROCESS FOR ASSURED SOFTWARE

VOLUME 1

CSIAC Cyber Security & Information Systems Information Analysis Center

ABOUT THE CSIAC

As one of three DoD Information Analysis Centers (IACs), sponsored by the Defense Technical Information Center (DTIC), CSIAC is the Center of Excellence in Cyber Security and Information Systems. CSIAC fulfills the Scientific and Technical Information (STI) needs of the Research and Development (R&D) and acquisition communities. This is accomplished by providing access to the vast knowledge repositories of existing STI as well as conducting novel core analysis tasks (CATs) to address current, customer focused technological shortfalls.

OUR MISSION

CSIAC is chartered to leverage the best practices and expertise from government, industry, and academia in order to promote technology domain awareness and solve the most critically challenging scientific and technical (S&T) problems in the following areas:

- Cybersecurity and Information Assurance
- Software Engineering
- Modeling and Simulation
- Knowledge Management/Information Sharing

The primary activities focus on the collection, analysis, synthesis, processing, production and dissemination of Scientific and Technical Information (STI).

OUR VISION

The goal of CSIAC is to facilitate the advancement of technological innovations and developments. This is achieved by conducting gap analyses and proactively performing research efforts to fill the voids in the knowledge bases that are vital to our nation. CSIAC provides access to a wealth of STI along with expert guidance in order to improve our strategic capabilities.

WHAT WE OFFER

We provide expert technical advice and assistance to our user community. CSIAC is a competitively procured, single award contract. The CSIAC contract vehicle has Indefinite Delivery/Indefinite Quantity (ID/IQ) provisions that allow us to rapidly respond to our users' most important needs and requirements.

Custom solutions are delivered by executing user defined and funded CAT projects.

CORE SERVICES

- Technical Inquiries: up to 4 hours free
- Extended Inquiries: 5 - 24 hours
- Search and Summary Inquiries
- STI Searches of DTIC and other repositories
- Workshops and Training Classes
- Subject Matter Expert (SME) Registry and Referrals
- Risk Management Framework (RMF) Assessment & Authorization (A&A) Assistance and Training
- Community of Interest (COI) and Practice Support
- Document Hosting and Blog Spaces
- Agile & Responsive Solutions to emerging trends/threats

PRODUCTS

- State-of-the-Art Reports (SOARs)
- Technical Journals (Quarterly)
- Cybersecurity Digest (Semimonthly)
- RMF A&A Information
- Critical Reviews and Technology Assessments (CR/TAs)
- Analytical Tools and Techniques
- Webinars & Podcasts
- Handbooks and Data Books
- DoD Cybersecurity Policy Chart

CORE ANALYSIS TASKS (CATS)

- Customer tailored R&D efforts performed to solve specific user defined problems
- Funded Studies - $500K ceiling
- Duration - 12 month maximum
- Lead time - on contract within as few as 6-8 weeks

CONTACT INFORMATION

266 Genesee Street
Utica, NY 13502

1 (800) 214-7921

info@csiac.org

/DoD_CSIAC

/CSIAC

/CSIAC

DoDIACs Department of Defense Information Analysis Centers

Distribution Statement
Unclassified and Unlimited

ABOUT THIS PUBLICATION

The Journal of Cyber Security and Information Systems is published quarterly by the Cyber Security and Information Systems Information Analysis Center (CSIAC). The CSIAC is a Department of Defense (DoD) Information Analysis Center (IAC) sponsored by the Defense Technical Information Center (DTIC) and operated by Quanterion Solutions Incorporated in Utica, NY.

ARTICLE REPRODUCTION

Journal of Cyber Security and Information Systems

Design and Development Process for Assured Software - Volume I

INTRODUCTION

DESIGN AND DEVELOPMENT PROCESS FOR ASSURED SOFTWARE - VOLUME 1

By: Michael Weir

Welcome to this special Software Assurance (SwA) edition of the Journal of Cyber Security & Information Systems, published by the Cyber Security & Information Systems Information Analysis Center (CSIAC).

Software is ubiquitous. It is at the core of every deployed critical system in the DoD (and our society for that matter). As our systems become more complex and the software that supports these systems explodes in size, our adversaries are presented with an ever increasing attack surface which they have repeatedly demonstrated the capability to exploit. The need to gain confidence that this software is free from exploitable vulnerabilities and malicious behavior has never been more important. Gaining confidence — that is "assurance" — in software is more than simply testing the software to show correct functionality or running tools against the code to identify known flaws; it requires an acquisition and development discipline augmented with technology, supported by sound policy, measurement practices, and deployment processes that achieve the necessary confidence our systems are fit to protect our country's most valuable assets.

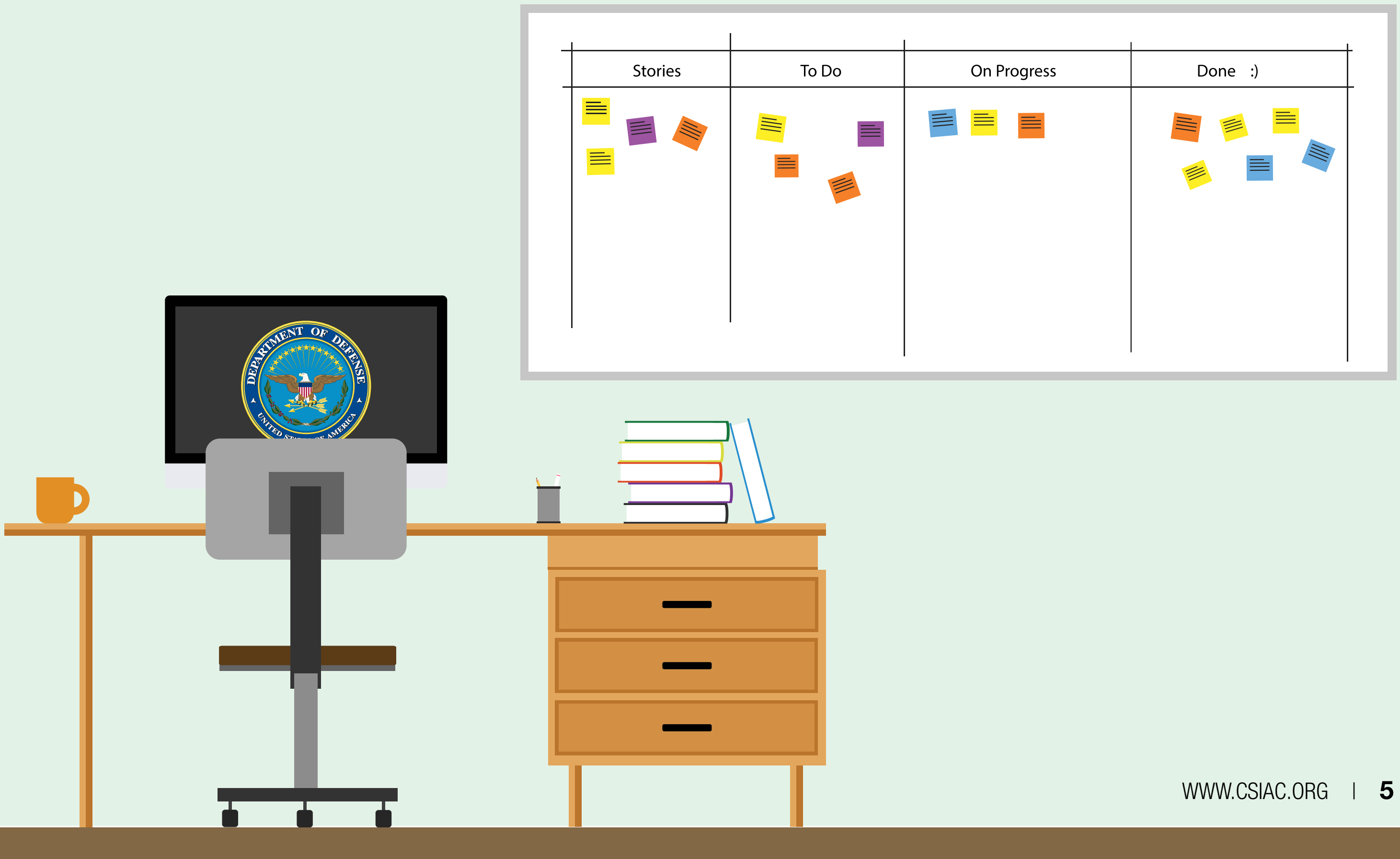
Although it is easy to acknowledge that "assured software" is a critical national priority, we still do not hold strong examples of truly securely designed, implemented and deployed assured software. To date, the DoD has not demonstrated a full understanding of the shape of the field that underlies the process of producing, sustaining and acquiring secure software. Decision makers often have trouble "connecting the dots" among the detailed, disparate data available from interactively complex systems. As a result, they can find it difficult to understand a system's macro-level behavior and the risks that their deployed software faces. Over the past 20 years, the rules of the game have changed – building software without accounting for security is no longer an acceptable risk.

This edition explores different aspects of developing, deploying and training on how to build assured software. Articles are contributed by software assurance practitioners from the DoD and civil government that are devoted to the advancement of secure development principles in U.S government critical systems. We hope that you can get a flavor of some of the exciting things happening in this space, identify some principles that will increase your software assurance posture and find opportunities to connect with key players in the community to support your assured software development/acquisition process. ■

KEYS TO SUCCESSFUL DoD SOFTWARE PROJECT EXECUTION

By: Joe Heil, Naval Surface Warfare Center Dahlgren Division (NSWCDD)

Software is inherent in today's complex systems and is often the primary cost, schedule, and technical performance driver in Department of Defense (DoD) programs. For DoD mission critical systems, the associated software size, complexity, interdependencies, reliance-on for mission and safety critical functionality, and software assurance (high quality and free from vulnerabilities) related challenges are all continuing to rapidly increase. Successful software project execution for DoD mission critical programs is vital to maintaining our national security. This in turn requires the ability to efficiently as possible develop and deliver high quality software systems that fully meet the warfighter's operational needs and that are safe, secure, reliable, maintainable and scalable. There are many reports from various software acquisition and performance assessment organizations such as the Defense Science Board (DSB), Government Accounting Agency (GAO), and the Carnegie Mellon University Software Engineering Institute (SEI) that document the common challenges that have contributed to the inconsistent execution of DoD software system projects. This paper provides a brief high-level introduction to some of the proven key approaches and techniques required for successful software project execution.



In general, the primary reason for software project failure is often not due to the lack of technical expertise by the software development engineers; but rather it is due to poor project estimation, planning and control. Per the referenced reports above, some of the most commonly reported reasons for software project execution failure include:

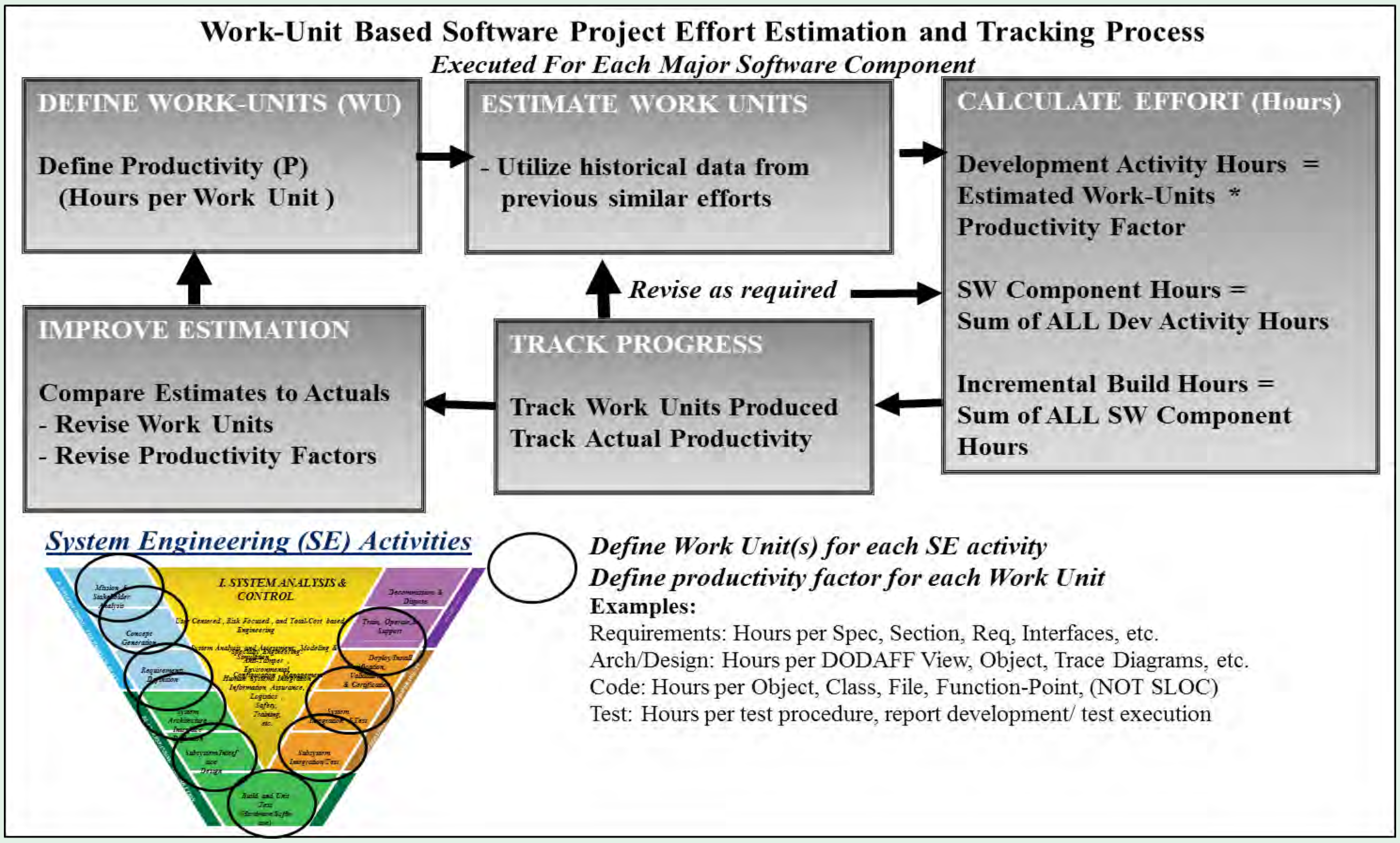
- Poor software effort cost and schedule estimation and tracking
- Poor requirements and interface management
- Limited data-driven execution control and continuous improvement
- Limited awareness and enforcement of best software engineering practices
- Lack of integrated software assurance techniques in all development phases
- Lack of frequent, regular, and structured cross-discipline communication
- Lack of formal risk management
- Lack of investment in automated testing, simulations, and data-extraction
- Lack of multi-mission-platform capable software architectures
- Program leaders lacking applied software engineering experience and expertise
- Over-reliance on private industry for system and software development
- Failure to apply lessons-learned from previous efforts

Despite the common challenges to success listed above, some DoD software intensive projects have been consistently successful with regards to cost, schedule, technical, quality and operational performance. These consistently successful software projects span a wide range of missions (e.g. strategic, tactical, simulations), development methodologies (e.g. nuclear certified, DoD 5000 waterfall, agile, prototyping), complexity (numerous interfaces), and sizes (multi-million lines of code to tens of thousands); but they all utilize many of the same software system acquisition and development techniques to ensure success. Some of the common keys to software project execution success are addressed in the following sections.

Software efforts are often "hidden" under the "system engineering"

KEY: Data Driven Software Effort Estimation and Tracking

One of the primary root causes for many software project failures is that the effort was poorly estimated; or that the effort was accurately estimated but the program's senior leader drove the development organizations to "accept the challenge" of significantly reduced cost and accelerated schedule without reducing the planned capabilities or requirements. Aggravating and increasing the negative impacts of poor estimation and tracking is the lack of control over requirements and interface volatility.



Work-Unit Based Software Project Effort Estimation and Tracking Process
Executed For Each Major Software Component
DEFINE WORK-UNITS (WU)
Define Productivity (P) (Hours per Work Unit)
ESTIMATE WORK UNITS
- Utilize historical data from previous similar efforts
CALCULATE EFFORT (Hours)
Development Activity Hours = Estimated Work-Units * Productivity Factor
SW Component Hours = Sum of ALL Dev Activity Hours
Incremental Build Hours = Sum of ALL SW Component Hours
Revise as required
IMPROVE ESTIMATION
Compare Estimates to Actuals
- Revise Work Units
- Revise Productivity Factors
TRACK PROGRESS
Track Work Units Produced
Track Actual Productivity
System Engineering (SE) Activities
I. SYSTEM ANALYSIS & CONTROL
Define Work Unit(s) for each SE activity
Define productivity factor for each Work Unit
Examples:
Requirements: Hours per Spec, Section, Req, Interfaces, etc.
Arch/Design: Hours per DODAFF View, Object, Trace Diagrams, etc.
Code: Hours per Object, Class, File, Function-Point, (NOT SLOC)
Test: Hours per test procedure, report development/ test execution

The key to success in estimating software efforts is to establish and maintain detailed historical data on cost, schedule and technical performance. Ideally, software projects should use the historical data from their own programs for effort estimation/validation versus trying to use data from other programs as this is rarely successful given that there are too many variables involved to ensure an apples-to-apples comparison (different tools, team experience, development processes, requirements, constraints, etc.).

Software efforts are often "hidden" under the "system engineering" efforts when planning and controlling a project. Project teams must take the time and make the investment to establish a well-defined work-break-down structure (WBS) that breaks out the various software development activities (requirements, architecture, design, code, integration, test phases, etc.) and includes the ability to develop and track associated productivity factors (development-hours-required per work-unit). Teams must establish the process, tools and discipline to accurately collect and utilize the estimated-vs-actual data. It is NOT recommended to estimate and track coding efforts by source-lines-of-code (SLOC). Higher level work-unit abstractions such as Objects, Files, or Function-points are much better than SLOC. Note that although SLOC based estimates are a poor method for estimating and tracking, it is still important to know the system size as measured in SLOC and for normalizing software quality (calculating defect ratios).

There should be detailed planned vs actual cost and schedule plans for each Computer Software Configuration Item (CSCI). Caution must be exercised in combining the cost and schedule performance indicators for multiple CSCIs. This is because over-performance by one CSCI may mask high risk under-performance by another CSCI. The software cost and schedule plans should be traceable and linked directly into the higher level integrated system level cost and schedule plans.

Program leaders that drive volatility but refuse to trade off other capabilities or extend cost and schedule are destined to fail. A formal Change Management (CM) process must be institutionalized and strictly adhered to. All requirements and associated quantifiable and verifiable Key Performance Parameters (KPPs) must be allocated and traced to architecture, design, code and test organizations and artifacts. Program leaders must resist requirements creep/volatility and changes late in the cycle. All content changes must be accompanied by revised cost, schedule and technical performance impact assessments.

The project leaders and software team lead(s) must review planned content, cost and schedule performance indicators, and technical performance indicators on a very frequent, regular, and structured basis. Variance thresholds must be defined and formal performance risks and mitigation plans must be documented and tracked to closure.

Software assurance (quality AND resiliency against cyber vulnerabilities) must be engineered-in throughout all development activities.

KEY: Data Driven Management and Technical Execution Best Practices

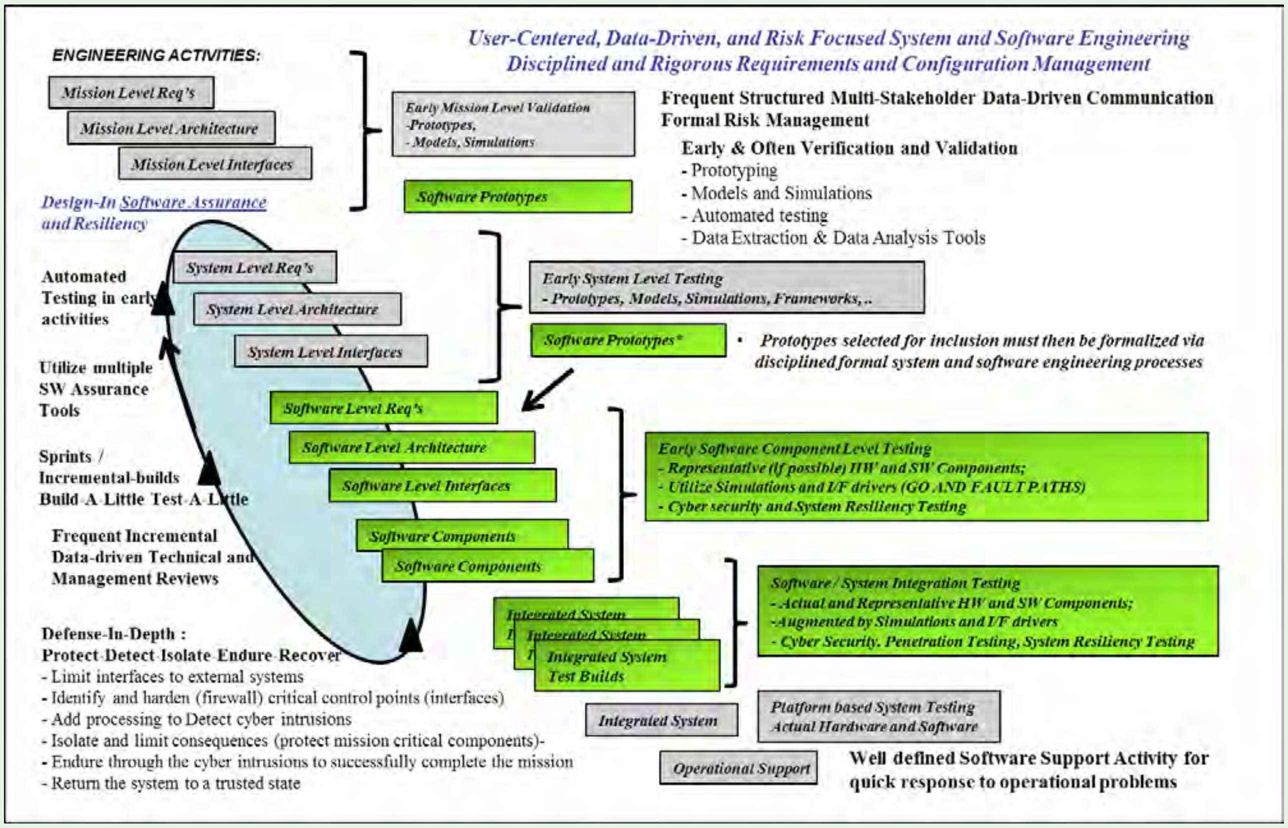
Mature data-driven best software project management and technical engineering practices are required to consistently achieve the goal of delivering high quality, safe, secure, and reliable systems on schedule and within budget.

