Software and Cyber Solutions Symposium 2018

How Should We Address Cybersecurity Risk in an Agile or DevOps Environment?

Carol Woody, Ph.D.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Document Markings

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM18-0347

Tutorial: *Topics*

Cybersecurity Risk

Agile and DevOps Impacts

Cybersecurity Risk Frameworks

- Build Security In Maturity Model (BSIMM)
- Standards
- NIST Risk Management Framework (RMF)
- Software Assurance Framework (SAF)

Applying Critical Cybersecurity Requirements

Case Studies & Summary

Cybersecurity Risk

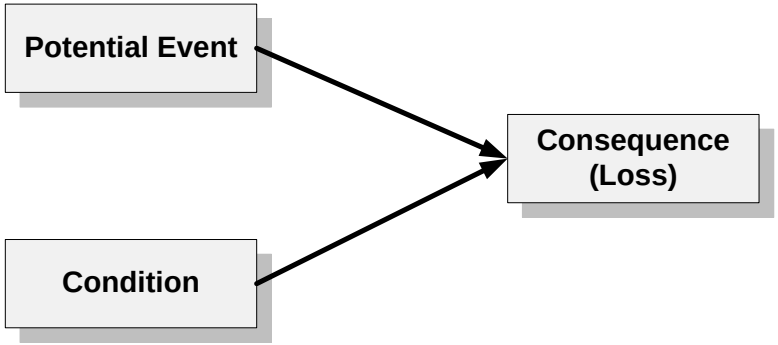
What Is Risk?

The probability of suffering harm or loss

A measure of the likelihood that an event will lead to a loss coupled with the magnitude of the loss

Risk requires the following conditions:[1]

- A potential loss
- Likelihood
- Choice



1. Charette, Robert N. *Application Strategies for Risk Analysis*. New York, NY: McGraw-Hill Book Company, 1990.

Software is Addressing More Functionality

All software has defects.

Software quality determines the defect volume and 1-5% are vulnerabilities

Software volume is increasing

- F-22 fighter aircraft (2005)
  - 1.7 MLOC
- F-35 Lightning II fighter aircraft (2016)
  - 24 MLOC

Likelihood of software vulnerabilities is increasing as software volume increases

Defects per million lines of code (MLOC)

Best-in-class code:
<600 defects per MLOC

Very good code:
600 to 1,000 defects per MLOC

Average quality code:
6000 defects per MLOC

Capers Jones, sqgne.org/presentations /2011-12/Jones-Sep-2011.pdf

1-5 % of defects are vulnerabilities.

Woody, Carol; Ellison, Robert; and Nichols, William. Predicting Software Assurance Using Quality and Reliability Measures. CMU/SEI-2014-TN-026. Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=428589)

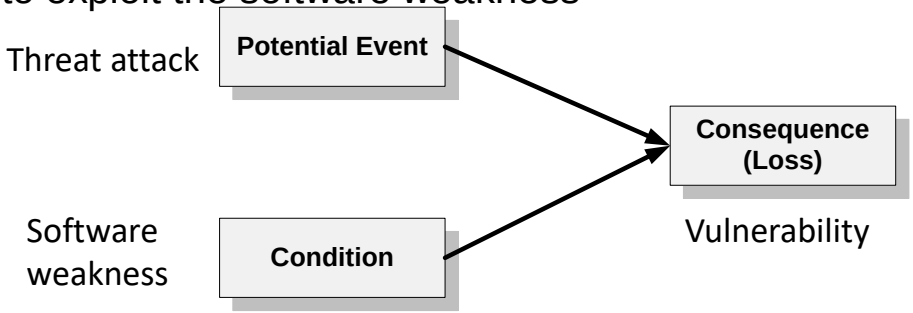
Software Vulnerability

Software Weakness: A deficiency, flaw, defect, or limitation in code, design, or architecture that can lead to a software vulnerability.

Software Vulnerability: A weakness in software that may be exploited, resulting in a negative impact to confidentiality, integrity, or availability.

Each exploit requires three elements

- software weakness
- threat source access to the software weakness
- threat source capability to exploit the software weakness



Software Vulnerabilities are Increasing

As the use of software increases, the three required elements are increasingly present:

System weaknesses

- Millions of lines of software code handling an ever increasing amount of system functionality
- Thousands of reported software vulnerabilities (see National Vulnerability Database)

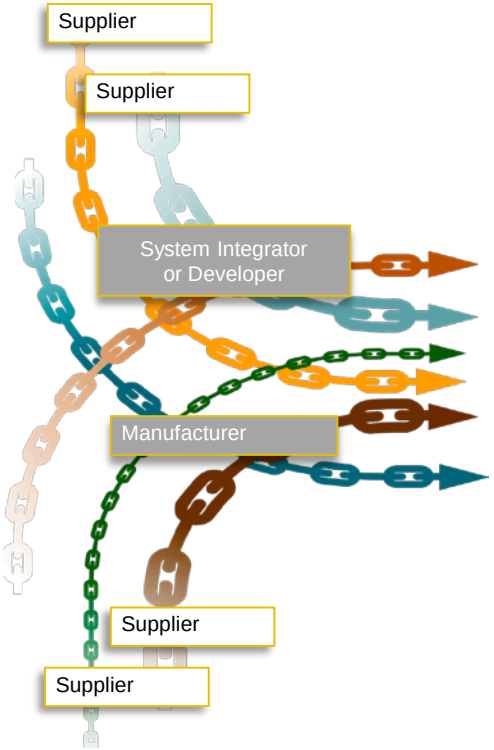
Threat source access to software weaknesses

- Many vendors have been compromised and their IP (e.g. designs and source code) stolen to inform attackers
- Increased connectivity linking systems to other systems and connecting systems to new types of devices expands the potential attack surface
- Increased system and device remote communication capability

Threat source capability to exploit the weakness

- Attacker access to the same tools and techniques used to build and defend software
- Reverse engineering capabilities for software patches provides information about the most recently identified vulnerabilities

Growing Supply Chains Increase Sources for Software Vulnerabilities and Threat Access



Intentional threats

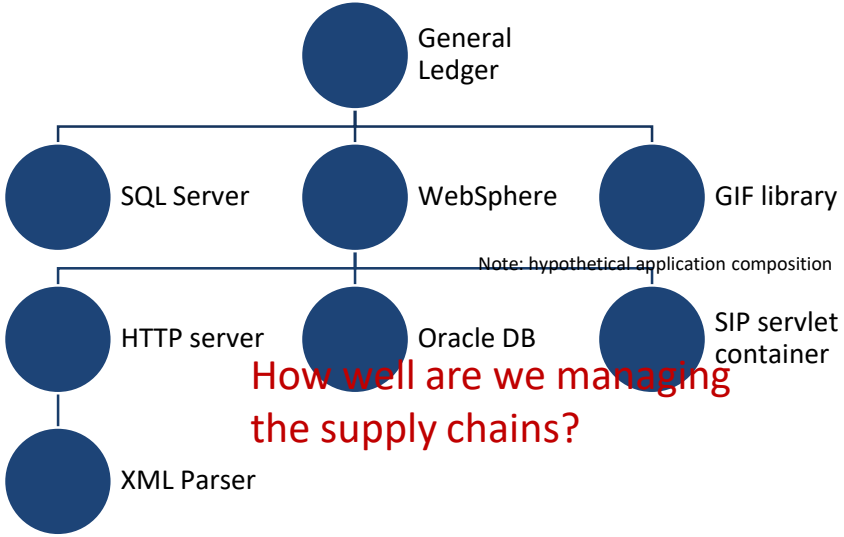
- counterfeit hardware and software
- tampering
- theft
- malware insertion

Unintentional threats

- poor code quality
- software vulnerabilities unintentionally inserted

Result: Systems that contain software vulnerabilities and do not “function as intended”

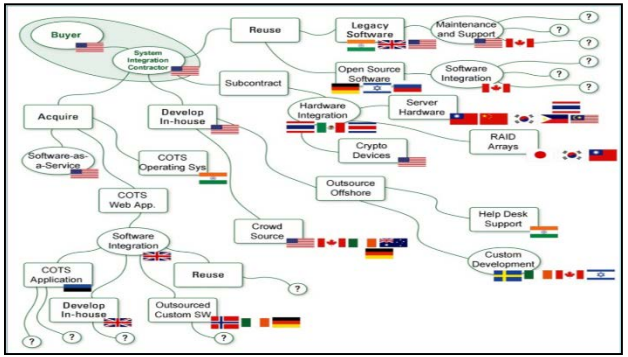
Contractors Perform Supply Chain Risk Management (SCRM)



Development is now assembly using collective development

- Too large for single organization
- Too much specialization
- Too little value in individual components

Supply chains are long, convoluted and international



Recent Supply Chain Attacks Indicate Growing Risk

Types of supply chain attacks that leveraged compromised code:

Source Code Attacks

- Shadowpad (2017), Anti-Virus Code attack (2017)

Software Tool Attacks

- XcodeGhost (2015), Expensive Wall (2017), HackTask (2017)