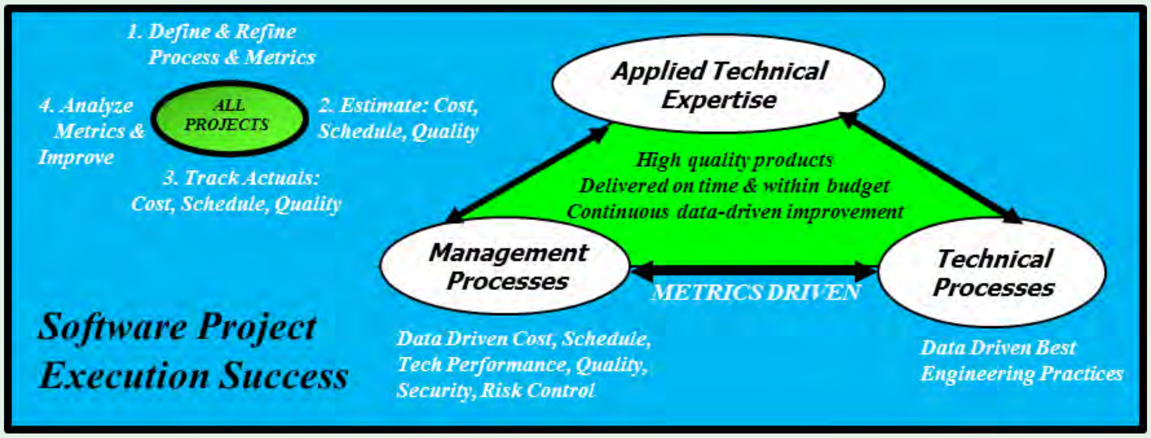
The software project management processes and technical development processes must be documented, institutionalized and enforced. The software development plan must specify the steps, activities, roles and responsibilities, and required reviews and metrics that are used for both the initial system development (pre-IOC) and sustainment (post-IOC) efforts. This includes the set of required metrics and measures-of-success that will be utilized to proactively control cost, schedule, technical performance, quality, and risk for the current effort as well as facilitating analysis and continuous improvement for cost, schedule, technical, and quality performance of future efforts. At a minimum, each software development organization must collect, maintain, **share and report on a frequent, regular and structured basis the quantitative and qualitative information** to address all of the critical execution questions listed below:

1. Are the expected system requirements stable and understood?
2. Is the scope and size of the effort understood?
3. Is the activity adequately staffed?
4. Is the activity making the required progress?
5. Is the activity being executed within budget?
6. Is the activity meeting technical performance, assurance, and quality goals?
7. Is the activity formally successfully identifying and mitigating risks?
8. Is the activity continually improving efficiency and effectiveness?

Continuous improvement requires the software teams to maintain awareness of and apply emergent best practices which include tools, techniques, methods, technologies, etc. For example, a few proven best sw engineering technical practices include:

› User Centered and Model-based system and software engineering.
› Documented traceability between requirements,

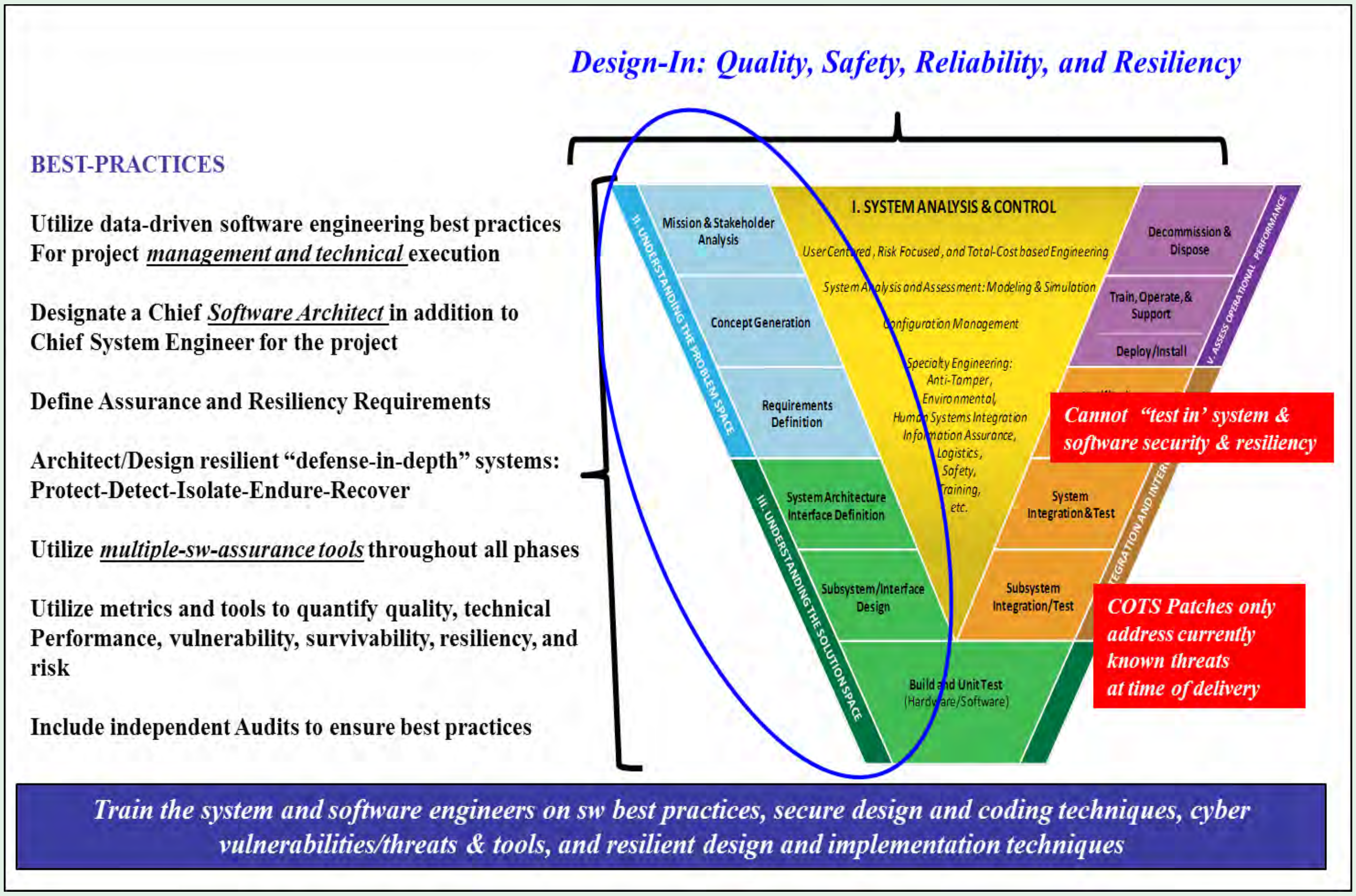

User-Centered, Data-Driven, and Risk Focused System and Software Engineering
Disciplined and Rigorous Requirements and Configuration Management
ENGINEERING ACTIVITIES:
Mission Level Req's
Mission Level Architecture
Mission Level Interfaces
Early Mission Level Validation -Prototypes, - Models, Simulations
Frequent Structured Multi-Stakeholder Data-Driven Communication
Formal Risk Management
Early & Often Verification and Validation
- Prototyping
- Models and Simulations
- Automated testing
- Data Extraction & Data Analysis Tools
Software Prototypes
Design-In Software Assurance and Resiliency
Automated Testing in early activities
System Level Req's
System Level Architecture
System Level Interfaces
Early System Level Testing - Prototypes, Models, Simulations, Frameworks,..
Software Prototypes*
• Prototypes selected for inclusion must then be formalized via disciplined formal system and software engineering processes
Utilize multiple SW Assurance Tools
Software Level Req's
Software Level Architecture
Software Level Interfaces
Early Software Component Level Testing
- Representative (if possible) HW and SW Components;
- Utilize Simulations and I/F drivers (GO AND FAULT PATHS)
- Cyber security and System Resiliency Testing
Sprints / Incremental-builds Build A Little Test A Little
Frequent Incremental Data-driven Technical and Management Reviews
Software Components
Software Components
Software / System Integration Testing
- Actual and Representative HW and SW Components;
-Augmented by Simulations and I/F drivers
- Cyber Security, Penetration Testing, System Resiliency Testing
Integrated System
Integrated System
Integrated System Test Builds
Defense-In-Depth :
Protect Detect Isolate Endure Recover
- Limit interfaces to external systems
- Identify and harden (firewall) critical control points (interfaces)
- Add processing to Detect cyber intrusions
- Isolate and limit consequences (protect mission critical components)-
- Endure through the cyber intrusions to successfully complete the mission
- Return the system to a trusted state
Integrated System
Platform based System Testing Actual Hardware and Software
Operational Support
Well defined Software Support Activity for quick response to operational problems


1. Define & Refine Process & Metrics
2. Estimate: Cost, Schedule, Quality
3. Track Actuals: Cost, Schedule, Quality
4. Analyze Metrics & Improve
ALL PROJECTS
Applied Technical Expertise
High quality products
Delivered on time & within budget
Continuous data-driven improvement
Management Processes
Technical Processes
METRICS DRIVEN
Software Project Execution Success
Data Driven Cost, Schedule, Tech Performance, Quality, Security, Risk Control
Data Driven Best Engineering Practices

design, code and test artifacts.
› Multi-Discipline-expert peer reviews of artifacts (specifications, code, tests, etc.).
› Build-a-Little Test-a-Little (Rapid prototyping, Agile development, etc.).
› Automated testing (at CSCI level) and simulators for go/fault/stress testing.
› Tracking defect detection and removal in each development phase.
› Regular causal analysis of defects to improve earlier detection and removal.

Project teams must take the time to formally and regularly assess their cost, schedule, technical, quality and risk management performance trends and then identify and track to closure the associated specific process improvement actions.

Software assurance (quality AND resiliency against cyber vulnerabilities) must be engineered-in throughout all development activities. This entails much more than applying the latest COTS security patches prior to delivery. SW assurance requirements must be defined, the software design must not only defend against cyber intrusions, but also be resilient enough to detect and complete mission critical functions after intrusion; coders must be trained on and apply secure coding techniques; multiple tools must be integrated into all activities to identify and remove vulnerabilities as early as possible; and all testing phases should include penetration testing.



Design-In: Quality, Safety, Reliability, and Resiliency

BEST-PRACTICES

Utilize data-driven software engineering best practices For project *management and technical* execution

Designate a Chief *Software Architect* in addition to Chief System Engineer for the project

Define Assurance and Resiliency Requirements

Architect/Design resilient "defense-in-depth" systems: Protect-Detect-Isolate-Endure-Recover

Utilize *multiple-sw-assurance tools* throughout all phases

Utilize metrics and tools to quantify quality, technical Performance, vulnerability, survivability, resiliency, and risk

Include independent Audits to ensure best practices

Train the system and software engineers on sw best practices, secure design and coding techniques, cyber vulnerabilities/threats & tools, and resilient design and implementation techniques

I. SYSTEM ANALYSIS & CONTROL
User Centered, Risk Focused, and Total-Cost based Engineering
System Analysis and Assessment: Modeling & Simulation
Configuration Management
Specialty Engineering: Anti-Tamper, Environmental, Human Systems Integration Information Assurance, Logistics, Safety, Training, etc.
II. UNDERSTANDING THE PROBLEM SPACE
Mission & Stakeholder Analysis
Concept Generation
Requirements Definition
III. UNDERSTANDING THE SOLUTION SPACE
System Architecture Interface Definition
Subsystem/Interface Design
Build and Unit Test (Hardware/Software)
Subsystem Integration/Test
System Integration & Test
Deploy/Install
Train, Operate, & Support
Decommission & Dispose
V. ASSESS OPERATIONAL PERFORMANCE
Cannot "test in' system & software security & resiliency
COTS Patches only address currently known threats at time of delivery

KEY: Structured Communication and Formal Risk Management

Open and honest communications up and down the chain of command as well as across the various development stakeholders and organizations is critical for success. Program leaders must create a culture where all team members are empowered and encouraged to identify and proactively communicate risks and associated mitigation techniques and plans. Leaders must not "shoot the messenger" when emergent high severity risks are identified.

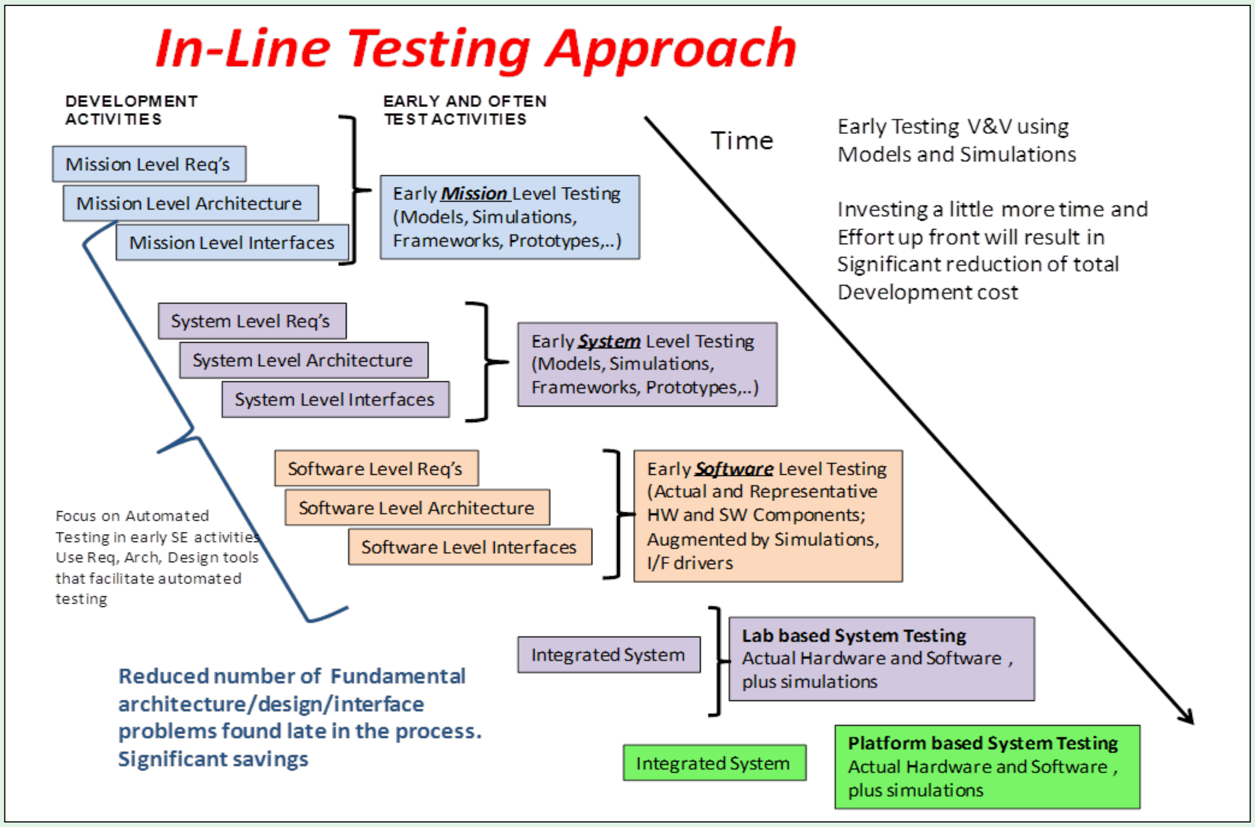
Effective communication often suffers due to poor project estimation. Team members are behind schedule from day one and therefore "do not have time" to communicate on a regular basis with their peers, stakeholders and leaders. Lack of communication and miscommunication due to over-allocated engineers often results in requirements, design and interface problems detected very late in the development cycle which are very costly.

Project teams must establish a set of hierarchical and linked cost, schedule, technical performance indicators and quality measures. Cross discipline and team delivery interdependencies must be identified and closely tracked. All development teams must establish timely regular data-driven reviews to proactively assess and mitigate cost, schedule, technical, and quality performance risks.

A formal risk and opportunities board and process must be established and executed with discipline. The process must facilitate risks being identified and communicated on a frequent, regular interval and at the appropriate levels of leadership. All risks must always be addressed from the three perspectives of cost, schedule and technical performance impact. Risks must be formally documented via the standard 5x5 risk cubes; and each risk must have a documented mitigation plan with assigned individual(s) responsible for driving the risk to closure.

All status and risk reviews must have an assigned leader and well defined agenda and required participants. The discussions must be supported by objective data (planned vs actual cost and schedules, technical performance, and quality indicators, open versus closed risks over time, etc.) rather than subjective "red, yellow, green stoplight" type indicators.

One strength of Agile based development approaches is the requirement for key stakeholders to communicate on a **daily basis**.


In-Line Testing Approach
DEVELOPMENT ACTIVITIES
EARLY AND OFTEN TEST ACTIVITIES
Time
Early Testing V&V using Models and Simulations
Investing a little more time and Effort up front will result in Significant reduction of total Development cost
Mission Level Req's
Mission Level Architecture
Mission Level Interfaces
Early Mission Level Testing (Models, Simulations, Frameworks, Prototypes,..)
System Level Req's
System Level Architecture
System Level Interfaces
Early System Level Testing (Models, Simulations, Frameworks, Prototypes,..)
Software Level Req's
Software Level Architecture
Software Level Interfaces
Early Software Level Testing (Actual and Representative HW and SW Components; Augmented by Simulations, I/F drivers
Focus on Automated Testing in early SE activities Use Req, Arch, Design tools that facilitate automated testing
Integrated System
Lab based System Testing Actual Hardware and Software , plus simulations
Reduced number of Fundamental architecture/design/interface problems found late in the process. Significant savings
Integrated System
Platform based System Testing Actual Hardware and Software , plus simulations

The project/product owners and leaders from the key engineering disciplines (requirements, design, code, test, safety, security, end-user, etc.) frequently communicate and stay in synch using efficient and well-structured meetings. This philosophy of frequent multi-stakeholder continuous communication can be adopted by waterfall based development teams by establishing regular multi-discipline and multi-stakeholder reviews. Projects must resist the temptation to frequently delay or cancel periodic risk and status reviews due to schedule pressure. Projects must establish the tools and processes to formally capture action items resulting from the frequent project control communication events and ensure that all actions are appropriately assigned and tracked to closure.

avoid "vendor lock" where they must rely on the original developer for all fixes and upgrades

KEY: Early Defect Detection and Removal

It is well known that detecting and fixing defects late in the development cycle is very costly. However, many projects fail to make the investment in the tools, techniques, and methods that may cost a bit more in the short run, but provide for significant reduction in the program's total ownership cost. As illustrated in the diagram above; models, simulations, tools, and test-drivers should be utilized throughout the development cycle to identify and remove defects as early as possible. But note that this requires projects to fund and account for the time and resources to develop and maintain these testing products and processes.

Automation of testing requires a steady investment and applied resources. Automated testing must be implemented at the unit and CSCI level as much as possible and not just at the later and higher level system integration activities. Investment must also be made in implementing a Data-Extraction and Data-Reduction capability. At a minimum, all data that is exchanged between the CSCIs and external systems must be extracted and time-tagged. In addition, key data elements, state changes, and system performance data should be recorded as well. The associated Data-Reduction system and tools must provide for automated detection of data out-of-sequence, out-of-range, and other processing not in accordance with requirements and design intent. The reduction tools must also enable the users to sort and search for specific extraction elements.

KEY: Architecting Multi-Mission and Multi-Platform Capable Software

It is not inherently natural for a profit based organization to provide Modular Open System Architecture (MOSA) approaches resulting in common system and software architectures that can be easily reused or quickly tailored to support multiple warfare missions on various multiple platforms. Government software teams are more apt to provide non-proprietary architectures and designs that are modular, scalable, portable, configurable, and with standardized interfaces that lead to higher system quality while also reducing cost and schedule.

Program Offices should address MOSA from both business and technical aspects in order to avoid "vendor lock" where they must rely on the original developer for all fixes and upgrades. The government and industry software development approach discussed in later sections facilitates:

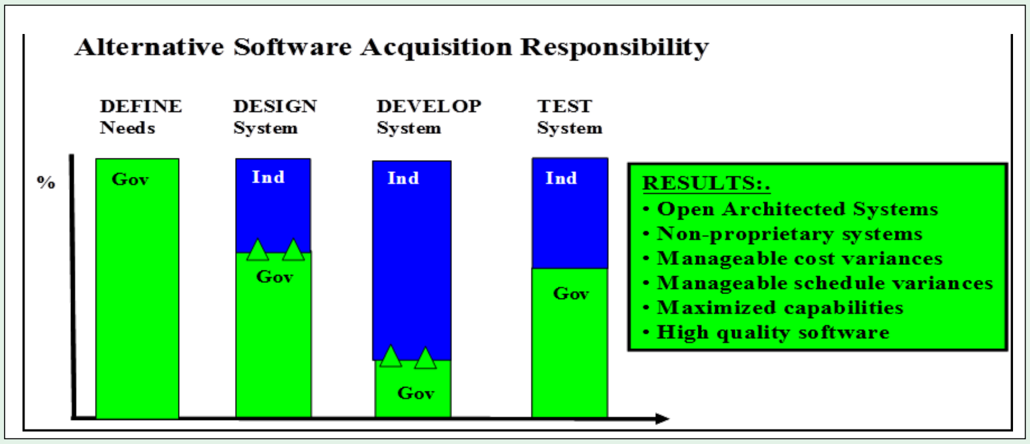
- Increased: Competition, Innovation, Protection of government data rights
- Increased: System and Software Modularity, Scalability, Commonality, Maintainability, Reliability, Usability, Security, and Quality
- Decreased: Proprietary components, Duplication of design and implementation

The root cause for lack of well-architected MOSA systems is due to poor project estimation and the emphasis on rapid prototyping and speed to fleet. Rapid prototyping by definition prevents teams from investing the time required to establish a sound foundational architecture. They are focused on creating a "one off or 80%" solution as quickly as possible. While this approach may be required to address emergent critical threats; it is not a sound approach for significantly reducing DoD software acquisition cost and development timelines in the long run.

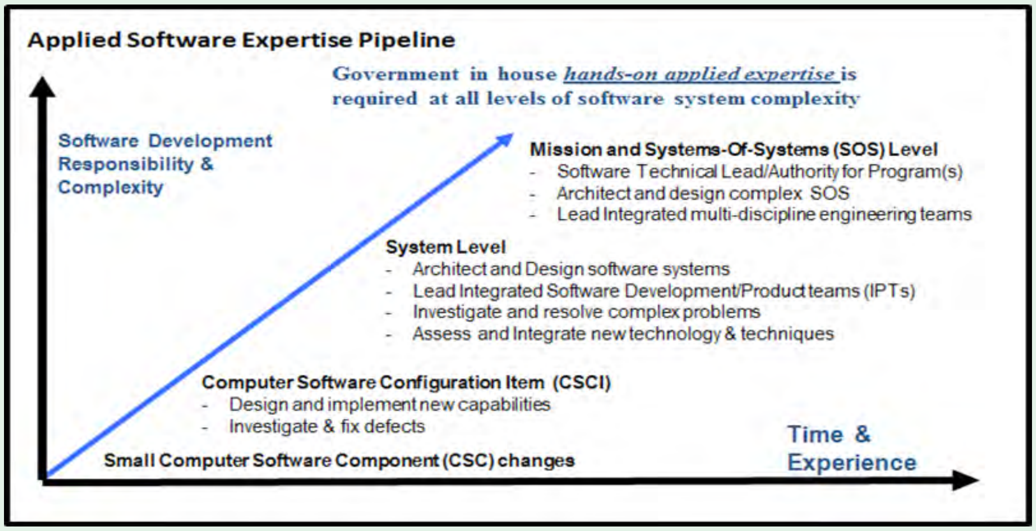
The vast majority of software engineers understand how to architect, design and implement MOSA systems (i.e. abstraction of the software application layer from the hardware, utilization of Virtualization, separation of the user-interface and application layers, utilization of Object Oriented Design (OOD) to abstract and make common the communication, sensor, weapon, engagement, and other interfaces; establishment of open standardized interfaces, etc.). The simple key is to allocate a bit more time and resources up front for sound architecture and design efforts.