Download Site Attacks

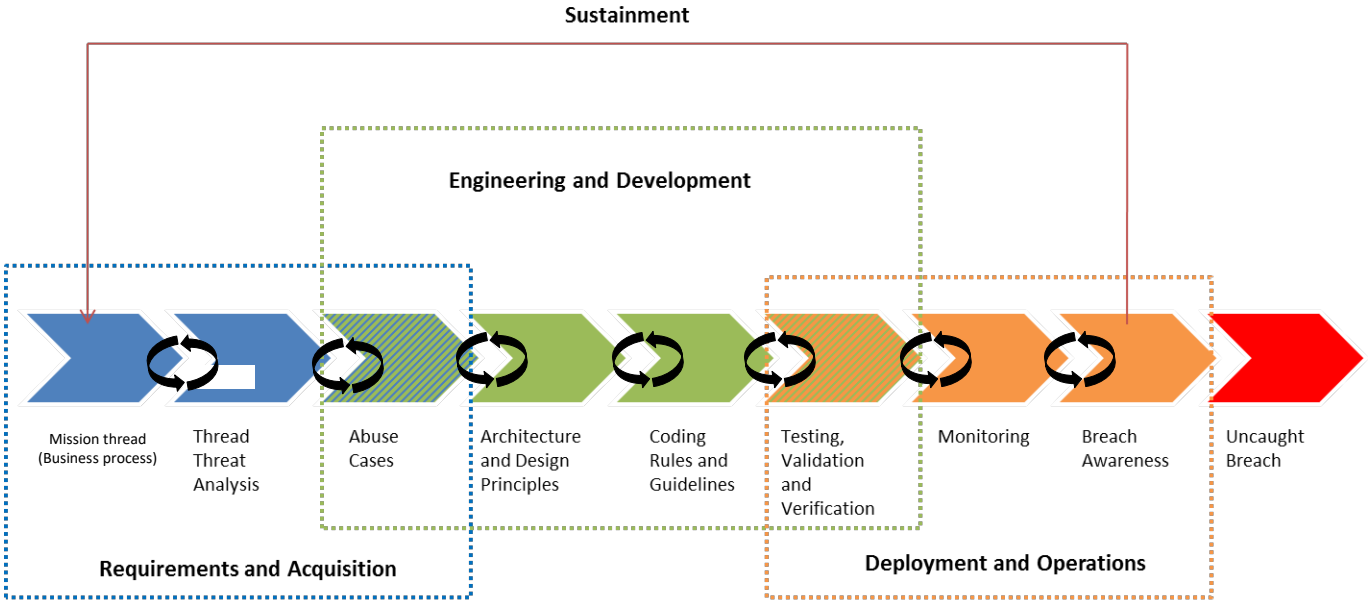
- Havex/Dragonfly (2014), KingSlayer (2015), Fioxif/CCleaner (2017)

Patch Site Attacks

- NotPetya/MeDoc (2017) paralyzed networks worldwide

Agile and DevOps Impacts

Opportunities to Reduce Cybersecurity Risk



Increased Software Use - Agile & DevOps Risk?

Extensive reliance on automation – even more software driving decisions

- What if the attacker could hide an attack by changing the monitoring software (integrity problem and undiscovered intrusions)

Stakeholders define the work sequence

- What if stakeholders choose to delay upgrades to finish some functionality first (potential vulnerabilities remain exposed)

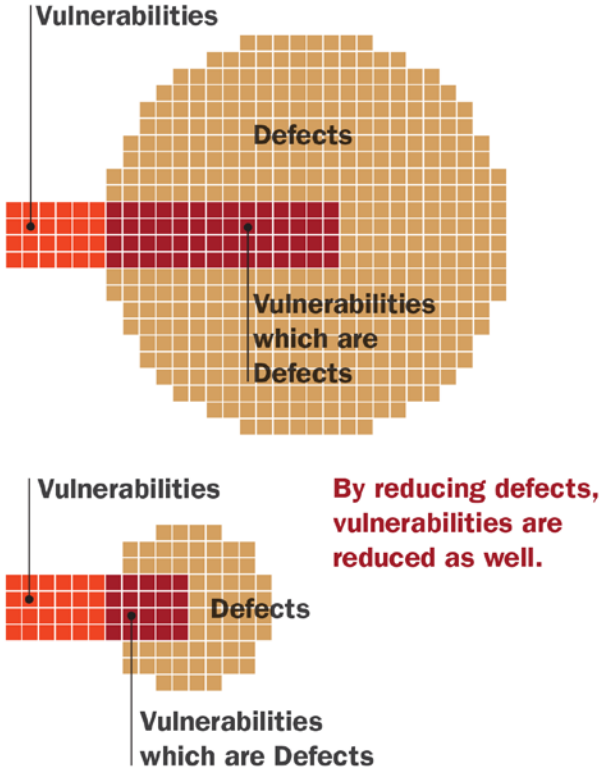
What if the integration of a COTS product provides functionality quickly but there are many vulnerabilities in the code?

- Who finds the vulnerabilities and fixes them?
- Who decides what is implemented and when?

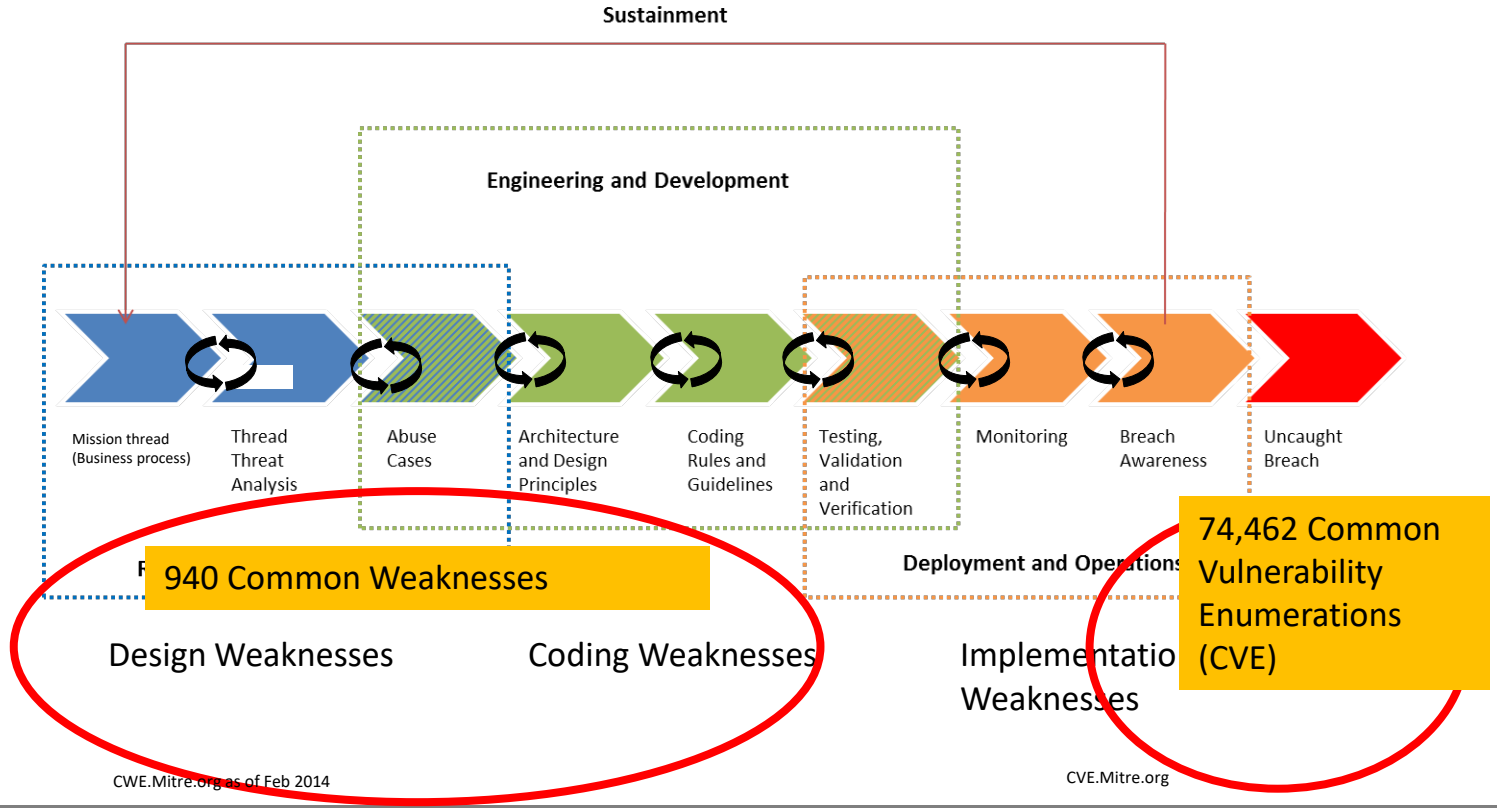
What if the use of Cloud services will speed up the implementation substantially in replacing a legacy system but the Cloud vendor will not demonstrate security testing logs and controls without extensive cost increases?

What Level of Quality Software Should We Build?

All software has defects. Research clearly shows the higher the quality of software, the lower the number of defects and the lower the number of vulnerabilities.



Cybersecurity Is a Lifecycle Challenge



Cybersecurity Risk Frameworks

Building Security In Maturity Model (BSIMM)

The Building Security In Maturity Model (BSIMM) is a multi-year study (8 years and counting) led by Cigital and Fortify (http://bsimm.com/)

- Reviewed the efforts of over 100 organizations
- Objective was to identify what is currently done (state of the practice) rather than to promote specific [unproven] practices.