KEY: Government In-House Applied Software Expertise

Many studies document that program offices frequently do not have the software experience, skills, training, or expertise required to successfully execute. Although software has evolved into one of the most significant, complex, and critical elements of DoD systems, a common acquisition approach is to treat the software components as "black boxes"; with the detailed understanding of the software (and ownership rights) left almost entirely in the hands of private industry. Government in-house software engineer participation (if any) is typically limited to the reactive (versus proactive) responsibility of reviewing industry developed artifacts and supporting milestone reviews. This over-reliance on private industry can result in costly, non-modular, proprietary system architectures, protracted schedules, and poor performance. Sustainment of these systems is very expensive as the government is "locked into" the original industry software development organization and does not have the leverage (technical knowledge and ownership of the software) required to negotiate better cost and performance. This over reliance on industry has also reduced the ability to maintain an in-house government **applied software** expertise pipeline, leading to a dearth of program



NOTE: Percentages shown above are just notional examples; each program must determine appropriate level of government and contractor mix


Applied Software Expertise Pipeline
Government in house hands-on applied expertise is required at all levels of software system complexity
Software Development Responsibility & Complexity
Mission and Systems-Of-Systems (SOS) Level
- Software Technical Lead/Authority for Program(s)
- Architect and design complex SOS
- Lead Integrated multi-discipline engineering teams
System Level
- Architect and Design software systems
- Lead Integrated Software Development/Product teams (IPTs)
- Investigate and resolve complex problems
- Assess and Integrate new technology & techniques
Computer Software Configuration Item (CSCI)
- Design and implement new capabilities
- Investigate & fix defects
Small Computer Software Component (CSC) changes
Time & Experience

leadership that fully understands software development best-practices. As documented in the 2008 Mr. Donald Winter SECDEF memo: "This combination of personnel reductions and reduced RDT&E has seriously eroded the Department's domain knowledge and produced an over-reliance on contractors to perform core in-house technical functions. This environment has led to outsourcing the 'hands-on' work that is needed in-house, to acquire the Nation's best science and engineering talent and to equip them to meet the challenges of the future Navy. In order to acquire DoN Platforms and weapons systems in a responsible manner, it is imperative the DoN maintain applied technical domain expertise at all levels of the acquisition infrastructure."

A proven successful alternative software system acquisition, development, and sustainment approach utilizes government in-house software engineers teaming with industry software engineers. Government software engineers do not just monitor/review industry software efforts, but rather they are also responsible for the hands-on architecting, designing, coding, integrating, and testing of a **subset** of the mission critical complex software components. Government organic software experts are involved both in the original software component development effort for the system (i.e. pre Initial-Operational-Capability (IOC)) and throughout the software sustainment efforts (i.e. post IOC capability upgrades, enhancements, and defect corrections). The percentage of software work allocated between government and industry software organizations will vary between programs based on multiple factors such as size, complexity, and system maturity. In the example programs that utilize this approach listed below, the percentage of government in-house versus industry software developers varies significantly.

DoD systems, a common acquisition approach is to treat the software components as "black boxes"

This software development teaming approach has been successfully utilized for over 50 years by the Naval Warfare Centers for a wide range of systems (e.g. missiles, guns, directed-energy, lethal and non-lethal detect-track-engage systems) and for a wide range of development approaches (e.g. Waterfall, Incremental, Agile, Rapid Prototyping). Specific programs include the: Strategic Systems Submarine Launched Ballistic Missile (SLBM) Fire Control System (FCS) and Mission Planning System (MPS), Tactical Tomahawk Weapon Control System (TTWCS), the Precision Guided Munition (PGM) and Gun Battle Management System (BMS), Laser Weapon Systems components, and several ground vehicle Detect-Track-Engage systems. These programs have all utilized government and industry software teaming and data-driven best practices to consistently deliver high quality, safe, reliable, modular, scalable, maintainable, reusable, and operationally proven software systems developed within cost and schedule constraints.

By assigning actual software development responsibility to in-house engineers, the Government maintains a software expertise pipe-line as shown in the figure below, and thereby maintains the **applied hands-on software expertise** required to perform as **technical peer level team-mates** with private industry software engineers.

Maintaining the government in-house software expertise pipeline provides DoD Program Leaders with access to in-house software experts required to successfully:

- Assess industry approaches, processes, and effort estimates.
- Offer alternative non-profit-focused technical approaches.
- Mitigate the risk of program office personnel turnover.
- Apply lessons learned and metrics for continuous improvement.
- Be assigned emergent tasks for technology investigation, rapid prototyping, or other technical tasks without costly contract modifications.
- Control industry cost as the software development tasks can be easily transferred to the government team if industry cost growth becomes too great or for poor technical performance (and vice versa). This leverage works best when the government software development organization(s) have been involved from the initial system development efforts and throughout the sustainment phases.

KEY: Applying Lessons Learned

The majority of challenges and best-practices addressed in this paper have been previously reported in DoD software system acquisition and engineering assessment reports (e.g. Defense Science Board (DSB), Government Accounting Organization (GAO), and Software Engineering Institute (SEI)). However, there are software intensive system programs that continue to repeat the mistakes of the past.

Although the majority of programs conduct formal system engineering technical reviews (requirements reviews, design reviews, delivery readiness reviews, etc.), these programs do not collect project execution metrics and conduct periodic formal software process improvement events where the planned-versus-actual cost, schedule, technical performance, quality, assurance, and risk **metrics are analyzed and used** to identify specific **process improvement actions that are then assigned and tracked to closure**.

Program offices lack the leaders and staff with applied software development experience, expertise; training; or awareness of the findings and recommendations from the many software assessment reports required to fully appreciate and adequately resource (funding and schedule) best-practice based software engineering and project control. The significant pressure to reduce cost and schedule drives program managers into "short-term" thinking and decision making which frequently results in the long-run of driving total ownership cost up, significant schedule delays, poor quality, and results in non-maintainable, non-scalable, and non-multi-system-platform architected systems.

Summary

The associated software size, complexity, interdependencies, reliance-on, and software assurance related challenges continue to increase. DoD programs are challenged to consistently develop and deliver high quality software systems that fully meet the warfighter's operational needs and that are safe, secure, reliable, maintainable and scalable. The common challenges that prevent consistent software development and delivery are well documented and known. The primary reason for software project failure is usually not due to the lack of technical expertise by the software development engineers; but rather due to poor project estimation, planning and control.

There are some DoD software projects that have been consistently successful with regards to cost, schedule, technical, quality and operational performance. These projects span a wide range of missions (e.g. strategic, tactical), development methodologies (e.g. nuclear certified, waterfall, agile, rapid prototyping), complexity (numerous interfaces to external systems to stand-alone), and sizes (multi-million lines of code to tens of thousands); and they all utilize many of the same software system acquisition and development techniques to ensure success.

The common keys to success include utilizing a software system acquisition approach that relies on government software engineers to not just monitor/review industry software efforts, but also perform hands-on architecting, designing, coding, integrating, and testing of a subset of the complex software components for mission critical systems. This teaming approach combined with **data-driven project-management and technical execution best practices** has been successfully utilized for decades for several mission critical warfare programs and has consistently resulted in the delivery of high quality, safe, reliable, multi-mission-platform capable and operationally successfully software systems that were developed within cost and schedule constraints. ■

REFERENCES

[1] Assistant Secretary of the Navy Research Development and Acquisition (ASN/RDA), Chief Engineer, Software Process Initiative Software Acquisition Management Focus Team, "*As-Is and To-Be State Reports*", 2007, 2008.

[2] Assistant Secretary of the Navy Research Development and Acquisition (ASN/RDA), ASN/RDA Memo: "*Department of the Navy (DoN) Software Measurement Policy for Software Intensive Systems*", July 2009.

[3] Assistant Secretary of the Navy Research Development and Acquisition (ASN/RDA), ASN/RDA Memo: "*Strategy to Balance Acquisition In-House and Contractor Support Capabilities*", December 2008.

[4] Assistant Secretary of the Navy Research Development and Acquisition (ASN/RDA), ASN/RDA Memo: "*Meeting of the Navy Laboratory/Center Competency Group*", November 2008.

[5] Government Accounting Office (GAO), Report to Congressional Committees Best Practices, February 2008.

[6] Office of the Under Secretary of Defense for Acquisition, Technology and Logistics, Report of the Defense Science Board (DSB) Task Force on Developmental Test and Evaluation, May 2008.

[7] Office of the Under Secretary of Defense for Acquisition, Technology and Logistics, Report of the Defense Science Board (DSB) Task Force on Defense Software, November 2000.

[8] Secretary of Defense (SECDEF), SECDEF Memo: "*Department of the Navy Acquisition*", December 2008.

[9] Senator Carl Levin, U.S. Senate Committee of Armed Services Press Release, March 2009.

ABOUT THE AUTHOR

Joe Heil has worked as a Software Engineer for the Naval Surface Warfare Center Dahlgren Division (NSWCDD) for over 30 years. The majority of Joe's career was spent as the Lead of Government and Industry Software Development Integrated Product Team (IPT) for the Tactical Tomahawk Cruise Missile Weapon Control System (TTWCS). As the TTWCS Software Development IPT lead, Joe was responsible for defining the software development processes, leading, coordinating the software development efforts and ensuring successful cost, schedule, technical, safety, and quality performance

Joe founded and is the current lead of the Naval Software Community of Practice (SW COP). The NAVAL SW COP has over 300 registered government in-house software experts from across 19 different organizations, including the various Naval Warfare Centers, Naval System Commands, Army and Air Force engineers, and leaders from the Naval Post Graduate School (NPS), Defense Acquisition University (DAU), and Carnegie Mellon University Software Engineering Institute (SEI).

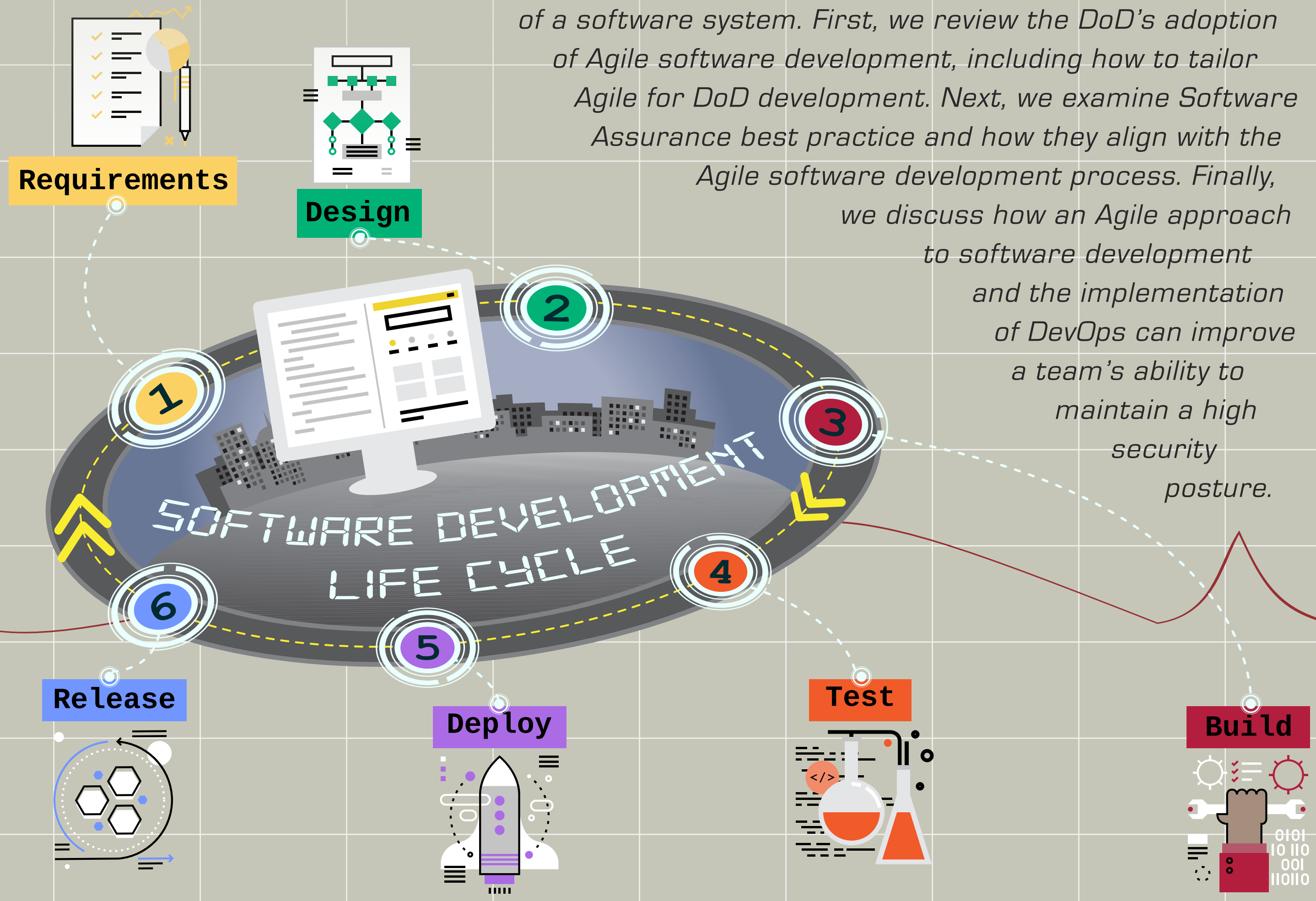
Joe's current leadership roles and responsibilities are focused on increasing awareness and application of data-driven software engineering best-practices across the naval enterprise. These leadership roles include:

- Deputy Chief Engineer for the Naval Surface Warfare Center Dahlgren Division
- Navy Advisor to the Defense Science Board (DSB) Task Force (TF) for Software
- Lead of the Naval Software Community of Practice (SW COP)
- Lead of the Naval System Engineering Stakeholders Group (SESG) SW Group
- Contributor to the DOD SW Assurance Community of Practice (SWA COP)

SOFTWARE ASSURANCE IN THE AGILE SOFTWARE DEVELOPMENT LIFECYCLE

By: Bradley Lanford, Engility Corporation

Over the last 30 years, the DoD has struggled to adapt to the ever-changing world of software development. Of these many struggles, implementing Agile software development and practicing systems security engineering are two struggles that continue to plague the DoD. In an attempt to overcome both of these hurdles, this paper presents a Software Assurance approach that is tightly woven into the Agile software development lifecycle and emphasizes the benefits that Agile development best practices can have on the security posture of a software system. First, we review the DoD's adoption of Agile software development, including how to tailor Agile for DoD development. Next, we examine Software Assurance best practice and how they align with the Agile software development process. Finally, we discuss how an Agile approach to software development and the implementation of DevOps can improve a team's ability to maintain a high security posture.



Agile Development in the Department of Defense

Building and delivering software in incrementally has always been a part of software development. The commercial world has been modifying and enhancing that process since the publication of the Agile Manifesto in 2001 [1]. The Manifesto identifies 4 values:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

These are then explained based on 12 principles that outline a high level, highly collaborative, time boxed process that focuses on delivering working software to users and provides a method for adjusting to changes in requirements. Since its publication software has become more complex and is now the most costly effort in almost all DoD programs [2]. In response, the DoD has adopted many of the Agile development practices made popular by the commercial industry. The many struggles of that adoption were documented in the 2012 GAO report [3] and five years later the DoD continues to struggle.

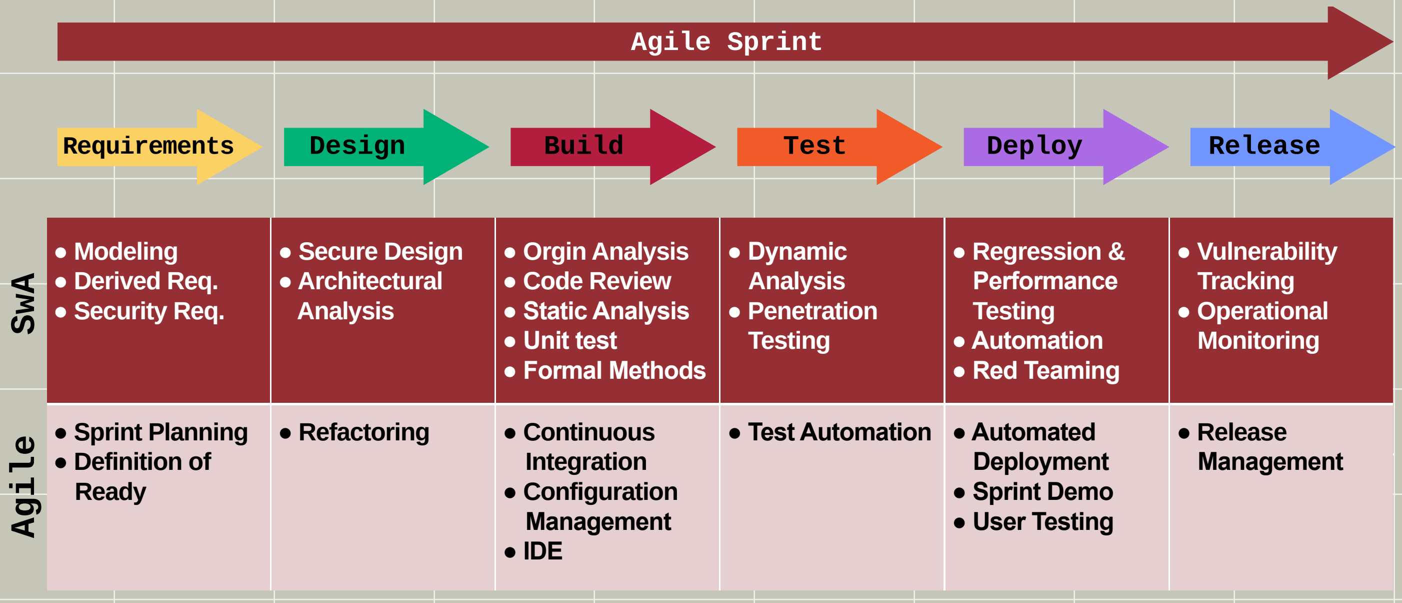
Cyber security is a high priority for all programs in the DoD

A barrier to adopting a true Agile methodology is often the Acquisition process and the strict requirements that are placed on government program offices. Even as industry has evolved to only offer Agile solutions, those solutions must be tailored to fit within Acquisition. The constraints placed on any Agile implementation are confined to the time between the finalization of the Capability Development Document (CDD), which defines all requirements for the entire period of performance, and operational test, which is designed to determine the program's ability to meet CDD requirements. These two road blocks which are essential to the acquisition process are fundamentally in opposition to Agile's flexible requirements and user interaction throughout development. As development methodologies continue to move further from rigid requirements, programs remain confined by requirements that must be defined prior to contract award and eventually tested to with limited operational test interaction in development.

Despite these constraints, the defense industry has developed its own variety of Agile that derives many of the benefits of the Agile process while still meeting the requirements of acquisition. What is lost in the adherence to Acquisition is the flexibility in user requirements that evolve throughout the development lifecycle. What is retained is the built in quality that comes from the cadence of Agile development. Through this cadence, DoD programs can apply and maintain software assurance best practice throughout the life of the software.

Software Assurance in the Agile Software Development Lifecycle

Software Assurance is fundamental to the systems engineering process and ensures high quality software is delivered with limited vulnerabilities. In order to achieve this goal software assurance must be applied across the full Software Development Lifecycle (SDLC). Many organizations, such as the National Institute of Standards



	Requirements	Design	Build	Test	Deploy	Release
SwA	• Modeling • Derived Req. • Security Req.	• Secure Design • Architectural Analysis	• Orgin Analysis • Code Review • Static Analysis • Unit test • Formal Methods	• Dynamic Analysis • Penetration Testing	• Regression & Performance Testing • Automation • Red Teaming	• Vulnerability Tracking • Operational Monitoring
Agile	• Sprint Planning • Definition of Ready	• Refactoring	• Continuous Integration • Configuration Management • IDE	• Test Automation	• Automated Deployment • Sprint Demo • User Testing	• Release Management

Figure 1: SDLC with Software Assurance and Agile Development Process Overlays

and Technology (NIST), have detailed this process, but do so in a traditional waterfall approach [4]. In order to transition this software assurance approach to an Agile software development lifecycle it is important to utilize not only the cadence for development and testing but also the cross functional team structure to reinforce your assurance practices. Figure 1 provides an overlay of software assurance best practices onto a single Agile development sprint. Although similar to a standard development lifecycle each phase has a unique Agile implementation that provides a structure for assurance practice.

Requirements

The foundation for software assurance is defined with the requirements. Requirements should be written and decomposed focusing not only on what the system needs to do functionally, but how it will be protected. To inform these decisions, programs model threats, complete criticality analysis, and define functional and non-functional software security requirements. Due to the level of requirements that must be defined prior to Engineering & Manufacturing Development (EMD) phase of the DoD acquisition lifecycle, programs should have a more complete definition of software assurance requirements than a typical Agile development effort.

While requirements may be more fully defined it does not mean they are fully understood or even evolved to meet the changing threats required to complete the mission. Prior to the start of an Agile sprint, the team reviews the requirements for any new capabilities being developed. From an assurance perspective all relevant security requirements should be documented and included in these user stories for the upcoming sprint. In addition to new requirements, all acceptance criteria for sprint work should be included in the user stories, referred to as the definition of ready, to ensure that stories are actionable for developers. This includes code reviews, completion of unit tests, and use of static analysis tools prior to delivery of new code.

Detection framework will also improve the ability of the human analysts to acquire and maintain situational awareness in cyberspace

Design

Along with defining requirements, the team should make design decisions prior to the first sprint and then review these designs with each sprint planning meeting. This includes following secure architectural design patterns and doing an architectural analysis of risk. Once architectural drawings and system modeling is complete, the team can make changes and reassess risk with each subsequent sprint planning session. This also allows programs to identify any new vulnerabilities affecting the initial design and plan rework efforts based on the prioritized backlog. The built-in quality expected of Agile development relies on the ability to refactor existing code to address changes in requirements. As threats change and design pattern vulnerabilities are discovered, the flexibility to refactor becomes far more important in the development of a secure system.