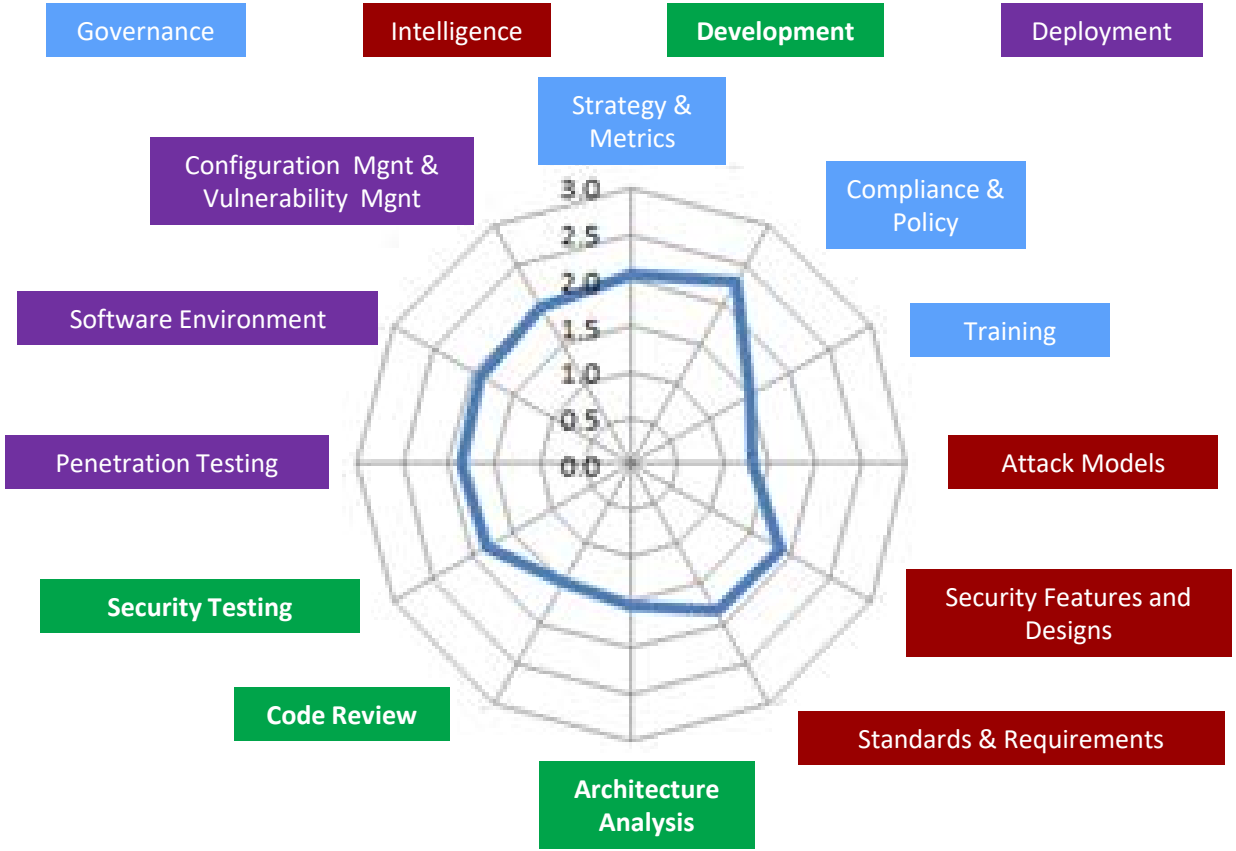
Identified 113 security activities across the surveyed organizations and organized those activities into 12 practice areas.

The number of collected activities demonstrates that there are multiple ways to improve software security and that the BSIMM activities should be used as specific acquisition requirements.

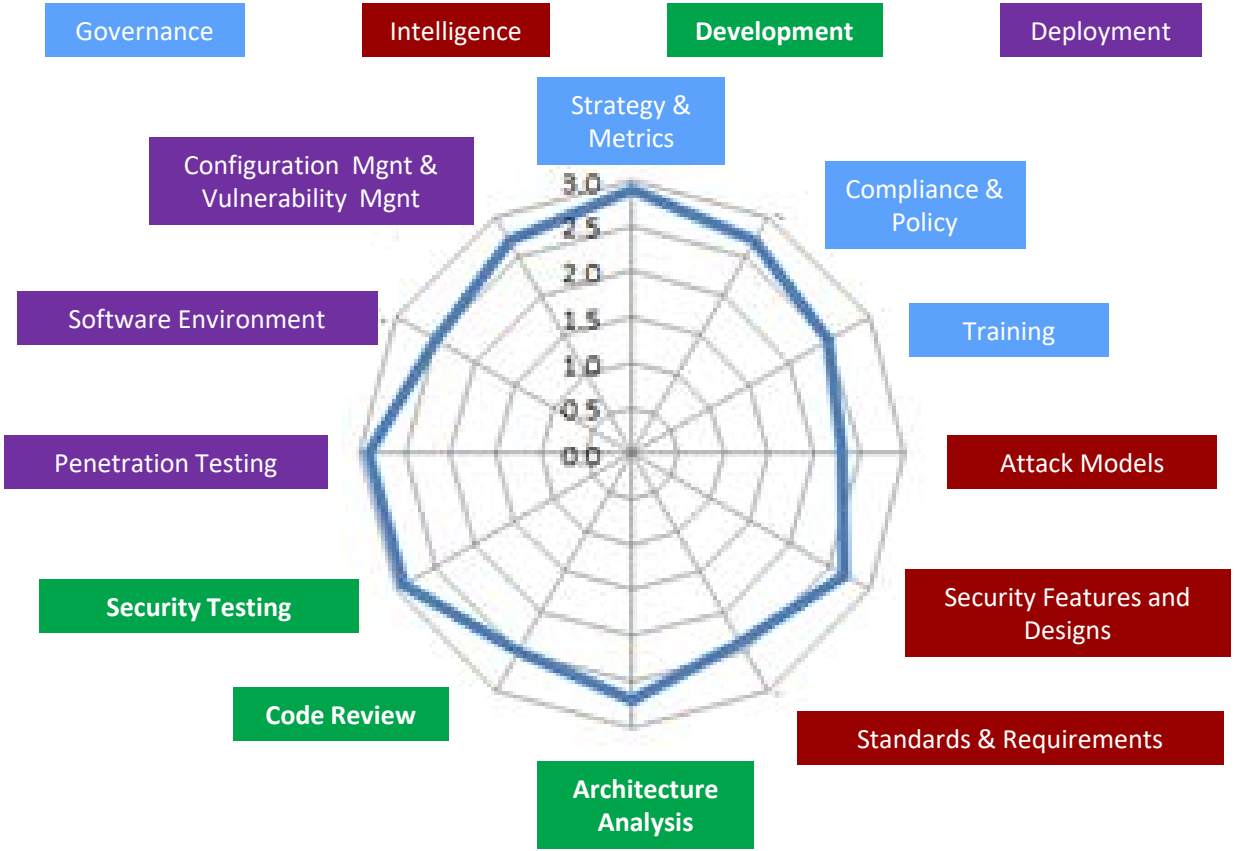
Defines three levels of levels of maturity for each practice based on usage.