Build

An Agile development methodology is only as good as the tools and environments used to facilitate continuous integration. It is also these tools and environments that enable software assurance practices to be incorporated into the software development. All members of a development team can have access to an integrated development environment (IDE) to ensure secure coding standards are being followed. Additionally check-in procedures for new code can require static analysis of new code, code review by peer programmers, and origin analysis to determine the source and existing vulnerabilities of all code added to the stream. Integrated team testers should identify vulnerabilities and ensure they are resolved prior to check-in. Daily stand-ups include representatives from cross-functional teams including database administrators, architects, and Information Assurance to address system assurance and other related questions to ensure development teams are aware of potential sources of vulnerabilities.

The effectiveness and efficiency of Agile teams relies on the automation of day to day procedures. Automation is also key to software assurance because it enables a system to be thoroughly and accurately tested for vulnerabilities on a continuous basis without overburdening a test team. This automation begins with the development team and then is provided for reuse later in the lifecycle. Once automated, unit and regression testing can take place as needed to ensure working software that is free of vulnerabilities. The practice of assuring software, once thought to be burdensome to software developers, can be aligned with the Agile cadence and integrated into development, compilation, and delivery tools to become a standard part of the development process. In return, vulnerabilities are found earlier and fixed prior to delivery to the test environment. As a side effect, developers learn secure coding practices through experience and reduce similar issues from occurring in the future.

Test

Through Agile development, parts of the test process are moved into the software development phase to fix defects prior to integration into the code base. This process formalizes test cases and often automates them for reuse. Independent Verification and Validation (IV&V) teams use existing test and develop additional testing to discover defects prior to user acceptance testing. Through Agile's continuous integration model, testing can occur continuously with testers having access to the code base in an environment designed to mirror the operational environment. Static and dynamic analysis tools can scan and examine the entire code base. These results, along with penetration testing, provide direct feedback to developers and increases defect/vulnerability reporting into the product backlog.

At the end of each sprint all working software is delivered to one or multiple test environments. Test teams work a single sprint behind development to identify defects and vulnerabilities that can be prioritized in the program backlog for the next release. In addition to test teams, Agile relies on the involvement of users in the sprint process. Along with the ability to provide sprint demo's the continuous development environment and automated deployment allows users the opportunity to test functioning code before release to production. This adds an additional layer of assurance as users can determine if software functions as intended and only as intended without simply relying on requirements. Through the integrated development environment, users can also provide feedback in the form of defects to the product backlog and development teams.

automation limits the ability to compromise the system through the addition of vulnerabilities at the release stage

Deploy

Due to the acquisition process, code cannot be released every sprint, but Agile dictates frequent release of working software. In order to facility this, delivery teams should maintain a pre-production environment that mirrors production and accommodates frequent releases. This allows early operational monitoring, red teaming activities, and identifies vulnerabilities prior to release. As mentioned above automation is integral to the success Agile development. Along with the automation of test cases, release management should also automate the deployment process to ensure a thorough and repeatable process. This automation is also important to the assurance of the system as automation limits the ability to compromise the system through the addition of vulnerabilities at the release stage. Projects can automate the configuration, code signing, unit testing, versioning, code analysis, and test deployment to ensure proper release to all environments. Release acceptance testing, taking place on pre-production, can also be automated based on test cases developed throughout the sprint. Acceptance test should include regression, performance, and integration testing to identify vulnerabilities. Deployment follows the software development cadence with working software being delivered at the completion of each sprint and releases to the production aligning with completion of functional capabilities. Once again, projects can use the IDE for configuration management of all defects and vulnerabilities to include software version and the environment where they were identified, with mitigation and fixes tracked and included in regression test cases.

Release

Release of an acquisition system into the production environment requires that programs complete operational testing and obtain an authority to operate. This is a very detailed and, for Major Defense Acquisition Programs (MDAP), waterfall process. Many times it is this process that discovers a majority of vulnerabilities, when the costs of rework are expensive. The Agile approach outlined above discovers defects and vulnerabilities within development sprints, when the cost to mitigate or fix is comparably low. Once operational, projects should continue to monitor and maintain these systems using tools designed to run along with the application, operational monitoring, to identify any changes in system performance or runtime.

Benefits of Agile and Introduction to DevOps

Maintaining a high security posture is becoming increasingly difficult as the cyber security threats become more complex. Although the fundamental systems engineering process for developing secure software remains the same, new methodologies, tools, and technologies are always emerging to protect our systems. The Agile manifesto was written to place an emphasis on the importance of responding to change and through the implementation of Agile teams can not only streamline software assurance best practice, but can also adapt to changes when new vulnerabilities or assurance techniques are discovered. These are some of the key Agile processes that can be used to facilitate software assurance best practice:

Agile Cadence

Cyber security is a high priority for all programs in the DoD. Unfortunately it is not always funded and often times it is viewed as resource intensive for programs trying to implement it outside of the systems engineering process. Using an Agile methodology, the cadence of sprint development makes it possible to neatly align all elements of software assurance. Sprint planning requires review of design, architecture, and requirements. Development teams perform code reviews, develop unit test, and run all code through static analysis tools before delivery. Team testers ensure test plans are developed and acceptance criteria are met. All of this takes place in a two to four week sprint and ensures that software assurance activities are not burdensome due to the reduced scope. At the completion of a sprint, IV&V and release management teams operate on the same cadence once code is integrated. All of this culminates in a scheduled release of working and secure software that has been rigorously tested before moving to pre-production for user acceptance or the production environment.

Continuous Integration

No matter the framework chosen for the implementation of Agile the engine that makes a team successful is the continuous integration environment. This is the infrastructure which allows all team members to work on and deliver code to a single development stream. Agile emphasizes early delivery of working code. For systems engineers the continuous integration environment serves as a means to ensure all code is properly scanned, reviewed, and tested prior to delivery. This includes origin analysis of libraries and functions, unit test, code reviews, and static code analysis. Smaller incremental delivery allows code to be scanned quickly and with little impact on performance. Once delivered, the single repository is ideal for regression testing, static analysis of the full codebase, and dynamic analysis of deployed software. In addition to the process, IDEs used for continuous integration can provide instant feedback on adherence to coding standards and best practice as well as configuration management of vulnerabilities that is accessible by all team members.

Continuous Delivery and DevOps

Continuous delivery is the process in which code that has been delivered to the development stream is automatically built, tested, and prepared for release. Although it is not vital to Agile development it has been adopted in most instantiations. Automation is an important practice in securing a system, as it ensures a repeatable and consistent process that does not introduce vulnerabilities into the system. Through continuous delivery engineers, can automate software assurance tool usage into the build process and provide feedback to developers based on test results. Another important element of continuous delivery is the infrastructure required to support release to multiple environments. Once the build and release process is automated, new code can be released to test pre-production, or production environments allowing regression testing, code analysis, red teaming, and penetration testing to start immediately. Additionally having the flexibility in infrastructure allows for operational monitoring prior to release to production.

A concept that has grown from the movement to Agile is DevOps. DevOps is in many ways similar to Agile but with a focus on delivering and evolving products at a high velocity. Continuous deployment is key in realizing this objective and many times programs merge development and operations teams to streamline deployment. In this case many of the roles of the operations team are realized using code such as infrastructure, policy, and monitoring. Infrastructure as code allows development teams to provision and manage infrastructure instead of manual configuration. This can provide environments for more thorough security testing or a location to deploy known malware to test applications. Infrastructure as code also adds security because it can be tracked, validated, and reconfigured automatically, flagging non-compliant resources. Agile and DevOps methodologies both focus on optimizing the process to allow faster delivery to the user, the end result is a well-defined process that can be used to build in assurance practices to maintain a high security posture [5]. ■

REFERENCES

[1] Agile Manifesto. 2001. “Manifesto for Agile Software Development” retrieved from http://agilemanifesto.org/ on March 31, 2017

[2] Hagen, Christian; Sorenson, Jeff 2013. “Delivering Military Software Affordably,” *Defense AT&L*, Mar-Apr 2013 http://dau.dodlive.mil/2013/04/24/delivering-military-software-affordably/

[3] GAO 2012 “Effective Practices and Federal Challenges in Applying Agile Methods”

[4] GAO-12-681: Published: Jul 27, 2012. Publicly Released: Jul 27, 2012.

[5] Jarzombek, Joe. 2012. “Software Assurance: Enabling Security and Resilience throughout the Software Lifecycle” http://csrc.nist.gov/groups/SMA/forum/documents/october-2012_fcsm-jjarzombek.pdf

[6] Amazon Web Services. 2016. “What is DevOps?” retrieved from https://aws.amazon.com/devops/what-is-devops/ on March 31, 2017

HERE TO SUPPORT YOUR MISSION.

Is your organization currently facing a challenging Information Technology oriented research and development problem that you need to have addressed in a timely, efficient and cost effective manner?

HOW CAN CSIAC HELP?

In a time of shrinking budgets and increasing responsibility, CSIAC is a valuable resource for accessing evaluated Scientific and Technical Information (STI) culled from efforts to solve new and historic challenges. Our CSIAC SME network includes experienced engineers and technical scientists, retired military leaders, information specialists, leading academic researchers, and industry experts who are readily available to help prepare timely and authoritative answers to complex technical inquiries.

Once submitted, the inquiry is sent directly to an analyst who then identifies the staff member, CSIAC team member, or SME that is best suited to answer the question. The completed response is then compiled and sent to the user. Responses can take up to 10 working days, though they are typically delivered sooner.

WANT TO SUBMIT A TECHNICAL INQUIRY?

The CSIAC provides up to **4 hours of Free Technical Inquiry research** to answer users' most pressing technical questions. Our subject matter experts can help find answers to even your most difficult questions.

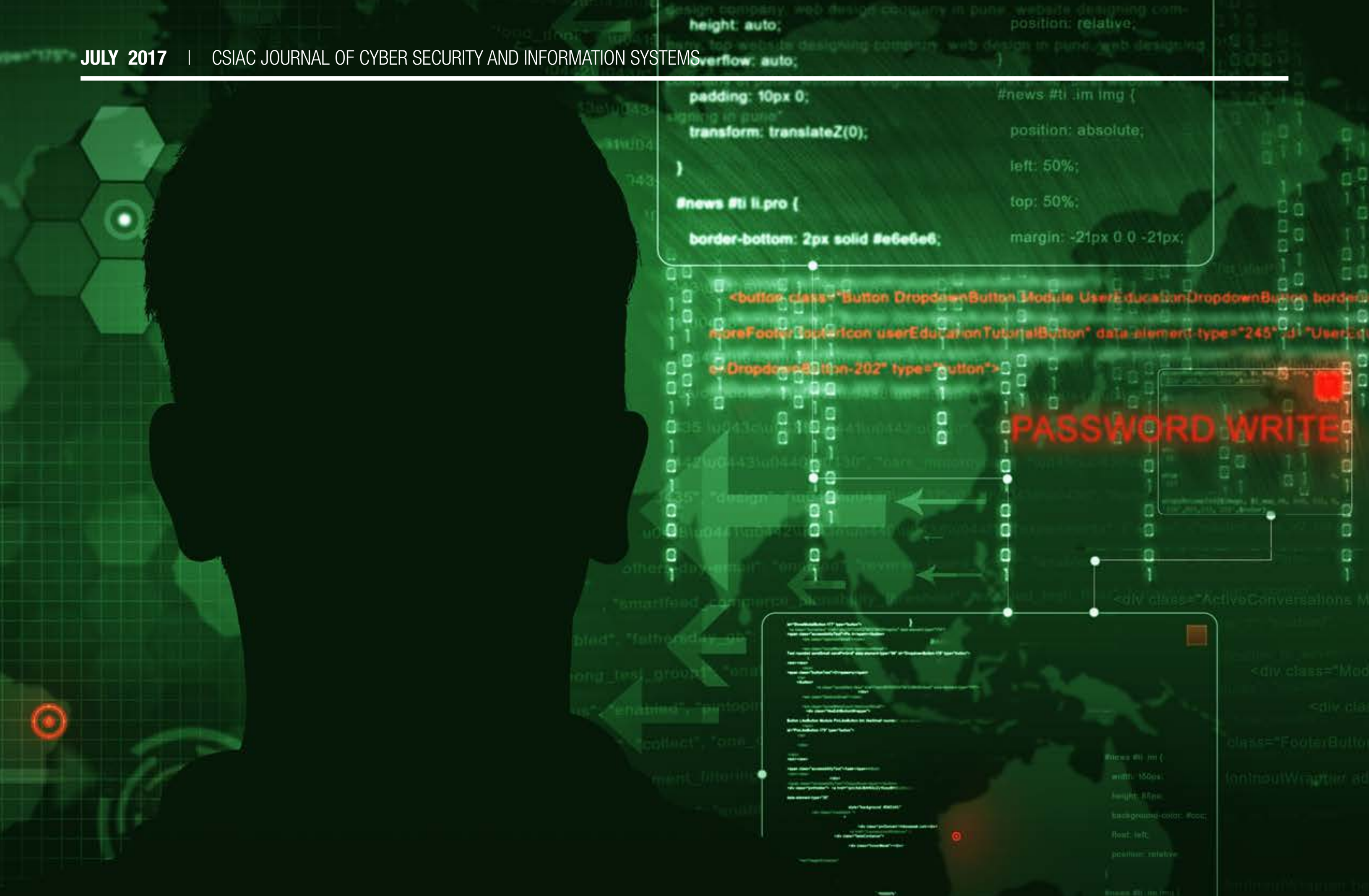
Technical inquiries can be submitted to CSIAC via our csiac.org, or by email, phone or fax.

CALL NOW! 800-214-7921

EMAIL AT: INFO@CSIAC.ORG

CSIAC *Cyber Security & Information Systems Information Analysis Center*

FOR MORE INFO, GO TO:
https://www.csiac.org/free-inquiries/



HACKER 101 & SECURE CODING: A Grassroots Movement towards Software Assurance

By: Carol Lee, Jasen Moran, Joel McCormick, Kolby Hoover, Matt Hackman, Paul McFall, Roger Lamb, Scott Nickeson

The frequency and complexity of attacks upon the software assets of the United States Military is increasing at a rate which requires a massive organized response from the defense community. This threat is unlike anything encountered before and the response must be swift and focused. Currently the Navy and the Department of Defense are working multiple fronts in order to keep pace with the actual threats. The predominance of the attacks are focused in one area which should help focus a part of our defense. The Gartner report[1] stated that 84% of all attacks are at the application layer. Therefore, securing the application layer should be the top priority. To achieve security in this area, computer scientists need to build software with security in mind from the beginning. However, most software developers have not been trained in secure coding techniques within their undergraduate programs. The solution lies with driving the culture of software development toward software assurance knowledge and practices; which is not a trivial undertaking. The goal of this article is to describe a grass roots training class that was created at the Naval Surface Warfare Center Dahlgren Division (NSWCDD) to provide software developers with an introduction to the fundamentals of software assurance and secure coding.

Introduction

The Cyber War has not only begun, but it is well underway. Sun Tzu in The *Art of War*[2] offers not only insight but also a potential method for assessing whether one is prepared for battle.

If you know the enemy and know yourself…
You need not fear the result of a hundred battles.

If you know yourself but not the enemy…
For every victory gained you will also suffer a defeat.

If you know neither the enemy nor yourself…
You will succumb in every battle.

There have been a significant number of successful cyber attacks on the U.S. Government over the past several years, from the 2014 Office of Personnel Management Data Breach to the successful cyber attack on the IRS in 2016 and those are just the openly known attacks. Using Sun Tzu's philosophy as an assessment, one is forced to admit that at best we don't know our enemy (where and how they are most likely to attack) and at worst we don't know ourselves either (where most of our vulnerabilities are located). The primary response to this scenario has been to create a wave of new defense methods and tools. The goal of this article is to review and outline the successes and lessons learned from a "grass roots" training class that was created at the Naval Surface Warfare Center Dahlgren Division (NSWCDD) to provide software developers an introduction to the fundamentals of software assurance to include secure coding.

Why Train Developers in Software Assurance?

The beginning was simple, a team of software engineers moved from satellite and mobile development to the mysterious realm of cyber R&D. In the software development community, there is a belief that network defenses, such as firewalls and intrusion detection systems, safeguard our software systems and therefore developers do not have to concern themselves with security at large. One of the early realizations the team had was that software applications are an attacker's main target and network defenses can be defeated. Hackers try to use developers' tools, such as input fields, and computer resources, such as memory, in ways that weren't intended by the original designers. This is one of the primary ways hackers can obtain system access and information. For example, developers write code with the expectation of what constitutes normal inputs that the user will give to an application. Developers often test for accidental input errors, but they don't design or code with the idea that someone is intentionally trying to take advantage of their application through a buffer overflow weakness.

Gary McGraw, IEEE Senior Member and Secure Coding expert, notes that 50% of vulnerabilities that attackers take advantage of occur in software design.[3] The 2014 Gartner Research report stated that 84% of breaches exploit vulnerabilities in the applications themselves.[1] These facts are not well known or understood among the majority of developers who are still not trained in secure software development in their undergraduate or graduate programs. However, as we came to realize, if the software itself can be the target and the weakest link in a system, then secure software can be the best defender. Even security defense tools are themselves software that can have vulnerabilities, and they must also be coded securely.

Therefore, secure software development became the focus and software developers became the fundamental solution. Why? Software developers take pride in their code and inherently strive to make their software solid and robust through areas such as reliability, scalability and maintainability. If software security was added to this list, through exposure and adoption of secure coding knowledge, then software would become intrinsically more secure. Code security would be naturally and automatically included in the design, architecture and daily development. Software assurance includes secure software development practices, processes and tools. It is part of the overarching software engineering umbrella. Upcoming new accreditations and processes are attempting to address cyber issues. However, success will be achieved most efficiently if software designers and developers understand and adopt software assurance principles in order to thwart hackers and fulfill their missions.

Getting Developers Interested in Software Assurance Training

As the team progressed in studying cyber security and software assurance, the more it became clear that this was not only an ambitious undertaking but also an urgent need. While there is a vast amount of information out there, developers do not know where to look or even that they should be looking. The material came mostly in two varieties. Either at a general level with instructions such as "implement secure programming practices" with no details to help a developer get started or at too detailed a level for developers with no previous training or subject awareness to easily understand. The team spent a great deal of time collecting and digesting the volumes of information and training each other on information they found. The obvious next step was to create a "grass roots movement" in software assurance and secure coding. By presenting this information in an easy to understand manner, developers could immediately proceed to look for insecurities in their own code and fix them.

However, often when new rules or processes are added to our work, the initial and natural response is to resist and attempt to subvert the extra work, especially if it is not seen as adding value. Therefore, the software assurance training needed to create excitement and immediate interest. Thoughts of past government training in Information Assurance (IA) came to mind ("*Bueller, Bueller…*"). While IA should be performed during the development lifecycle, unfortunately, it became a checklist at the end of development and therefore not as effective. Software assurance activities need to be intimately integrated with software development in order to be part of the Navy's solution against its cyber enemies. The plan was to create a set of classes, held over a couple of days with fun hands-on activities to relate the volume of information in an interesting and easy to understand manner while keeping the students awake and engaged. The week would be divided into three sections: Hacker 101, Secure Coding, and Software Assurance. The Hacker 101 "*be a hacker*" portion gained early interest and filled the seats to maximum capacity. Who doesn't want to pretend to be a hacker, even if they aren't really sure what that means? This portion would help the developers understand the hacker's mindset, satisfying Sun Tzu's concept of understanding your enemy. How can you truly create secure code if you don't know how the hackers are attacking? After this class grabbed everyone's attention, as well as concern about how to protect their software, the Secure Coding class would follow to begin teaching secure coding techniques, reinforced with more hands-on activities. Additionally, Software Assurance highlights would be presented before both Hacker 101 and Secure Coding to help introduce the topic as well as illustrate the connection between software assurance, cybersecurity and secure coding. Finally, the Software Assurance class would be presented in more depth to round out the week with new processes and testing tools the developers could adopt to help them code more securely.

developers understand the hacker's mindset, satisfying Sun Tzu's concept of understanding your enemy

Setting the Stage

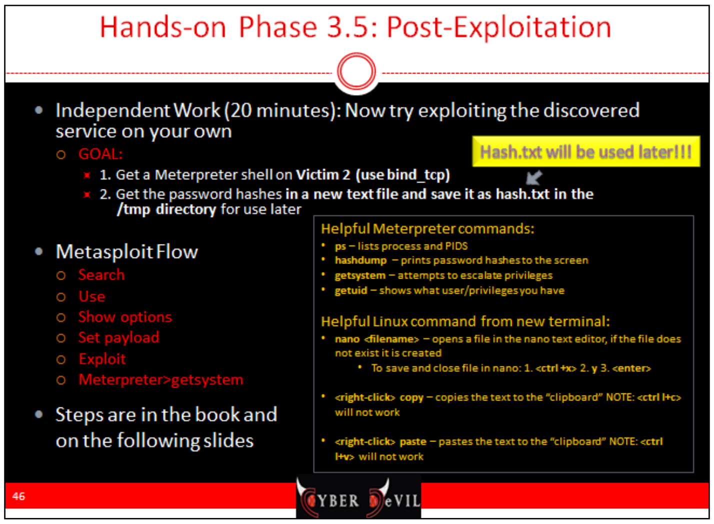
The final version of the training ended up encompassing over 850 slides with many hands-on activities. The Cyber Defense Vulnerability Insight Laboratory (Cyber DeVIL) was set up to accommodate groups of students with each student having a computer with virtual machines, network connections to the attack server, the hands-on activities and follow-along steps so no one would fall behind. Two pilots of the classes were advertised across two different geographical locations to maximize the variety of developer knowledge bases. The goal of the pilot classes was not only to train developers but also to find out what developers already knew about secure development, what would need to be added or removed from the training to make it more viable across the Navy, and most importantly, to find out if it was interesting. If the training wasn't interesting, the software assurance game would be lost before the movement even started. The pilot classes received more candidate requests than there were seats to accommodate people, so the selection was based upon two criteria. The first criterion was experience with software development. Software developers were chosen with a range of experience from recently out-of-school to seasoned professionals. This would provide a sense for what skills were being taught in universities as well as what had been learned during a significant career length. The second criterion was prior knowledge of software assurance. Developers were chosen with a range of software assurance knowledge from none to some. This would help know how well the training compared to other information as well as providing feedback about the style of training. Additionally, it would validate the assumption that most developers were unaware of software assurance as well as observe how well they responded to the topic and the extra work that this effort was going to demand. All three topics were created with open-source information and provided those resources to the attendees for further use. The final day included a guest speaker who shared his substantial experience as a software assurance tester and what he had seen work to greatly improve security in government software.