- Level 1 activities (straightforward and simple) are commonly observed,
- Level 2 (more difficult and requiring more coordination) slightly less so
- Level 3 (rocket science) are much more rarely observed.

BSIMM Average Scores



BSIMM Scores for Top 10 Firms



Standards Organizations Focused on Cybersecurity

ISC/ESC – secure coding, supply chain

NIST – security risk management, secure practices, security systems engineering, supply chain risk management

IEEE – software engineering

NDIA – system engineering

ISACA – process management (recently acquired CMMI)

ISC^2 – certification for information assurance, compliance with SwA curriculum model

INCOSE – system engineering

OWASP - Open Web Application Security Project

Other Industry Standards Efforts

Object Management Group (http://www.omg.org/)

Software Assurance Working Group is developing specifications that enable creation of tools related to data collection for automation of assurance

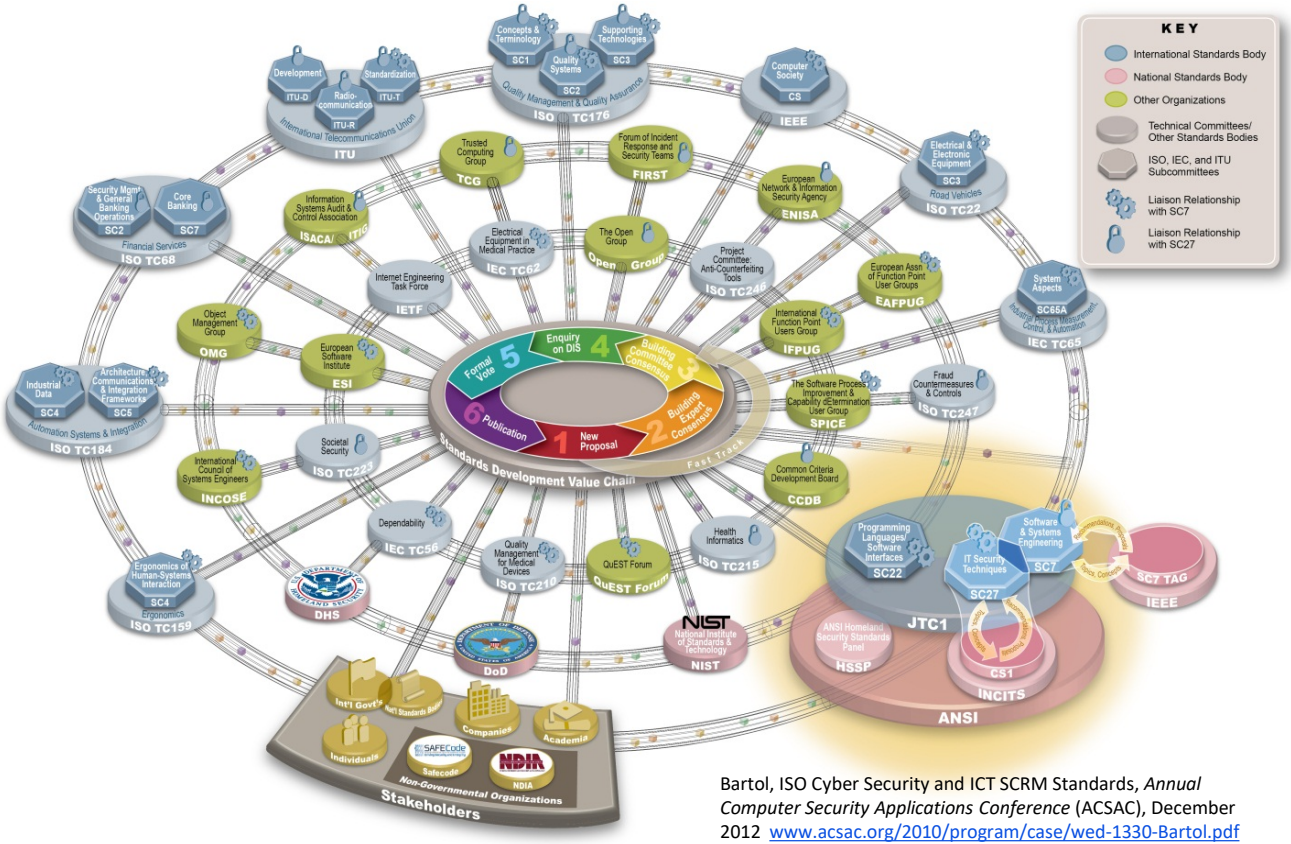
Trusted Technology Forum (http://www.opengroup.org/subjectareas/trusted-technology)

Standard that certify conformance to best practices of ICT Providers to mitigate the risk of tainted & counterfeit products.