➜ KNOW YOUR ENEMY - HACKER 101

The purpose of the Hacker 101 class was twofold. First, the class was meant to illustrate the mindset of the hacker and what they could do with weaknesses in code. It was important for students to understand that functionally correct code can provide an attack pathway into the overarching system if it has even minor security oversights. Second, the class allowed the developers to play at being a hacker. This concept piqued their interest, got them to sign up for the class, and provided a fun approach to a new critical topic. It also gave them the impetus to take ownership to find and fix the weaknesses that could be in their code.

The introduction to the class quoted experts stating that security weaknesses in code are rampant and that software security is not understood as an essential priority alongside functionality. The topic of Software Assurance as a component of Software Engineering was introduced. Also discussed were the National Defense Authorization Acts, which Congress had mandated to direct the Department of Defense to perform software assurance to better secure our military systems. The unclassified open-source Mandiant report[4] was noted as an example of real-world attack activities.

The class covered the different phases of an attack: Reconnaissance, Network Scanning, Exploitation, Post-exploitation, Maintaining Access and Covering Tracks. Several demonstrations were given to show how an entire attack would look across the phases for a more in-depth look into an activity which would require a higher skill level and extended time to fully complete. The students used Kali Linux, Metasploit and the command line for tools such as Nmap across the phases to get hands-on experience. An overview of each hands-on activity was presented at the end of each topic discussion (ex. Figure 1) followed by detailed steps.



Hands-on Phase 3.5: Post-Exploitation

- Independent Work (20 minutes): Now try exploiting the discovered service on your own
  - GOAL:
    - 1. Get a Meterpreter shell on **Victim 2 (use bind_tcp)**
    - 2. Get the password hashes **in a new text file and save it as hash.txt in the /tmp directory** for use later
- Metasploit Flow
  - Search
  - Use
  - Show options
  - Set payload
  - Exploit
  - Meterpreter>getsystem
- Steps are in the book and on the following slides

Hash.txt will be used later!!!

Helpful Meterpreter commands:
- **ps** – lists process and PIDS
- **hashdump** – prints password hashes to the screen
- **getsystem** – attempts to escalate privileges
- **getuid** – shows what user/privileges you have

Helpful Linux command from new terminal:
- **nano <filename>** – opens a file in the nano text editor, if the file does not exist it is created
  - To save and close file in nano: 1. **<ctrl +x>** 2. **y** 3. **<enter>**
- **<right-click> copy** – copies the text to the "clipboard" NOTE: **<ctrl l+c>** will not work
- **<right-click> paste** – pastes the text to the "clipboard" NOTE: **<ctrl l+v>** will not work

46

CYBER DeVIL

Figure 1: Example of Student Hands-On Activity

Reconnaissance

The Reconnaissance section discussed how hackers gain information against their targets by using a variety of websites for open source information gathering, tools and social engineering tactics. Armed with this information, hackers can send phishing emails or use help desk personnel to get accounts and passwords. In this manner, attackers build their information base of a target's potential weakness areas.

Network Scanning

The Network Scanning section let the students try their hand at identifying the operating system information, open ports and available services of their target. The hacker would then recreate a company's infrastructure based upon the information they found. This would be completed on their own equipment so they could then laboriously search for vulnerabilities and experiment with creating exploits against them until they had success with an effective attack against this test infrastructure. Only once flawlessly successful, would they try the attack against the actual target.

Exploitation

The Exploitation section emphasized how software vulnerabilities such as stack/heap overflows, Structured Query Language (SQL) injection, cross-site scripting and other code weaknesses, are a gold mine for hackers and provide critical pathways to achieve success. The activities centered on using Metasploit, which is a hacking framework designed to streamline the attack process. Metasploit has a library of prebuilt exploits against applications and services and corresponding payloads for those exploits to support attacks. The hacker community creates and shares successful exploits for others to use. Using the information they gained in the previous phases, even novice hackers can use Metasploit to look for a service like a SQL Server or a product such as Adobe Flash and use a prebuilt attack against it.

Skilled attackers can relatively easily thwart our current security perimeter defenses

In the hands-on portion of the Exploitation section, the students used Metasploit to attack the provided lab entities with the same exploit used in the Stuxnet attack. Additionally, a full exploitation demonstration was performed for the class using WarFTP to show how to use tools such as fuzzers and debuggers to crash an application, find weak code, and then create or inject malicious code for a successful attack on a system. Again, this stressed the necessity for developers to understand how code could be exploited and how to prevent the associated weaknesses so they could remove the paths hackers use.

Post-Exploitation & Maintaining Access

Hackers can remain active after an attack. In the Post-exploitation phase, the students reviewed network traffic, obtained passwords, moved to other systems and hid their traffic. Once an exploit is successful, hackers want to maintain access. They want to ensure that they can easily get back into a system and not be dependent on using their initial entrance to the system in case the initial path was detected or patched. For example, perhaps a watchful administrator responded to an intrusion detection system noting an unusual log-in. By the time the administrator closes the door the attacker came through, the attacker may have already created other accounts, installed a backdoor Trojan to easily get back in, set up callbacks to be able to send data out from the system back to themselves, or started stealing and cracking passwords to get into other areas of the system.

Covering Tracks

Finally, an attacker wants to cover their tracks so their existence within the system is not discovered. They do this by changing log files and other artifacts to remove traces of their activities. The students were given an opportunity to use the sum of the knowledge they had gained and tackle a mini Capture-the-Flag challenge, as well as other activities such as bypassing a firewall. A handout of references and topics was given to the students at the end of the class.

Hacker 101 – Summary & Take Away Message

Skilled attackers can relatively easily thwart our current security perimeter defenses that have been set up to keep them out. Unfortunately, an administrator and protector of the perimeter defenses has an overwhelming set of tasks to accomplish; an enemy with ever-changing tactics; and tools that can only address a part of the problem, are only successful against already known attacks and which may have been created with insecure code themselves. Once an attacker has obtained access to any part of an entire system of systems, they can then begin to install their own software. This software could potentially let them back into the system, control parts of the system, and/or move to more critical components in a system. While it is important to apply additional rigor to the more critical components, if a supporting non-critical component has access or connections into the system, it could be the weakest link and the key to the back door of the entire defense system. This is why securing code is so important. It is the main target and the last defense.

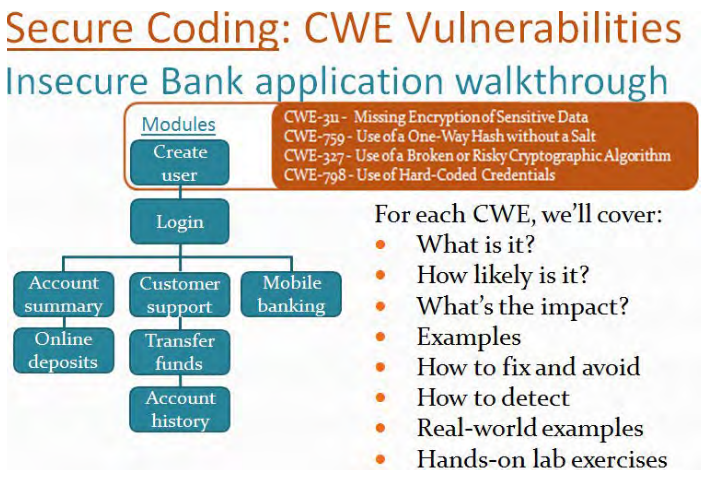
➜ KNOW YOURSELF – SECURE CODING

The purpose of the Secure Coding class was to introduce the topic of secure software development and illustrate what developers could do to find and fix weaknesses in their current code and prevent weaknesses in the future. While this topic has been around for more than a decade and there are volumes of information available, developers have not been trained in this topic in their undergraduate programs; know how to use it or even that it exists.

The introduction to the class highlighted the significant amount of open-source and Department of Defense (DoD) information on secure coding. It delved deeper into the resources by presenting the Common Weakness Enumeration (CWE) and its Top 25 security issues, the Open Web Application Security Project (OWASP) and its Top 10 issues, the Common Vulnerabilities and Exposures (CVE), National Vulnerability Database (NVD) and the Common Attack Pattern Enumeration and Classification (CAPEC). The CAPEC is a classification of common attacks and helps identify risks to a system (what an attacker would do). In order to better familiarize the developers with this large body of knowledge, the discussion covered what the purpose for each is, what the differences are between them, how they all fit together and how they could help developers. Additional resources were mentioned such as the State of the Art Report (SOAR) on Software Security Assurance, CERT Coding Standards, and the body of work the Department of Homeland Security (DHS) has produced in the Software Assurance Pocket Guide Series. Finally, the introduction laid the context for the class. The class would cover the top 27 CWEs that developers needed to know and understand. The CWEs are hosted by the MITRE Corporation, cosponsored by DHS.

Setup - Insecure Bank & Common Weakness Enumeration

During the class, the CWEs were presented in a single context of a fictitious banking application with various functionality modules similar to the ones that developers may code themselves. Each module (ex. Create user or Account summary), would exhibit two to four CWEs detailing how the weaknesses could be leveraged by an attacker in that area and how developers could help securely code that function. The bulk of the class went through each of the 27 CWEs presented in its own vignette. For each CWE, eight items were discussed to cover the topic fully. A depiction of this structure and the discussion items are listed in Figure 2. As an example, a subset of the vignette on CWE-120, Buffer Overflow is shown in Figure 3.



Secure Coding: CWE Vulnerabilities

Insecure Bank application walkthrough

Modules

Create user

CWE-311 - Missing Encryption of Sensitive Data
CWE-759 - Use of a One-Way Hash without a Salt
CWE-327 - Use of a Broken or Risky Cryptographic Algorithm
CWE-798 - Use of Hard-Coded Credentials

Login

Account summary | Customer support | Mobile banking

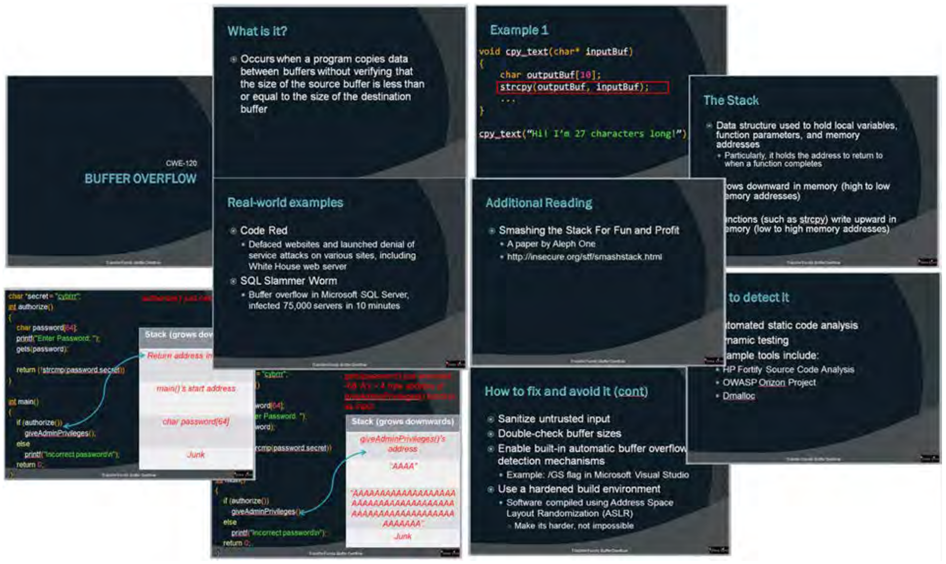
Online deposits | Transfer funds

Account history

For each CWE, we'll cover:

- What is it?
- How likely is it?
- What's the impact?
- Examples
- How to fix and avoid
- How to detect
- Real-world examples
- Hands-on lab exercises

Figure 2: Class Presentation Structure and Discussion Items to Support the Top 27 CWEs



CWE-120

BUFFER OVERFLOW

What is it?

- Occurs when a program copies data between buffers without verifying that the size of the source buffer is less than or equal to the size of the destination buffer

Example 1

```
void cpy_text(char* inputBuf)
{
    char outputBuf[10];
    strcpy(outputBuf, inputBuf);
    ...
}

cpy_text("Hi! I'm 27 characters long!")
```

The Stack

- Data structure used to hold local variables, function parameters, and memory addresses
  - Particularly, it holds the address to return to when a function completes
- rows downward in memory (high to low emory addresses)
- nctions (such as strcpy) write upward in emory (low to high memory addresses)

Real-world examples

- Code Red
  - Defaced websites and launched denial of service attacks on various sites, including White House web server
- SQL Slammer Worm
  - Buffer overflow in Microsoft SQL Server, infected 75,000 servers in 10 minutes

Additional Reading

- Smashing the Stack For Fun and Profit
  - A paper by Aleph One
  - http://insecure.org/stf/smashstack.html

to detect it

- tomated static code analysis
- namic testing
- ample tools include:
  - HP Fortify Source Code Analysis
  - OWASP Orizon Project
  - Dmalloc

```
char *secret = "cybrr";
int authorize()
{
   char password[64];
   printf("Enter Password: ");
   gets(password);

   return (!strcmp(password,secret));
}

int main()
{
   if (authorize())
      giveAdminPrivileges();
   else
      printf("Incorrect password\n");
   return 0;
}
```

How to fix and avoid it (cont)

- Sanitize untrusted input
- Double-check buffer sizes
- Enable built-in automatic buffer overflow detection mechanisms
  - Example: /GS flag in Microsoft Visual Studio
- Use a hardened build environment
  - Software compiled using Address Space Layout Randomization (ASLR)
    - Make its harder, not impossible

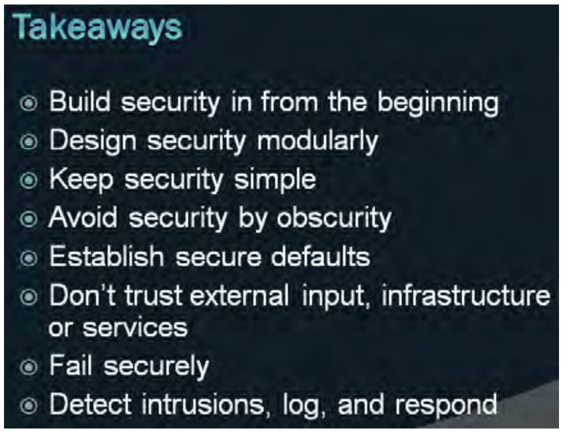
Figure 3: Subset of the CWE-120 Buffer Overflow Vignette

The idea was to break up the volume of knowledge into easily understandable pieces that applied to functions with which developers were already familiar. Additionally, maintaining a single banking application context for all of the CWEs kept the focus on the coding issues rather than focusing on the details of the underlying training applications. This setup would also provide an easy conceptual reference for the future when they wanted to review the information.

The class also covered other topics. Common terms were discussed such as dynamic testing and privilege escalation and the difference between a weakness, vulnerability and exploit. Web programming basics were also covered. While only three of the twenty-seven CWEs were solely for web applications, these basics would support other essential topics such as client-server paradigms when their respective security issues were discussed. Finally, different automated static source code analysis tools were also mentioned.

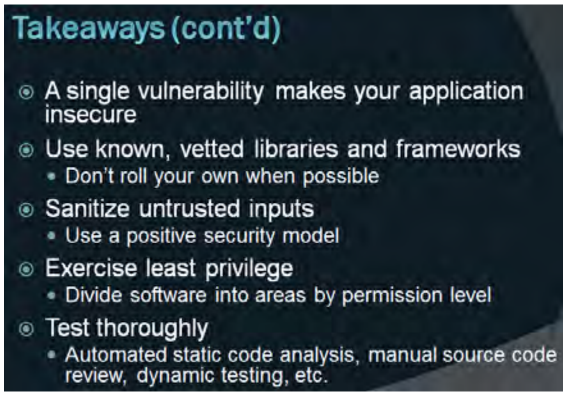
Secure Coding – Summary & Take Away Message

A summary included the important concepts the developers should take away from the class (Figure 4). The class was highly interactive to support the activities and questions from the students. The class covered over 650 slides in two days. However, the students were actively engaged by the structure of the class and remained interested throughout which is a significant success on its own. At the end of class, a handout of references and topics were given to the students as a takeaway. This class is essential for software developers as it introduces the subject of secure coding, widens their aperture and instills ownership to ensure code, applications and systems are developed securely.



Takeaways

- Build security in from the beginning
- Design security modularly
- Keep security simple
- Avoid security by obscurity
- Establish secure defaults
- Don't trust external input, infrastructure or services
- Fail securely
- Detect intrusions, log, and respond



Takeaways (cont'd)

- A single vulnerability makes your application insecure
- Use known, vetted libraries and frameworks
  - Don't roll your own when possible
- Sanitize untrusted inputs
  - Use a positive security model
- Exercise least privilege
  - Divide software into areas by permission level
- Test thoroughly
  - Automated static code analysis, manual source code review, dynamic testing, etc.

Figure 4: Summary of Important Secure Coding Concepts

➜ SOFTWARE ASSURANCE – SECURITY THROUGHOUT THE LIFECYCLE

Just as Hacker 101 set the stage for the importance of Secure Coding, both the Hacker 101 and Secure Coding classes were designed to lead into the final class on Software Assurance titled Secure Software Design, Lifecycle and Testing. The goal for the week was to introduce the concept of Software Assurance and the lifecycle activities which support it. Secure software design, architecture and development processes as well as testing are at the heart of software assurance. Software assurance is defined as the level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software, throughout the life cycle". [5,6] Software Assurance has been around for over a decade in industry. Guidance exists in our DoD instructions and is mandated by congressional National Defense Authorization Acts (NDAA). DoD instruction 5200.44 for Trusted Systems and Networks[5] states that software assurance will be used throughout the lifecycle to manage risk of key systems. NDAA FY13[6] and subsequent NDAAs state that software assurance will be implemented for the entire lifecycle for trusted defense systems. Software assurance is the security component of Software Engineering. Testing activities and tools can only find a small portion of the weaknesses and vulnerabilities in our DoD systems. Security software itself has been noted to introduce $^1/_3$ of the vulnerabilities[7]. Therefore, it lies with the software developers and validators themselves to fundamentally understand the tenants of secure software development, lifecycle processes, and tools in order to best protect and defend ourselves from attacks. The benefit from developers tackling this issue now is to be able to experiment with, inform and select activities that fit well into their current processes. These experiences could support their program manager's requirement to prove how they utilize software assurance on their programs.

This class touched upon secure software development lifecycle models and their components, secure coding recommendations for each phase of the lifecycle, password security and software security testing. As each one of these topics could support a class unto themselves, they were introduced with examples to show their importance and overall place in the lifecycle.

Lifecycle Models, Activities & Recommendations

First, the class introduced the notion of secure software development lifecycle models such as Gary McGraw's Touchpoints, Build Security In Maturity Model (BSIMM), Microsoft's Security Development Lifecycle (SDL) and OWASP's Open Software Assurance Maturity Model (Open SAMM). The class noted examples from the Microsoft SDL that covered the requirements and design phases. Additional examples for the design, implementation, distribution, installation, operation and maintenance and retirement activities were also covered. Each of these topics could require their own separate training so the topics were introduced with resources and examples so that the students were aware of them and their part in the secure development lifecycle. The requirements portion included security requirements, risk analysis and prioritization based on impact and likelihood. The design portion included design, development and test requirements. Recommendations included items such as keeping the code simple so that it is harder to inject malicious code or unintentionally insecure code, using standard libraries instead of creating your own (security by obscurity doesn't work), breaking down components into smaller modules to reduce the scope of privileges each had available, and using Application Program Interfaces (API) with pre-defined statements to control user input. Design activities included analyzing the attack surface to review settings, open ports, services and accounts for security as well as threat modeling to break down the system into components and data flows to identify areas an attacker could target. The Implementation phase mentioned items such as not turning off compiler warnings. Distribution touched on sending the key and crypto checksum separately for additional security. Installation noted configuration choices to support security and removing unneeded components. Operation and Maintenance noted that updates need to be given the same rigor as the initial development. Retirement was mentioned as an issue because backward compatibility supports flaws from past versions.

Passwords

Passwords were discussed as they are a significant security problem, greatly misunderstood and one that users can take action to improve. Random character passwords are almost impossible to remember and, as a result, are often written down (usually on a sticky note under the computer) which subverts security completely (Figure 5). Unfortunately, people are not very adept on their own at creating secure passwords. Passwords are usually created by using family member information; sports or hobby details; quotes from books, television or movies; or with leetspeak which is substituting numbers or symbols for letters such as a "3" for an "e" or "@" for an "a". However, none of these methods are secure. The hacker community has spent a great effort creating dictionaries to help them uncover passwords when these methods are used. The class discussed Diceware passphrases* and highlighted the need for a better password creation paradigm to easily and greatly improve security.

strive to fundamentally understand code weaknesses so they can vigilantly work to keep them from existing in the design

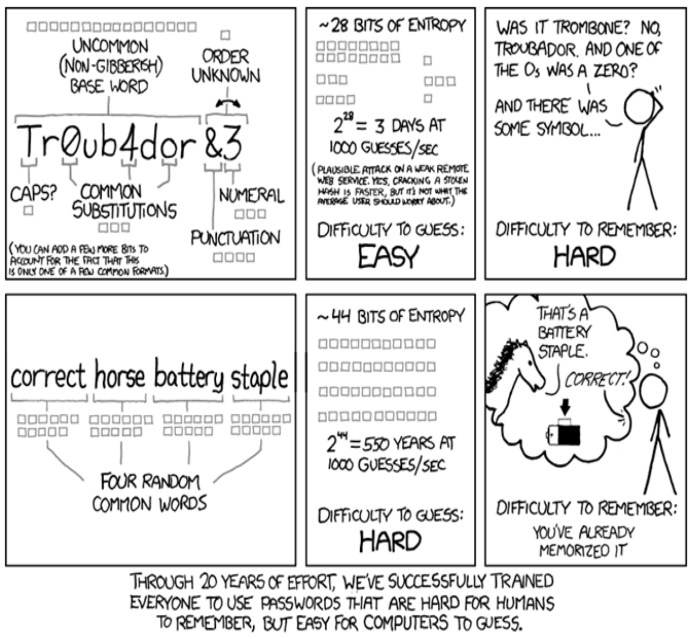

UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor &3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Figure 5: Cartoon of Creating Passwords https://xkcd.com/936/

The issues of memorability, security, calculability and elevation of security were discussed. The Diceware approach involves the user rolling a set of dice and then matching the resulting group of numbers to words in a large Diceware dictionary several time to create a phrase. The Diceware website compares the security strength of their type of password generation to normal techniques to generate sixteen character random passwords. The Diceware passphrase is much easier to remember by grouping the words together in small sets or creating a sentence or small story they tell. This method of selecting random words from a large dictionary results in a mathematically significant number of possible combinations. Selecting "random" words from memory is actually not a large enough dictionary as an individual will usually select from about 2000 words. Additionally, the Diceware words are not related to any specific characteristic of the user which prevents the passphrase from being guessed by an attacker knowing information about the user. The concept of password managers was discussed as well to help solve the problem of the considerable number of different passwords that are needed today.

Testing Tools

Finally, the topic of testing tools to support secure code development was summarized. There exist free open-source and commercially available testing tools. Entities such as the National Security Agency's Center for Assured Software have spent effort comparing these tools and identifying what types of weaknesses they do and do not find. The bottom line is that no one tool will find all the weaknesses in code so using two or three will find the most. Additionally, a data correlation tool will help remove the duplicates that the tools find and can present the data in a report format. These tools can help a developer during the coding process to find accidental issues similar to how compiler errors are used during development. However, even with using multiple tools, a major portion of the weaknesses are not found. This is why it is critical for software developers to strive to fundamentally understand code weaknesses so they can vigilantly work to keep them from existing in the design, architecture and code in the first place.

Software Assurance – Summary & Take Away Message

At the end of the week, the students were surveyed on their experiences with and opinions of the different classes. The responses and discussions were unanimous in that most developers, even experienced ones, had not previously been exposed to secure coding practices or vulnerable code detection. Additionally, they had not been aware of how software could be exploited, the impact of that exploitation or how to detect and fix the vulnerabilities in their code.

Teaching the classes was rewarding as the students remained engaged throughout the entire week and asked good questions. Also, they indicated they were going to take what they had learned back to their offices to check their code for vulnerabilities. Additionally, they requested we provide the training to their team members. Most importantly, they understood their role in the cyber problem; had taken ownership; and would be moving forward to practice software assurance during their daily development lives. Given this feedback, the class had achieved its goals and the training was a success. The next goal was to distribute the training far and wide and also support training the program managers to plan for, budget and support software assurance as a tool to defend our military systems.

Conclusion

Armed with the hacker mindset and safe coding strategies, a developer can "know the enemy" and "know yourself," which if one agrees with Sun Tzu, is the key to being successful at securing our national defenses to win the battles against the cyber enemy. ■

REFERENCES

[1] Feiman Joseph, "Maverick Research: Stop Protecting Your Apps; It's Time for Apps to Protect Themselves". (https://www.gartner.com/doc/2856020/maverick-research-stop-protecting-apps)

[2] Galvin, D., L. Giles, and G. Stade. "Sun Tzu: The Art of War." (2003).

[3] http://theinstitute.ieee.org/special-reports/special-reports/10-recommendations-for-avoiding-software-security-design-flaws

[4] http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf

[5] DoDI 5200.44 - Protection of Mission Critical Functions to Achieve Trusted Systems and Networks (TSN) - (http://www.dtic.mil/whs/directives/corres/pdf/520044p.pdf)

[6] National Defense Authorization Act for Fiscal Year 2013 (2013 NDAA S933) (http://www.dtic.mil/congressional_budget/pdfs/FY2013_pdfs/AUTH_CRPT-112hrpt705.pdf)

[7] http://www.cyber.umd.edu/sites/default/files/documents/symposium/fisher-HACMS-MD.pdf

[8] http://world.std.com/~reinhold/diceware.html

ABOUT THE AUTHOR(S)

Carol Lee works at the Naval Surface Warfare Center in Dahlgren, Virginia. She is the Navy counterpart lead for the Joint Federated Assurance Center Software Assurance Technical Working Group. Mrs. Lee has an M.S. in Computer Science from Virginia Commonwealth University (VCU) and over 15 years of experience in leading software development teams and developing software products for the Navy and DoD. She has developed code for mathematical algorithms for the Statistical Modeling and Estimation of Reliability Functions for Systems (SMERFS3) project; worked on web collaboration software; and lead teams in the development of tactical decision aids, a satellite operation center, mobile handset command and control and cyber situational awareness applications. Additionally, she created the vision for the training in this article and has built a Software Assurance and penetration testing team.

Jasen Moran is a computer scientist for the Department of Defense. He has a Master of Science degree in Secure Software Engineering and a Bachelor's degree in Computer Science, both from James Madison University. His professional areas of interest include network security, digital forensics, application hardening and all things Python.

Joel McCormick is a lead software developer with 15 years of experience in the field, including significant time spent in C and C++. He works at the Naval Surface Warfare Center in Dahlgren, Virginia and has a degree in Computer Science and certifications in exploit research and development.

Kolby Hoover is currently working for the Naval Surface Warfare Center Dahlgren Division as a technical lead for the newly formed NAVSEA Red Team. He has a B.S. in Computer Engineering from Christopher Newport University and is currently working on his Master's Degree in Cybersecurity Engineering from the University of Maryland.

Matt Hackman is a software engineer currently working with the Department of Defense. He has fifteen years of experience in the engineering and software development world. He has lead and contributed to projects involving a wide range of subject matter including: cyber security, machine learning, chemical modeling, satellite communications, geospatial modeling and analysis, and mission assurance. Mr. Hackman holds degrees in Computer Science and Chemistry and certifications in cyber security incident response handling.

Paul McFall is currently working for the Naval Surface Warfare Center Dahlgren Division as a technical lead for the newly formed NAVSEA Red Team. He has a M.S. in Computer Engineering.

Roger Lamb is a software developer at NSWC Dahlgren Division working on cyber situational awareness tools. Mr. Lamb has a Master of Science degree in Computer Science from Virginia Commonwealth University and a B.S. in Computer Science from University of Mary Washington. Mr. Lamb's experiences include software work in test development, satellite radios, cyber situation tools, mobile development, and various works in open source projects. He was has a certificate from Virginia Commonwealth University in Cyber security.

Scott Nickeson has a B.S. in Computer Science from Georgia Institute of Technology and 15 years of professional software development experience at NSWCDD.

IS OUR SOFTWARE REALLY SECURE?

By: Francis (Frank) Mayer

The answer to the question is NO – as noted in the DoD Director, Operational Test and Evaluation FY 2016 Annual Report[1] despite the significant progress the DoD has made in improving the cybersecurity of DoD programs and networks "missions remain at risk when subjected to cyber-attacks emulating an advanced nation-state adversary." The challenge of assuring that our software will only operate as intended is formidable given the ever-growing complexity of systems and networks. Considerations include the globalization of the defense industrial base, the cost-consciousness and competitiveness of many suppliers, concerns about the insertion of malicious functionality in software and heightened awareness of adversaries targeting DoD supply chains. "Black box" software functionality testing without knowledge of how the internal structure or logic will process the input will not catch many of the critical defects in software.

To address this challenge DODI 5000.02, Operation of the Defense Acquisition System Incorporating Change 2, Effective February 2, 2017[2], states that Program Managers will implement the use of automated software vulnerability detection and analysis tools and ensure risk-based remediation of software vulnerabilities is addressed in Program Protection Plans (PPPs), included in contract requirements, and verified through continued use of such tools and testing (as required by section 933 of Public Law 112-239)[3].

What is Software Assurance (SwA) and why should we care about SwA?

SwA is "The level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software throughout the lifecycle." -- Committee on National Security Systems Instruction (CNSSI) 4009 – April 2015[4].

We need SwA because Mission Critical Defense Systems (MCDS) built with inadequate security and unknown but critical flaws put military data, operations and sensitive information at significant risk, especially given that most of these systems operate on the Department of Defense Information Networks (DoDIN)[5]. Successive National Defense Authorization Acts (NDAAs) have identified the need for SwA as evidence of US Congressional and Presidential support for SwA. Section 933 of the 2013 NDAA mandated that the DoD implement a baseline SwA policy.

Major DoD Baseline SwA policy and key provisions of it are shown in Figure 1:

- **DoDI 5200.44**[6], "Protection of Mission-Critical Functions to Achieve Trusted Systems and Networks (TSN)," Incorporating Change 1, Effective August 25, 2016.
- **DoDI 5000.02**, Operation of the Defense Acquisition System, Incorporating Change 2, Effective February 2, 2017
- **DoDI 8500.01**[7], Cybersecurity, 14 March 2014
- **DoDI 8510.01**[8], Risk Management Framework (RMF), Incorporating Change 1, Effective May 24, 2016
- **CJCSI 6510.01F**[9], Information Assurance (IA) and Support to Computer Network Defense (CND) , Directive Current as of 9 Jun 2015

When acquiring systems managers are faced with the difficult task of balancing software performance, cost, and schedule trade-offs and the level of security needed to provide "survivability" of the resulting mission capability. Managers and system owners address survivability for hardware, such as for a combat vehicle, by engaging experts to address the vehicle's ability to withstand likely kinetic threats and then checking that the vehicles coming off the assembly line are built to meet the threat. While software is different, a similar approach should work for software.

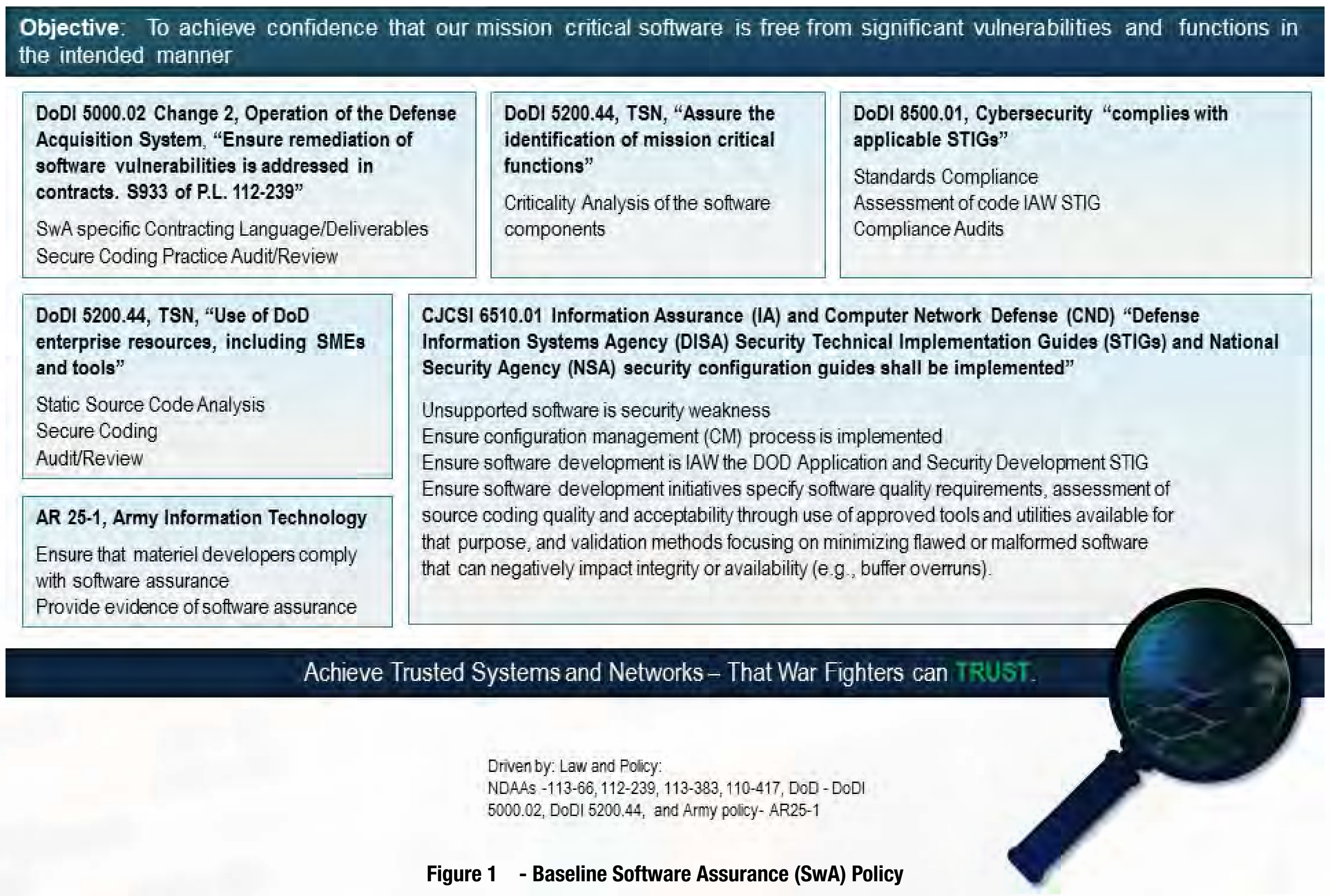
The current overly compliance focused approach often puts less emphasis on building the software capability to address likely advanced threats and expends considerable effort getting over the regulatory "speed bumps" to meet the "schedule" at the end of the Software Development Life Cycle (SDLC). Given the advanced threat, the approach focused only on compliance may not be effective. This potential ineffectiveness is indicated in the DoD Director, Operational Test and Evaluation Reports, and DoD Inspector General Reports[10] specifically the, DoD Cybersecurity Weaknesses as Reported in Audit Reports Issued From August 1, 2015, Through July 31, 2016[11], that identifies the need for "implementing secure information systems on major weapons systems throughout their lifecycle requires effective and continuous software assurance testing."

implementing secure information systems on major weapons systems throughout their lifecycle requires effective and continuous software assurance testing

The DoD has developed a significant body of information to aid in addressing the challenge, for example the Program Manager's Guidebook for Integrating the Cybersecurity Risk Management Framework (RMF) into the System Acquisition Lifecycle[12] helps program managers (PM) and their staffs clearly understand how to integrate cybersecurity into their programs throughout the system lifecycle in accordance with the Risk Management Framework (RMF). This guidebook identifies software assurance as a systems security engineering activity that is a countermeasure that mitigates cybersecurity risks.

An overall governance process that includes clear mandates for third party SwA assessments, along with a set of practices for ensuring proactive application security, provides the objective perspective and motivation to maintain an effective program to help address making sure the warfighters can trust the software product. Objective assessments must be implemented early on and continuously, to provide the powerful "forcing function" needed to get the developer to implement a fully integrated software assurance discipline throughout the SDLC.

At the system level, it is critical to first establish a coherent and disciplined process by developing a plan and statement of requirements for software assurance early in the acquisition lifecycle. This requires us to incorporate Cybersecurity, which includes software assurance, requirements into the requests for proposal (RFP). Programs then need to use the plan to track software assurance protection throughout the acquisition. The progress toward achieving the plan needs to be measured by actual results that are reported at each of the Systems Engineering Technical Reviews (SETR) just as noted in 2014 Deputy Assistant Secretary of Defense for Systems Engineering and Department



Objective: To achieve confidence that our mission critical software is free from significant vulnerabilities and functions in the intended manner

DoDI 5000.02 Change 2, Operation of the Defense Acquisition System, "Ensure remediation of software vulnerabilities is addressed in contracts. S933 of P.L. 112-239"

SwA specific Contracting Language/Deliverables
Secure Coding Practice Audit/Review

DoDI 5200.44, TSN, "Assure the identification of mission critical functions"

Criticality Analysis of the software components

DoDI 8500.01, Cybersecurity "complies with applicable STIGs"

Standards Compliance
Assessment of code IAW STIG
Compliance Audits

DoDI 5200.44, TSN, "Use of DoD enterprise resources, including SMEs and tools"

Static Source Code Analysis
Secure Coding
Audit/Review

CJCSI 6510.01 Information Assurance (IA) and Computer Network Defense (CND) "Defense Information Systems Agency (DISA) Security Technical Implementation Guides (STIGs) and National Security Agency (NSA) security configuration guides shall be implemented"

Unsupported software is security weakness
Ensure configuration management (CM) process is implemented
Ensure software development is IAW the DOD Application and Security Development STIG
Ensure software development initiatives specify software quality requirements, assessment of source coding quality and acceptability through use of approved tools and utilities available for that purpose, and validation methods focusing on minimizing flawed or malformed software that can negatively impact integrity or availability (e.g., buffer overruns).

AR 25-1, Army Information Technology

Ensure that materiel developers comply with software assurance
Provide evidence of software assurance

Achieve Trusted Systems and Networks – That War Fighters can TRUST.

Driven by: Law and Policy:
NDAAs -113-66, 112-239, 113-383, 110-417, DoD - DoDI 5000.02, DoDI 5200.44, and Army policy- AR25-1

Figure 1 - Baseline Software Assurance (SwA) Policy

of Defense Chief Information Officer's Software Assurance Countermeasures in Program Protection Planning[13] guide.

This level of rigor is economically justified because it saves resources in the long run, as noted in the Software Engineering Institute Special Report: Making the Business Case for Software Assurance[14]. This report provides evidence of the business case for SwA.

At both the system and enterprise level it is necessary to place more emphasis on developing the capability and capacity to leverage a broad range of software assessment tools and techniques for our portfolio of systems. For example, we need the capability for more "White Box Testing" - structural testing with insight into the internal logic and software structure such as static software code assessments using multiple tools. We also need enhanced capacity, to include enough well trained and motivated people, to actually perform the testing consistently across our portfolio of systems and to work with developers and maintainers to implement effective solutions.

Policy needs to be not only just enforced but also supported by a community

The third and critical step in succeeding in implementing an enduring SwA program is developing, executing, and then maintaining a SwA enterprise level strategic plan that addresses the planning, execution, capability and capacity to build security in[15].

At the DoD and Army level action has been taken to establish and support the Joint Federated Assurance Center (JFAC)[16]. Section 937 of the National Defense Authorization Act (NDAA) for Fiscal Year (FY) 2014 directed the Department of Defense (DoD) to establish a federation of capabilities to support trusted defense systems and ensure the security of software and hardware developed, acquired, maintained, and used by the Department. The JFAC Service Providers that help deliver the capabilities of the JFAC are available to assist program managers, developers, and maintainers in implementing an effective software assurance program.

Communications - Electronics Command (CECOM) has taken action by championing and supporting SwA. CECOM Software Engineering Center's (SEC's)[17] current software assurance program strategy that we developed using our lessons learned is based on three Lines of Effort (LoE):

1. **SwA Infrastructure:** Establish a sound SwA Infrastructure as a key enabler for SwA. Discover, develop, objectively assess, and then implement "best in breed" software assurance, mobile application, cyber-security and malicious code scan tools. Using the "best in breed" tools and techniques, create a common well-resourced enterprise software engineering capability that team members can leverage, rather than continuing with the current patchwork sets of capabilities. Resource the infrastructure by planning, programming, budgeting and executing the resources to put the infrastructure in place and to keep it relevant and ready
2. **Governance:** As we all know, a major program needs good requirements and senior leader support to succeed. SwA is no different. To do this it is necessary to leverage the best practices, requirements, emerging threat, and lessons learned from other stakeholders to include Department Level Stakeholders to include user representatives from the major commands, the research community, the acquisition community, Chief Information Officers (CIOs), the intelligence community, United States Cyber Command (USCYBERCOM), Department of Homeland Security (DHS), and National and Security Agency (NSA) Center for Assured Software (CAS) so that our governance approach remains relevant and unified. Policy needs to be not only just enforced but also supported by a community that stands ready to support program manages and application developers and maintainers with the formidable task of engineering in security and then maintaining the security of the software baseline.
3. **Workforce Development:** Develop, educate, motivate, and train the workforce. Conduct a strategic communications campaign for our workforce, partners, and leaders to promote the vision and purpose of SwA. Change the culture of our workforce so that they embrace software assurance & cyber-security. Provide educational experiences for the developers and sustainers to address both the theory and engineering application relevant to cybersecurity, which includes software assurance. Provide formal training experiences to the workforce, to include baseline cybersecurity certification training and training on specific and relevant technologies. Provide the workforce with professionally mentored "hands-on" work experience in applying software assurance practices, to include using cyber-security scan tools and implementing Tactics, Techniques, and Procedures (TTPs). Document and track training so that managers can make sure it is happening. This includes making sure that properly applying software assurance TTPs becomes part of performance objectives for all software engineering employees and as part of what we demand in contracts for our supporting contractor workforce.

In Conclusion, to effectively defend against the threats our systems and networks face a collaborative approach is really needed to understand the current and evolving threat, to develop and maintain effective solutions, to proactively address weaknesses in both our systems and software, and to make the smart trade-offs needed between functional mission capabilities and a viable security poster. Program managers, developers, system engineers, software engineers, the intelligence community, the operational organizations that use DoD systems and software, and expert service providers, such as the JFAC Service Providers, need to embrace a spirit of collaboration and team work because no single person or organization has all the knowledge or capability needed to address the daunting problem of assuring software by themselves. A successful program is about more than just simply measuring compliance and making fixes; it needs a unified team effort that is focused on real results that reduce risk given the current threat in a way that contributes to both survivability and mission effectiveness. ■

ABOUT THE AUTHOR

Frank Mayer has over thirty-seven years of service with the United States Army (USA). He served as an activated reserve officer assigned to Software Engineering Center (SEC) from 2001 until 2003. He has been a Department of the Army Civilian with SEC since 2003. He also has over seven years of corporate world technical experience to include work in the areas of Independent Verification and Validation (IV&V), software testing, project lead for the testing of a software intensive system, and cybersecurity. He has held both technical level positions and management positions as a Department of the Army Civilian. He is a Certified Information Systems Security Professional (CISSP) and holds a Master's Degree in Systems Management from Capitol College, Laurel Maryland, with a Graduate Certificate in Technology Management, Systems Integration, Systems Acquisition Management, Management of Research, Development, Testing and Evaluation. He is an Acquisition Corps member and is Certified Level III in both Systems Planning, Research, Development and Engineering and Information Technology.

He is currently assigned to the Unites States Army Communications Electronics Command's Software Engineering Center Services Directorate, can be contacted at e-mail francis.l.mayer.civ@mail.mil.