SafeCode Industry Consortium (https://www.safecode.org/)

Major vendors (e.g. Microsoft, Dell, Seimans, Adobe, Symantec) who build critical technology products share information about how they address security.

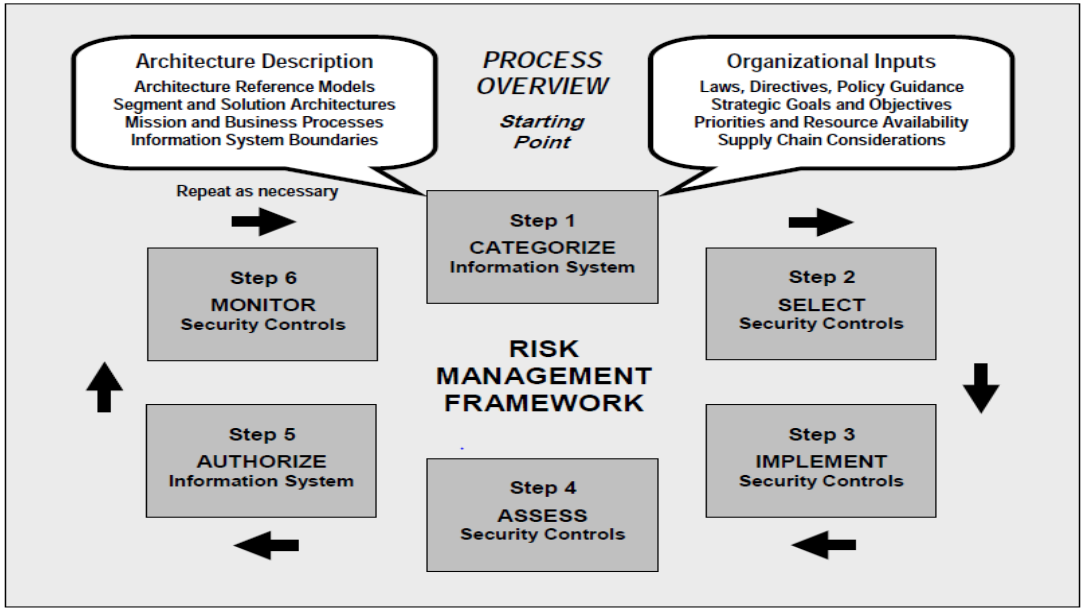
Standards Bodies



Bartol, ISO Cyber Security and ICT SCRM Standards, *Annual Computer Security Applications Conference* (ACSAC), December 2012 www.acsac.org/2010/program/case/wed-1330-Bartol.pdf

NIST Risk Management Framework (RMF)

NIST SP 800-37 (DoDI 8510.01) RMF



Security Practice Guidance: NIST 800-53, NIST 800-53A

Supply Chain Risk Guidance: NIST 800-161

Software Assurance Framework (SAF)

What

- Defines cybersecurity practices for acquiring and engineering software-reliant systems

Why

- Improve cybersecurity practices in acquisition programs

Benefits

- Provides the basis for assessing gaps in a program's cybersecurity practices and charting a course for improvement
- Establishes confidence in a program's ability to acquire software-reliant systems across the lifecycle and supply chain
- Reduces cybersecurity risk of deployed software-reliant systems

https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=496134

Key Practice Areas for Cyber Security

Process Management

- Process Definition
- Infrastructure Standards
- Resources
- Training

Engineering

- Product Risk Management
- Requirements
- Architecture
- Implementation
- Testing, Validation, and Verification
- Support Documentation and Tools
- Deployment