ENDNOTES

[1] Operational Test & Evaluation Office of the Secretary of Defense, Fiscal Year (FY) 2016 Report, Retrieved from http://www.dote.osd.mil/pub/reports/FY2016/pdf/other/2016cybersecurity.pdf

[2] DODI 5000.02, Operation of the Defense Acquisition System Incorporating Change 2, Effective February 2, 2017, Retrieved from http://www.dtic.mil/whs/directives/corres/pdf/500002_dodi_2015.pdf

[3] 112th Congress Public Law 239, U.S. Government Printing Office, Page 1631, National Defense Authorization Act for Fiscal Year 2013, Retrieved from https://www.gpo.gov/fdsys/pkg/PLAW-112publ239/html/PLAW-112publ239.htm

[4] Committee on National Security Systems Instruction (CNSSI) 4009 – April 2015, Retrieved from https://www.cnss.gov/CNSS/issuances/Instructions.cfm

[5] Defense Information Systems Agency Strategic Plan 2015-2020, Retrieved from http://www.disa.mil/~/media/files/disa/about/strategic-plan.pdf

[6] DoDI 5200.44, Protection of Mission Critical Functions to Achieve Trusted Systems and Networks (TSN), *Incorporating Change 1, Effective August 25, 2016* Retrieved from http://www.dtic.mil/whs/directives/corres/pdf/520044p.pdf

[7] DoDI 8500.01, Cybersecurity, March 14, 2014, Retrieved from http://www.dtic.mil/whs/directives/corres/pdf/850001_2014.pdf

[8] DoDI 8510.01, Risk Management Framework (RMF) for DoD Information Technology (IT), Incorporating Change 1, Effective May 24, 2016, Retrieved from http://www.dtic.mil/whs/directives/corres/pdf/851001_2014.pdf

[9] Chairman of the Joint Chiefs of Staff Instruction (CJCSI) 6510.01F, Directive Current as of 9 June 2015, Information Assurance (IA) and Support to Computer Network Defense, Retrieved from http://www.dtic.mil/cjcs_directives/cdata/unlimit/6510_01.pdf

[10] U.S. Department of Defense, Inspector General, Consolidated Listing of Reports, Retrieved from http://www.dodig.mil/pubs/index.cfm

[11] Cyber Security, DoD Cybersecurity Weaknesses as Reported in Audit Reports Issued From August 1, 2015, Through July 31, 2016 (Redacted) (Project No. D2016-D000RB-0139.000), Retrieved from http://www.dodig.mil/pubs/report_summary.cfm?id=7235

[12] Program Manager's Guidebook for Integrating the Cybersecurity Risk Management Framework (RMF) into the System Acquisition Lifecycle, Cleared for Open Publication, May 26, 2015 Retrieved from https://acc.dau.mil/adl/en-US/722603/file/80119/Cybersecurity%20Guidebook%20v1_0%20with%20publication%20notice.pdf

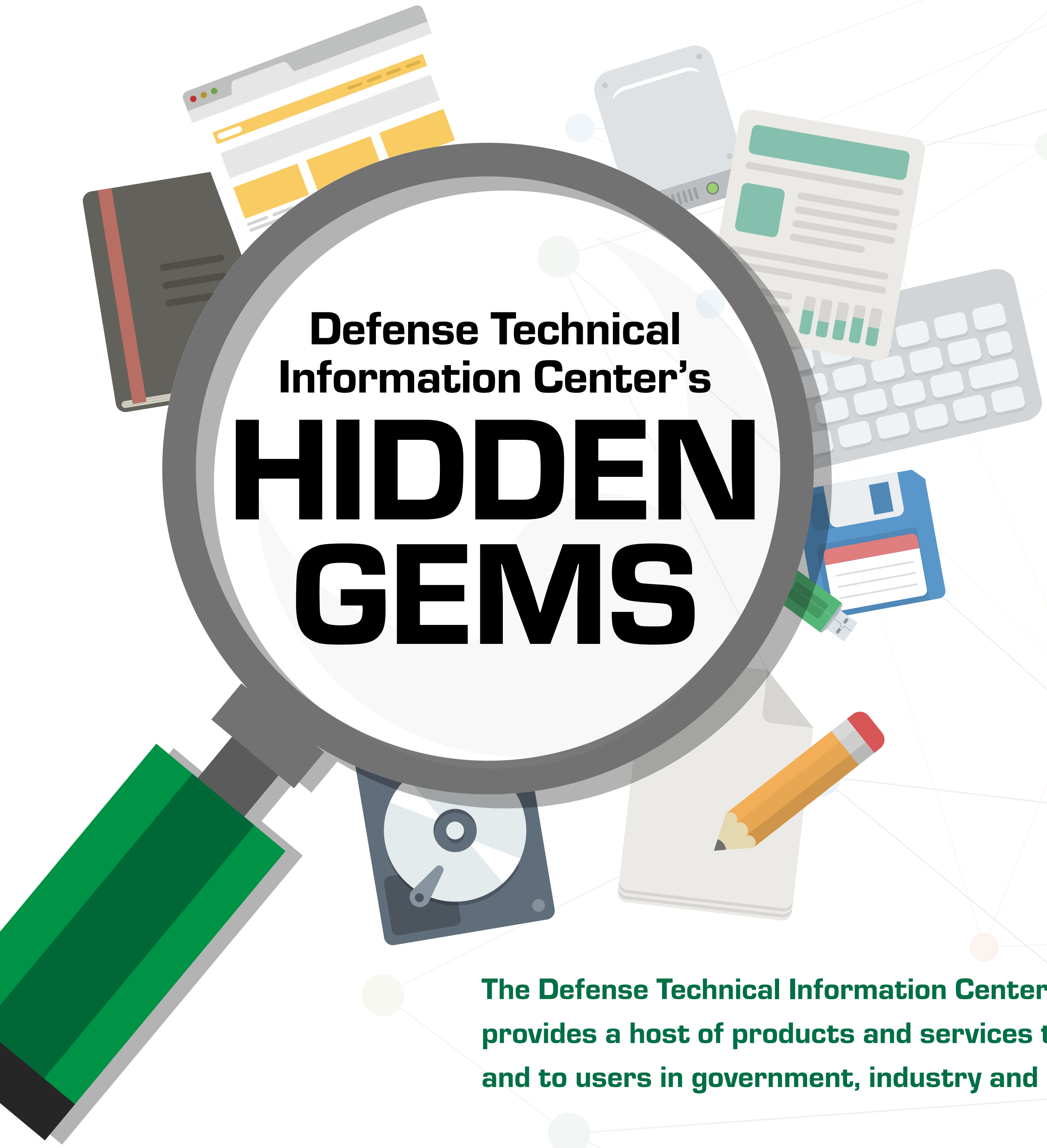
[13] Deputy Assistant Secretary of Defense for Systems Engineering and Department of Defense Chief Information Officer, Software Assurance Countermeasures in Program Protection Planning, dated March 2014, Retrieved from http://www.acq.osd.mil/se/docs/SwA-CM-in-PPP.pdf

[14] Mead, N.R., Allen, J.H., Conklin, W.A., Drommi, A., Harrison, J., Ingalsbe J., Rainey, J., Shoemaker, D. (University of Detroit Mercy), (April 2009) Making the Business Case for Software Assurance, Software Engineering Institute, CMU/SEI Report Number: CMU/SEI-2009-SR-001 Retrieved from http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=8831

[15] *Build Security In / Software & Supply Chain Assurance content is no longer updated. The reference is provided for historical reference* Retrieved from https://www.us-cert.gov/bsi

[16] Baldwin, K. (2014), Department of Defense (DoD) Joint Federated Assurance Center (JFAC) Overview, 17th Annual NDIA Systems Engineering Conference, Retrieved from http://www.acq.osd.mil/se/briefs/16950-2014_10_29_NDIA-SEC-Baldwin-JFAC-vF.pdf

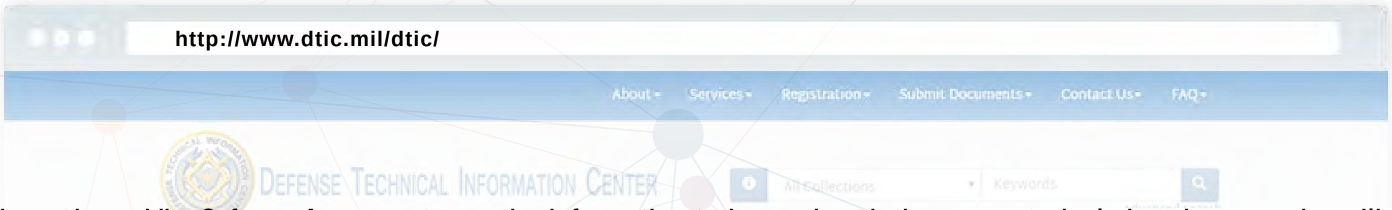
[17] CECOM SEC Software Assurance for the Acquisition Enterprise (2017), Retrieved from http://www.sec.army.mil/secweb/corecomp-CyberSA.html



Defense Technical Information Center's HIDDEN GEMS

The Defense Technical Information Center (DTIC) provides a host of products and services to the DoD and to users in government, industry and academia.

One of the important facets of their services is access to a huge trove of scientific and technical information (STI) covering close to seven decades of military research and development (R&D). This article, and succeeding articles in future CSIAC Journals, covers one small area of STI and its roots in 20th century military R&D by identifying early documents that addressed new concepts and ideas. These "hidden gems" are interesting and relevant for two reasons – first, it highlights the early investments that the DoD has made in virtually every aspect of science and technology; and second, it provides a glimpse into the underlying fundamentals which we are still researching today, including ideas and concepts that are surprisingly cutting edge and "ancient" (in technology timelines) at the same time. Since this is a special edition of the CSIAC Journal focusing on Software Assurance, here are a few hidden gems from the DTIC treasure chest that might warrant a closer look from both the curious and the serious researcher. Who knows, it might lead you in a new and different direction. To quote Winston Churchill, the farther backward you can look, the farther forward you can see.



Inherent in providing Software Assurance to a community is a testing methodology that provides a set of guarantees. Static Testing is an approach to looking through code and algorithms to find out what should happen, and checking out what it should do in as formal a way possible without executing the code explicitly. Some of the first concepts in this area came from military research. Our first document is from 1976, 41 years ago, and is titled "Protection Errors in Operating Systems: Validation of Critical Conditions", available at ***http://www.dtic.mil/get-tr-doc/pdf?AD=ADA026442***. This document recognizes and provides methodologies for reasoning about validation of complex software. There is also a very good description on page 6 (page 14 from the cover, pages aren't numbered) of a basic principle for validation of operating system kernel operations at the time of invocation. It identifies the possibility of incorrect operation when the timing of invocation is not consistent with the state of the entities being acted on; something that much later came to be called Time of Check/Time of Use (TOCTOU) errors or attacks.

Just to show the inclusion of concepts in early DoD guidance that we still find ourselves grappling with or discovering anew, the following two documents from 1972 and 1988 apply almost as well now as they tried to apply then. The second document we highlight is a DoD Manual from 1972, the ADP Security Manual DoD 5200.28-M (available from DTIC at ***http://www.dtic.mil/dtic/tr/fulltext/u2/a268995.pdf***). Need a definition of risk management applied to military computer systems? Think risk management is a 21st century concept? Look on page 2, where the following information is provided:

The potential means by which a computer system can be adequately secured are virtually unlimited. The safeguards adopted must be consistent with available technology, the frequency of processing, the classification of the data handled or the information to be produced, the environment in which the ADP System operates, the degree of risk which can be tolerated, and other factors which may be unique to the installation involved… … it is understood that all of the techniques described in this manual may not be economically justified after a cost versus risk evaluation. Therefore, selected subsets of the techniques included in this manual, with appropriate trade-offs, may be used to gain the level of security required for classification category, etc., to be secured.

Not bad for 45 years ago. There are additional ideas contained in this document and many others that highlight early perspectives on risk, malware, network vulnerabilities, etc. In the 1960's and 1970's, many of the ideas were not practically or technologically implemented, but the germinal ideas and coherent thinking about what was to come in computers, software and networks was well developed. Look at page 23 and 24 regarding the protections that should ensue from the operating system itself (although, at this point in time, the difference between operating system aspects and hardware-specific aspects were a bit different…)

- The execution state of a processor should include one or more variables, i.e., "protection state variables," which determine the interpretation of instructions executed by the processor. For example, a processor might have a master mode/user mode protection state variable, in which certain instructions are illegal except in master mode. Modification of the protection state variables shall be so constrained by the operating system and hardware that a user cannot access information for which he has no authorization.
- The ability of a processor to access locations in memory (hereinafter to include primary and auxiliary memory) should be controlled (e.g., in user mode, a memory access control register might allow access only to memory locations allocated to the user by the O/S).
- All possible operation codes, with all possible tags or modifiers, whether legal or not, should produce known responses by the computer.
- Error detection should be performed on each fetch cycle of an instruction and its operant (e. g., parity check and address bounds check).

Where would buffer overflows be if the last one had been integrated more completely over the last 45 years?

If we chase this document lineage forward to 1988, to DoD Directive 5200.18 from 1988, we can see the evolution of awareness for security across computer systems and networks. The document is now titled "Security Requirements for Automated Information Systems (AIS)", and it applies across classified, sensitive, and unclassified information. Also note the change from ADP to AIS. The earlier document was geared specifically for classified systems, thought at the time to be the only computer systems needing enhanced protections. This third document can be found at ***http://www.dtic.mil/dtic/tr/fulltext/u2/a272815.pdf***. It reflects the more advanced state of computers and networks at the time, and includes more guidance on risk management, accreditation, information sensitivity, etc.

Of course, it is easy in hindsight to pick the best parts and identify where they could have been helpful over time; that is not the intent here. We'd like to identify documents that convey foundational thought and concepts that help us place "where we are" in a stronger context, and point to ideas that are consistent across generations. It is amazing what you can find when you look. ■

DEVELOPMENT AND TRANSITION OF THE SEI SOFTWARE ASSURANCE CURRICULUM

By: Nancy R. Mead and Carol C. Woody, Software Engineering Institute, Carnegie Mellon University

In this article, we discuss the development and transition of the Software Engineering Institute's (SEI's) Software Assurance Curriculum. The Master of Software Assurance Reference Curriculum, developed under U.S. Department of Homeland Security (DHS) sponsorship, was endorsed by the Association for Computing Machinery (ACM) and IEEE Computer Society. Additional curriculum recommendations were made at the undergraduate and community college levels. Subsequently, a transition effort was undertaken that included more than 20 papers, keynote talks, and presentations. The Securely Provision section of the National Initiative for Cybersecurity Education (NICE) curriculum is based on the software assurance (SwA) curriculum work that preceded it. Transition of the SwA Curriculum also included faculty workshops, a LinkedIn group, transition to graduate programs, and course development. The SEI maintains a website on the SwA Curriculum Project that includes all of the documentation, donated course materials, and courses developed in-house. An important partnership between the SEI, the Central Illinois Center of Excellence for Secure Software (CICESS), and Illinois Central College (ICC) resulted in the creation of a two-year degree program in Secure Software Development. That program incorporated an apprenticeship model and the SEI's software assurance curriculum recommendations at the community college level. Subsequently, a one-semester course on assured software development at the master's level was modified and repurposed for delivery to the Space and Naval Warfare Systems Command, San Diego (SPAWAR SD). The SPAWAR SD audience included trainers and developers. In the future, we hope to continue our transition efforts with additional collaborations and course development.

The Need for SwA Education

Although software is ubiquitous in modern systems, the complexity of software and software-intensive systems poses inherent risk. This complexity, along with our reliance on these systems, suggests that attackers need to take down only the most vulnerable component to have far-reaching and damaging effects on the larger system. In this environment, attackers no longer need to possess technical sophistication. Due to the growing supply of shared attack strategies, an unsophisticated attacker can easily acquire and launch a sophisticated attack.

On the bright side, in recent years considerable research has been done to explore ways of developing assured software that is resistant to attack and capable of recovering from one. However, much of that research has not made its way into software engineering practice, nor is it routinely taught at our universities.

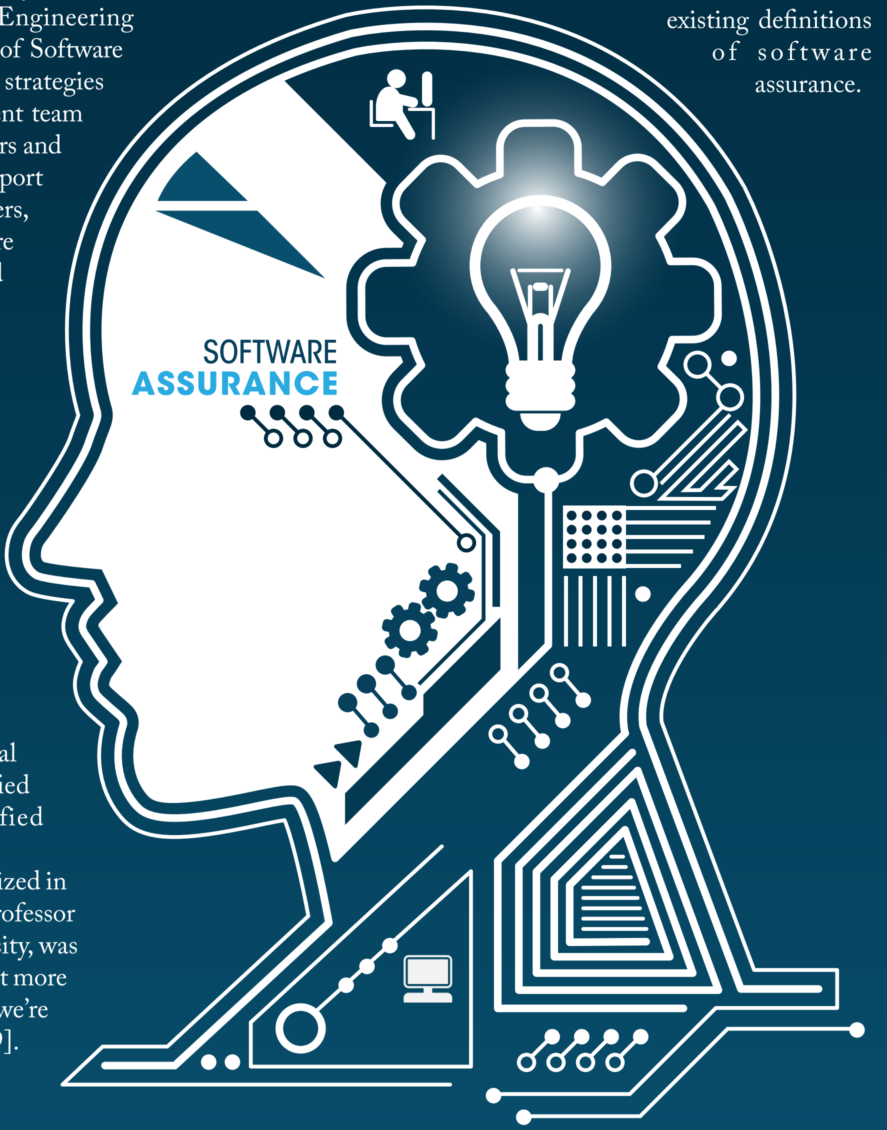
To address this disconnect between research, education, and the practical development of assured software, the U.S. Department of Homeland Security (DHS) National Cyber Security Division (NCSD) enlisted the Carnegie Mellon Software Engineering Institute (SEI) to develop a curriculum for a Master of Software Assurance degree program and to define transition strategies for future implementation. The curriculum development team that was assembled included a mix of SEI staff members and university faculty, with editorial and administrative support provided by the SEI. The development team members, collectively, had a considerable background in software assurance research, software engineering research and practice, and software engineering education.

As noted in our curriculum report, the need for a master's level program in this discipline has been growing for years [Mead 2010a]:

- A study by the nonpartisan Partnership for Public Service points out, "The pipeline of new talent [with the skills to ensure the security of software systems] is inadequate. . . . only 40 percent of CIOs [chief information officers], CISOs [chief information security officers] and IT [information technology] hiring managers are satisfied or very satisfied with the quality of applicants applying for federal cybersecurity jobs, and only 30 percent are satisfied or very satisfied with the number of qualified candidates who are applying" [PPP 2009].
- The need for cybersecurity education was emphasized in the New York Times when Dr. Nasir Memon, a professor at the Polytechnic Institute of New York University, was quoted as saying, "There is a huge demand, and a lot more schools have created programs, but to be honest, we're still not producing enough students" [Drew 2009].
- In discussions with industry and government representatives, we have found that the need for more capacity in cybersecurity continues to grow. Anecdotal feedback from the development team members' own students indicates that even a single course with a cybersecurity focus enhances their positioning in the job market. They felt that they were given job offers they would not have received otherwise.

Another aspect of the need for cybersecurity education occurs in educational institutions. Based on our collective experience in software engineering education, we know it can be very difficult to start a new program or track from scratch, and we want to assist organizations and faculty members who wish to undertake such an endeavor. Our objective is to support their needs, while recognizing that there are many implementation strategies.

Recognizing that software assurance is not exactly the same as software engineering or information security, one of our first tasks was to review existing definitions of software assurance.



We evolved from a definition that was in wide use [CNSS 2009] to one that we thought was a better fit for the curriculum work: "Software assurance (SwA) is the application of technologies and processes to achieve a required level of confidence that software systems and services function in the intended manner, are free from accidental or intentional vulnerabilities, provide security capabilities appropriate to the threat environment, and recover from intrusions and failures" [Mead 2010a].

This definition emphasizes the importance of both technologies and processes in software assurance, notes that computing capabilities may be acquired through services as well as new development, acknowledges the need for correct functionality, recognizes that security capabilities must be appropriate to the threat environment, and identifies recovery from intrusions and failures as an important capability for organizational continuity and survival.

While information security is important, academic programs in information security typically focus on system administrator activities for operational systems, whereas our focus was on systems under development. Software engineering provides ample excellent foundational material, and all the curriculum development team members have a software engineering background. However, we recognized that the development of assured software needs to go beyond good software engineering practice, and indeed the resulting curriculum reflects this.

Software engineering provides ample excellent foundational material

In the remainder of this article, we discuss our sources, the curriculum development process, our SwA education products, transition/adoption strategies, and adoption.

SwA Curriculum Development Process

We followed this eight-step process to develop the curriculum recommendations:

1. **Develop Project Guidelines:** We adapted a set of guidelines similar to the GSwE2009 project to fit our needs. These adapted guidelines helped to direct our work, especially when we were developing Outcomes and the Body of Knowledge (see Step 6)
2. **Identify and Review Sources:** We reviewed about 30 respected sources of security practices, including well-known textbooks and courses. These sources were particularly helpful in expanding details of the defined topics (see Step 3) and outcomes (see Step 6).
3. **Define Topics:** We expanded on the main topics from [Allen 2008] to identify important topics and practices throughout the software development lifecycle (SDLC). These topics served as a first step toward organizing all the material needed in the curriculum.
4. **Define SDLC Practices and Categories:** We expanded each topic (from the previous step) to the level of specific security practices used in industry, government, and academia. The sources identified in Step 2 were used to ensure that we included as many different practices as possible. Then we grouped related practices into higher level categories.
5. **Solicit External Feedback:** At this point, we asked practitioners, managers, and educators for feedback on our content so far. We were particularly interested in knowing whether graduates who acquired the knowledge and skills we had described would be valuable in their assigned positions. Results from a three-page questionnaire were used to revise our practices and categories.
6. **Develop Outcomes, Body of Knowledge, Curriculum Architecture, Course Descriptions, and Implementation Guidance:** We developed expected outcomes for graduates of a software assurance program starting with the categories we identified in Step 4. We also elaborated the categories and practices into a body of knowledge to be mastered by students. We developed a curriculum architecture and a set of example course outlines to be used in creating an academic program, and we produced some implementation guidance for faculty who might take on such a task.
7. **Compare Knowledge Units from the Body of Knowledge to SDLC Practices:** We checked to see that all the practices identified in Step 4 were adequately covered by the knowledge units of our body of knowledge. This analysis led to some minor revisions in both the body of knowledge and the outcomes.
8. **Conduct External Reviews and Make Revisions:** Finally, we solicited feedback from external reviewers in academia, industry, and government and made appropriate revisions.

Figure 1 shows the relationship of project artifacts to our process.

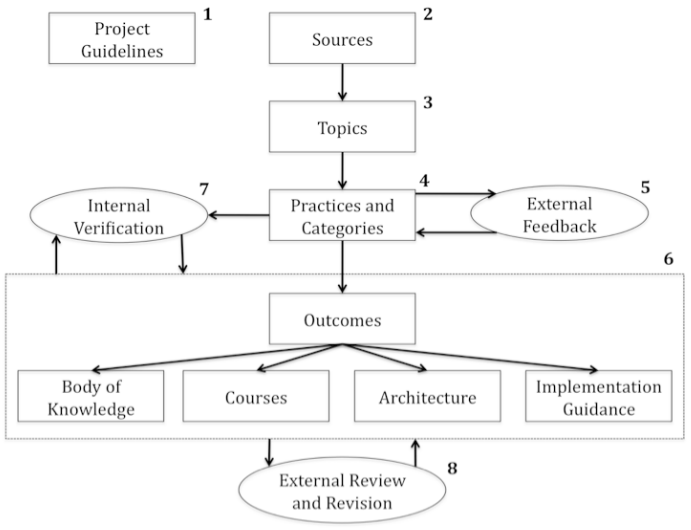


Figure 1. Relationship of Project Artifacts to Our Curriculum Development Process

As an example, an important artifact of the process was the body of knowledge, which included seven outcome areas. Brief descriptions of the outcomes follow:

Name	Description
Volume I: Master of Software Assurance Reference Curriculum	Provides material for establishing or revising a Master of Software Assurance (MSwA) program: curriculum development guidelines, graduate student outcomes, recommended student preparation, an SwA body of knowledge, a high-level MSwA curriculum architecture, and implementation guidelines
Volume II: Undergraduate Course Outlines	Provides the syllabi for seven undergraduate SwA courses: Computer Science I and II, Introduction to Computer Security, Software Security Engineering, Software Quality Assurance, Software Assurance Analytics, and Software Assurance Capstone Project (Each syllabus contains a course description, prerequisite knowledge, a list of learning objectives/topics, sources for the course, course delivery features, and course assessment features.)
Volume III: Master of Software Assurance Course Syllabi	Provides the syllabi for nine graduate SwA courses: Assurance Management, System Operational Assurance, Assured Software Analytics, Assured Software Development 1, Assured Software Development 2, Assured Software Development 3, Assurance Assessment, System Security Assurance, and Software Assurance Capstone Experience (The syllabi are organized similar to those in Volume II but also include a schedule of weekly in-class activities, suggested readings, and out-of-class assignments.)
Volume IV: Community College Education	Provides the syllabi for six SwA courses appropriate for community college students: Computer Science I, II, and III; Introduction to Computer Security; Secure Coding; and Introduction to Assured Software Engineering

Outcome 1. Assurance Across Lifecycles: Graduates will be able to incorporate assurance technologies and methods into lifecycle processes and development models for new or evolutionary system development, and for system or service acquisition.

Outcome 2. Risk Management: Graduates will be able to perform risk analysis, tradeoff assessment, and prioritization of security measures.

Outcome 3. Assurance Assessment: Graduates will be able to analyze and validate the effectiveness of assurance operations and create auditable evidence of security measures.

Outcome 4. Assurance Management: Graduates will be able to make a business case for software assurance, lead assurance efforts, understand standards, comply with regulations, plan for business continuity, and keep current in security technologies.

Outcome 5. System Security Assurance: Graduates will be able to incorporate effective security technologies and methods into new and existing systems.

Outcome 6. System Functionality Assurance: Graduates will be able to verify new and existing software system functionality for conformance to requirements and the absence of malicious content.

Outcome 7. System Operational Assurance: Graduates will be able to monitor and assess system operational security and respond to new threats.

Ultimately, the Software Assurance Curriculum Project developed the set of four volumes described in Table 1 [Mead 2010a, Mead 2010b, Mead 2011a, Mead 2011b].

Initial Transition Activities

It was clear to us from the outset that a comprehensive plan for promoting the transition and adoption of the curriculum would be needed. Introducing a single new elective course is a relatively easy undertaking. However, introducing a track is ambitious, and contemplating a whole new degree program can be a daunting task. The barriers can range from a lack of interested students in a particular geographic area, to a lack of qualified faculty, to a lack of administrative support. We therefore put a transition plan in place before the curriculum was published and executed. The activities included the following:

› **Publicity**: We prepared an announcement that was broadcast via email to SEI subscribers and posted on the DHS and SEI websites. We also developed a press release that went out to a number of educational publications, professional societies such as ACM and IEEE, and ACM and IEEE publications. We developed a flyer that was distributed by team members and their colleagues when they attend conferences.
› **Discussion group**: We established a LinkedIn discussion group that now has nearly 600 members.
› **Awareness:** We also conducted an awareness-raising workshop at the Conference on Software Engineering Education and Training (CSEET) in 2010 and videotaped it. We also recorded several webinars and podcasts to provide an overview of the work.
› **Mentoring**: Initially the curriculum development team provided free mentoring to universities or faculty members who wished to offer a course, track, or Master of Software Assurance (MSwA) degree program.
› **Publications**: We produced more than 20 papers and conference talks, including keynote presentations.
› **Professional society recognition**: We received official recognition of the curriculum from the ACM and the IEEE Computer Society.

As a consequence of our initial outreach activities, a number of universities and training organizations adapted various aspects of the curriculum work. Courses and tracks based on the curriculum recommendations were developed and offered by Carnegie Mellon University, Stevens Institute of Technology, The U.S. Air Force

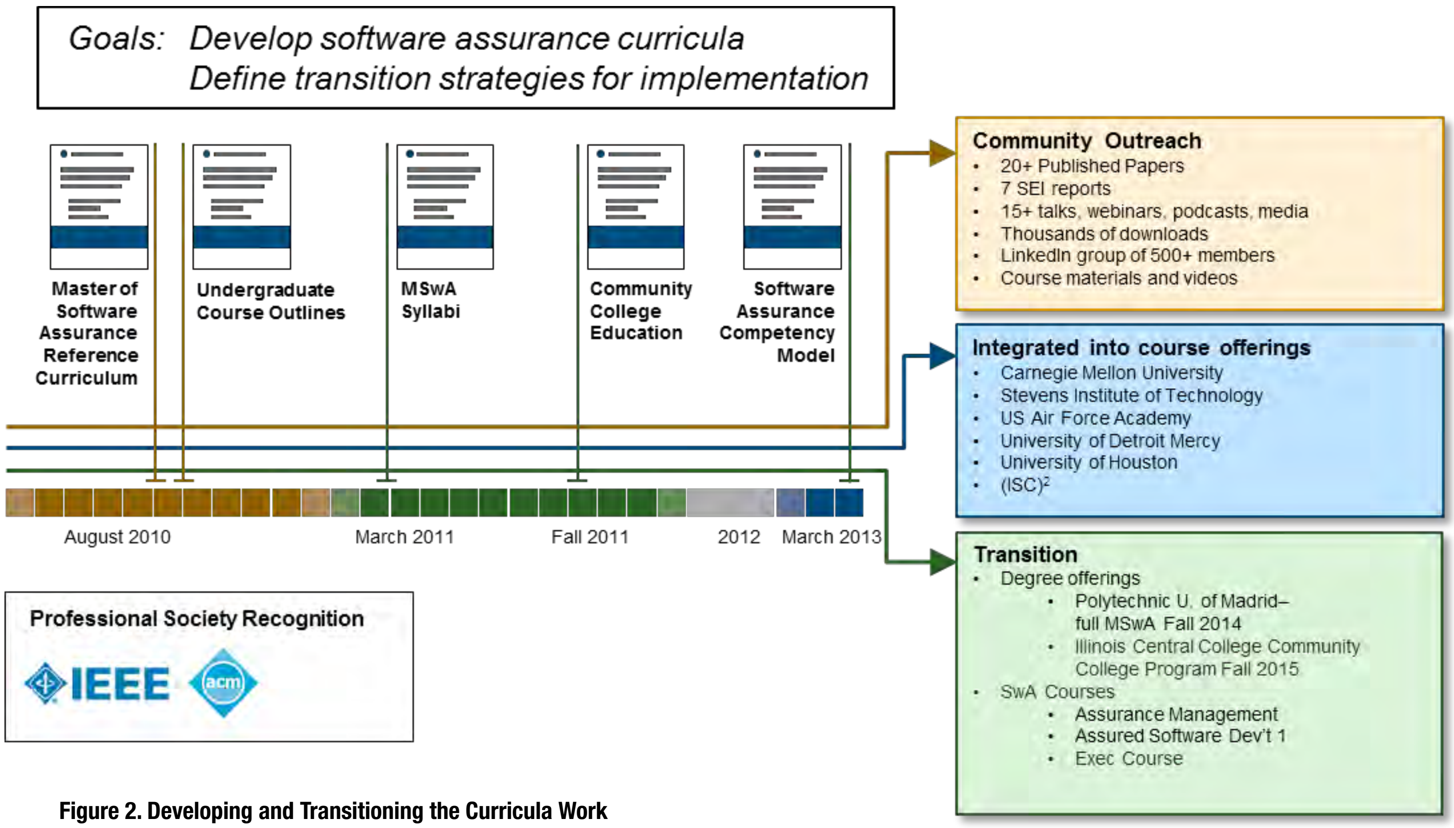

Goals: Develop software assurance curricula
Define transition strategies for implementation
Master of Software Assurance Reference Curriculum
Undergraduate Course Outlines
MSwA Syllabi
Community College Education
Software Assurance Competency Model
August 2010
March 2011
Fall 2011
2012
March 2013
Community Outreach
20+ Published Papers
7 SEI reports
15+ talks, webinars, podcasts, media
Thousands of downloads
LinkedIn group of 500+ members
Course materials and videos
Integrated into course offerings
Carnegie Mellon University
Stevens Institute of Technology
US Air Force Academy
University of Detroit Mercy
University of Houston
(ISC)²
Transition
Degree offerings
Polytechnic U. of Madrid– full MSwA Fall 2014
Illinois Central College Community College Program Fall 2015
SwA Courses
Assurance Management
Assured Software Dev't 1
Exec Course
Professional Society Recognition
IEEE
acm

Figure 2. Developing and Transitioning the Curricula Work

Academy, University of Detroit Mercy, University of Houston, and the International Information System Security Certification Consortium (ISC)². In addition, Polytechnic University of Madrid designed a Master of Software Assurance degree program.

The SEI developed three courses based on the initial curriculum recommendations. These included an Executive Overview course, and from the MSwA Curriculum, academic course materials for Assurance Management and Assured Software Development 1. These courses are available for free download from the SEI website.

The SwA curriculum work influenced other curriculum activities. For example, the Securely Provision area of the National Institute of Standards and Technology (NIST) NICE curriculum draws on the SwA curriculum work. More recently, the draft Cyber Security Curricula 2017 (CSEC) reflects aspects of the SwA curriculum work, particularly in the software security knowledge area.

In addition, a collaborative effort led to a successful community college degree program in secure software development. We also modified and transitioned our Assured Software Development 1 course materials to SPAWAR SD for in-house use in their own training programs. These are discussed in subsequent sections of this article. A timeline for the curriculum work and its transition is shown in Figure 2.

a growing awareness that the U.S. could reap substantial benefits from this model

SwA Community College Curriculum Recommendations

In Volume IV [Mead 2011b], after studying related degree programs, we introduced a suite of six courses that could form part of a two-year degree program in software assurance. The first three courses modified existing courses from the ACM Committee for Computing Education in Community Colleges (CCECC) to add a security emphasis. The other three courses are more specialized. In the report, we included prerequisites, syllabi, sources, and Bloom's taxonomy levels for each course.

The following is a list of the course names:

- Computer Science I (modified from standard curricula)
- Computer Science II (modified from standard curricula)
- Computer Science III (modified from standard curricula)
- Introduction to Computer Security (new course)

- Secure Coding (new course)
- Introduction to Assured Software Engineering (new course)

Subsequently, the project produced the Software Assurance (SwA) Competency Model [Hilburn 2013]. Two of the objectives of this model are listed below:

- Enhance SwA curricula guidance by providing information about industry needs and expectations for competent SwA professionals.
- Provide direction and a progression for the development and career planning of SwA professionals.

From the viewpoint of the curriculum project, the four curriculum documents and the competency model set the stage for transitioning the work to educational institutions that wished to offer software assurance concentrations or full-degree programs. Next, we discuss the ways we tried to meet these two objectives in a unique community college program.

The Illinois Central College Program

In September 2013, industry, government, and academic stakeholders met in Peoria, Illinois and proposed an initiative to create software developer jobs and make the Peoria area a national center of excellence for producing software that is secure from cyber attacks. The German apprenticeship model was proposed to create a skilled workforce that is trained, apprenticed, mentored, and certified in secure software production. (There is a growing awareness that the U.S. could reap substantial benefits from this model.)

Apprenticeships allow businesses to meet the growing demand for skilled workers and lead workers to higher wages and better employment outcomes. Furthermore, apprenticeships are a smart public investment. A recent study in Washington State found that for every $1 in state investment in apprenticeships, taxpayers received $23 in net benefits—a return that far exceeds that of any other workforce-training program in the state [State of Washington 2013, Olinsky 2013].

The initiative partnered with the school districts to encourage graduating high school seniors to pursue software development careers in the Peoria area. Ultimately the Central Illinois Center of Excellence for Secure Software (CICESS) was formed to collaborate on the community college/industry apprenticeship program.

The SEI collaborated with the CICESS and Illinois Central College (ICC) to develop a two-year degree program in Secure Software Development, incorporating an apprenticeship model. Part of the reason we focused on community college education (in addition to four-year undergraduate degree programs and master's degree programs) is that, according to the American Association for Community Colleges, roughly half of U.S. undergraduate students have attended community college [Mead 2010b].

ICC in East Peoria, IL is a comprehensive community college in the Illinois Community College system. Approximately 10,500 students are enrolled in 58 applied degrees, 72 certificates, and over 50 areas of study in associate of arts and associate of science degrees for transfer. ICC has a close working relationship with many local employers in central Illinois, particularly in the applied sciences.

In the information systems programs, these partnerships are usually in the form of student internships and work-study opportunities at the college. Apprenticeship programs with employers involved in the CICESS had not been considered in prior years. ICC faculty presented the option as part of their Applied Science degree, in which students would take approximately 42 credit hours of technical computer science and database courses and only 18 credit hours in general education. ICC had an existing Associate in Applied Science (AAS) degree in Computer Science and Database Development that seemed to more closely fit employer needs. The goal of the CICESS was to provide apprenticeships in secure software development; however, the new curriculum needed to include computer security and software assurance concepts.

ultimate goal was to develop and acquire software that was better able to resist cyber attacks.

This is the point at which ICC faculty members began integrating the SEI Software Assurance Curriculum with their own. The SwA curriculum recommendations for community colleges [Mead 2011b] consisted of the six courses mentioned earlier. ICC faculty consulted with employers to determine which SwA courses were needed in addition to the SEI recommended courses. Employers felt that students needed a good foundation in SQL, C#, and Mobile Applications in addition to programming and security courses.

The new AAS degree in Secure Software Development consists of the following program requirements. Courses in bold were modified or added as part of the new program, in collaboration with the SEI.

- **CS I: Programming in Java**
- **CS II: Programming in Java**
- **CS III: Advanced Programming in Java**
- Structured Query Language
- Introduction to Relational Database
- C# Programming
- Mobile Application Programming
- **Introduction to Computer Security**
- **Secure Coding**

- **Introduction to Assured Software Engineering**
- Database Administration
- Structured System Analysis
- two electives in computer programming, web, or networking, depending on employer needs
- 19 credit hours in general education courses

Developed courses were offered in a traditional 16-week semester in 8-week courses and in an online format. Students who wished to be eligible for the CICESS apprenticeship program took the courses in accelerated 8-week sessions. In addition, employers wanted to be assured that the student apprentices had an aptitude for computer programming. Therefore, students who wanted to be considered for apprenticeship had to take a commercial computer programming aptitude test, the Berger Aptitude for Programming Testing [B-APT], and achieve a minimum score of 20. The B-APT assesses the student's ability to do computer programming: "Organizations use the B-APT primarily to identify high aptitude candidates for programmer training. The examinee need have no prior experience in programming, and those with some experience gain no advantage over the inexperienced. The tutorial, which uses a hypothetical language, equates the potential of the inexperienced with the experienced."

ICC implemented and launched the AAS degree in Secure Software Development in the Fall 2015 semester with over 20 students in the program. In Fall 2016, the number of incoming students more than doubled and some of the students in the first cohort started apprenticeships with industry partners.

Collaboration Between the SEI and SPAWAR SD

SPAWAR in San Diego contacted the SEI to discuss their interest in in-house training. Their ultimate goal was to develop and acquire software that was better able to resist cyber attacks. After several conversations and meetings, and a review of the SwA curriculum work, SPAWAR SD concluded that their needs could best be served by modifying and delivering the existing Assured Software Development I course. This course delivered the fundamentals of incorporating assurance practices, methods, and technologies into software development and acquisition lifecycle processes and models, and provided rigorous methods for software assurance requirements engineering in support of development and acquisition; using threat identification, characterization, and modeling; performing assurance risk assessment; and evaluating misuse/abuse cases.

The materials that were intended to support a one-semester academic course would be modified and compressed into a two-week workshop offering. Support for SPAWAR sponsorship of this activity was obtained, and the work was executed over a six-month period, culminating in a workshop offered at SPAWAR SD in August 2016. The attendees were technical leaders and in-house instructors at SPAWAR SD, and the full set of workshop materials was provided for their internal use in training.

After joint review of the materials, it was decided that some of the theoretical research topics needed for an academic audience would not be useful to SPAWAR practitioners, so these were replaced with SEI materials intended for immediate use. In addition, videos from the SEI's online courses were provided as part of the package for SPAWAR staff to use as collateral material.

Class participants connected to all aspects of the SPAWAR acquisition and development lifecycle, including development, project management, quality control, enterprise and software assurance, supply chain coordination, and testing. This broad base provided an opportunity for class discussions to cover all aspects of current software assurance and security practices to identify key opportunities for improvement in applying the course lessons. Class content was composed of a mix of lectures, selected videos, case studies, and discussion.

The results for SPAWAR were immediate:

- Class participants identified 10 immediate actions that they could take to improve existing practices for SwA.
- Class discussions generated five pages of ideas for additional SwA improvements.
- Partnerships among participating disciplines were established with plans for a more integrated approach.
- Analysis of available evidence provided a prioritized list of where SPAWAR needed to focus immediate attention.
- SPAWAR management, in their review of the project, confirmed the success of the engagement as excellent in timeliness, quality, and value.

Summary and Future Plans

We completed the development and publication of the textbook, *Cyber Security Engineering: A Practical Approach for System and Software Assurance*, which was released November 2, 2016 as part of the SEI Book Series. For more information about the book, see **https://insights.sei.cmu.edu/sei_blog/2016/10/seven-principles-for-software-assurance.html**. Work is also underway for an online certificate in cybersecurity engineering to augment available resources.

Though we demonstrated strong success with the curriculum materials developed so far, the model cannot reach its full potential until we have full-course content (e.g., slides, instructor notes, homework, exams, and case studies) developed for all courses. Seven of the MSwA courses are still in need of material development.

In addition, courses for related disciplines where software assurance is an elective, such as software engineering, computer science, and information systems, are in need of materials. Undergraduate courses, particularly for use with specializations in software engineering, information systems, and computer science, are also lacking in materials for broad use. Opportunities for inclusion in high school instruction remain unexplored and students are learning how to write code and field programs even earlier in their education without the benefit of knowing how to do so securely. ■

Acknowledgments

We would like to acknowledge our colleagues, collaborators, and sponsors who contributed to the Software Assurance Curriculum Project and its transition to the community.

REFERENCES

[1] [Allen 2008] Allen, Julia H.; Barnum, Sean; Ellison, Robert J.; McGraw, Gary; & Mead, Nancy R. Software Security Engineering: A Guide for Project Managers. Addison-Wesley Professional, 2008.

[2] [B-APT] Psychometrics, Berger Aptitude for Programming Test (B-APT) website. Available at http://www.psy-test.com/Baptd.html

[3] [CNSS 2009] Committee on National Security Systems. “Instruction No. 4009,” *National Information Assurance Glossary*. Revised June 2009.

[4] [Drew 2009] Drew, C. “Wanted: ‘Cyber Ninjas.’” *New York Times*, 2009. Retrieved December 29, 2009 from website, available at http://www.nytimes.com/2010/01/03/education/edlife/03cybersecurity.html?emc=eta1.

[5] [Mead 2010a] Mead, N. R., et al. *Software Assurance Curriculum Project Volume I: Master of Software Assurance Reference Curriculum* (CMU/SEI-2010-TR-005). Software Engineering Institute, Carnegie Mellon University, 2010. Available at http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=9415.

[6] [Mead 2010b] Mead, N. R., et al. S*oftware Assurance Curriculum Project Volume II: Undergraduate Course Outlines* (CMU/SEI-2010-TR-019). Software Engineering Institute, Carnegie Mellon University, 2010. Website, Available at http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=9543.

[7] [Mead 2011a] Mead, N.R. et al. *Software Assurance Curriculum Project Volume III: Master of Software Assurance Course Syllabi*, (CMU/SEI-2011-TR-013), Software Engineering Institute, Carnegie Mellon University, March 2011. Available at http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=9981.

[8] [Mead 2011b] Mead, N.R. et al. *Software Assurance Curriculum Project Volume IV: Community College Education*, (CMU/SEI-2011-TR-017), Software Engineering Institute, Carnegie Mellon University, September 2011. Available at http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=10009.

[9] [Olinsky 2013] Olinsky, B. and Steinberg, S. “Training for Success - A Policy to Expand Apprenticeships in the United States,” November 2013, Center for American Progress. Available at https://www.americanprogress.org/issues/economy/reports/2013/12/02/79991/training-for-success-a-policy-to-expand-apprenticeships-in-the-united-states/.

[10] [PPP 2009] Partnership for Public Service & Booz Allen Hamilton. *Cyber IN-Security: Strengthening the Federal Cybersecurity Workforce*. Partnership for Public Service, 2009. Retrieved July, 2009. Available at http://ourpublicservice.org/OPS/publications/viewcontentdetails.php?id=135.

[11] [State of Washington 2013] State of Washington Workforce Training and Education Coordinating Board, “2013 Workforce Training Results by Program: Apprenticeship.” Available at http://www.wtb.wa.gov/Documents/2_Apprenticeship_2013.pdf.

Cyber Security and Information Systems
Information Analysis Center
266 Genesee Street
Utica, NY 13502

PRSRT STD
U.S. Postage
P A I D
Permit #566
UTICA, NY

THE CENTER OF EXCELLENCE IN CYBER SECURITY AND INFORMATION SYSTEMS

Leveraging the best practices and expertise from government, industry, and academia in order to solve your scientific and technical problems

HTTPS://WWW.CSIAC.ORG/JOURNAL/

*To unsubscribe from CSIAC Journal Mailings please email us at **info@csiac.org** and request that your address be removed from our distribution mailing database.*