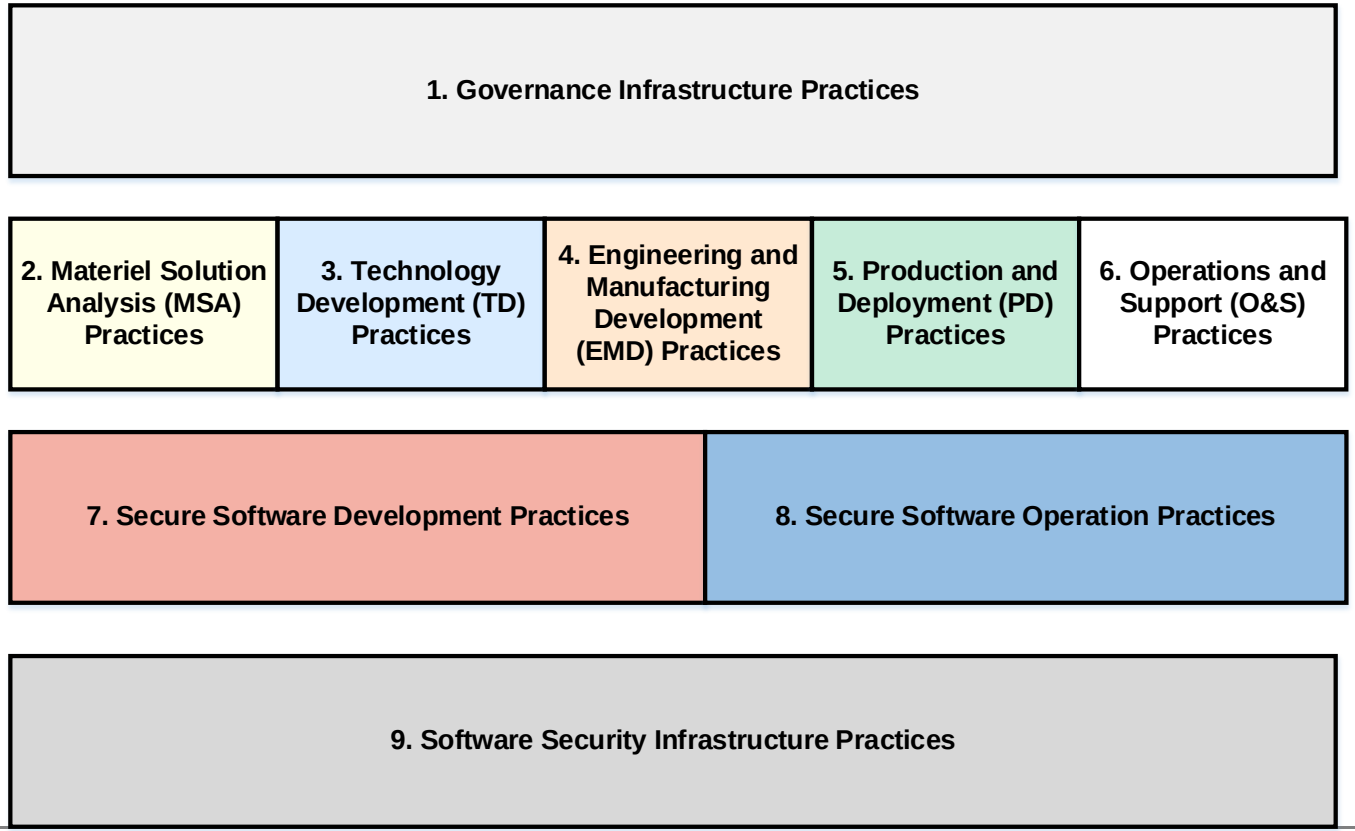
Project Management

- Project Plans
- Project Infrastructure
- Project Monitoring
- Project Risk Management
- Supplier Management

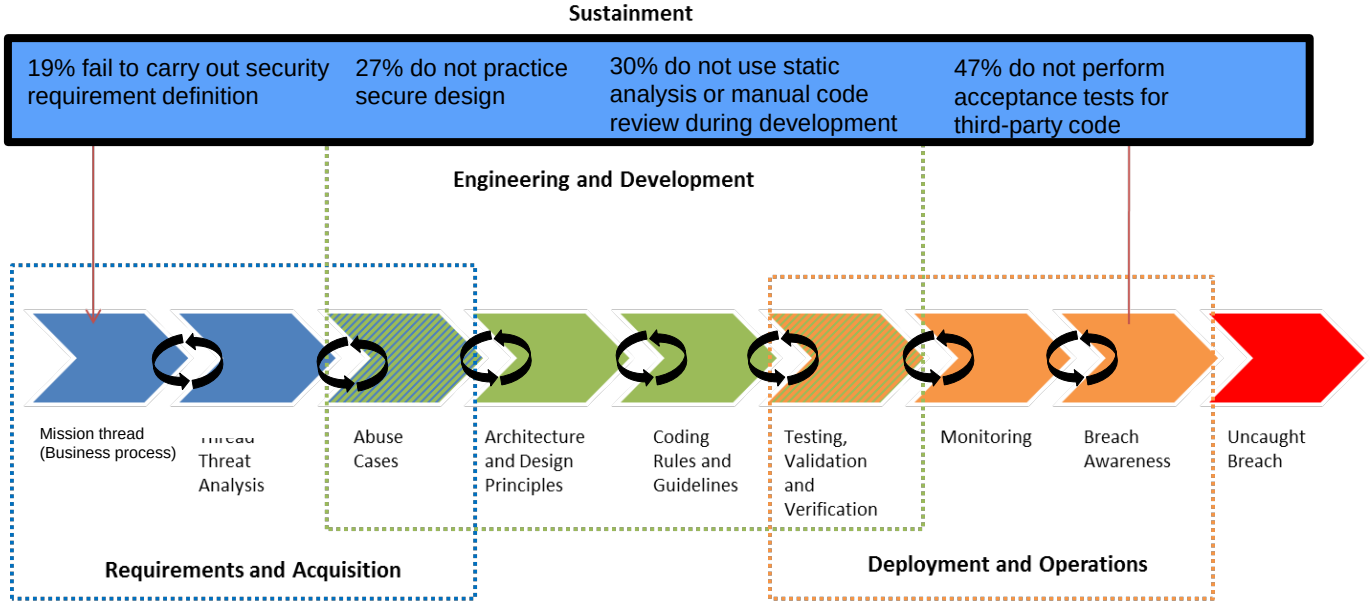
Support

- Measurement and Analysis
- Change Management
- Product Operation and Sustainment

SAF: *Policy, Process, & Practice Alignment*



Opportunities for Improvement

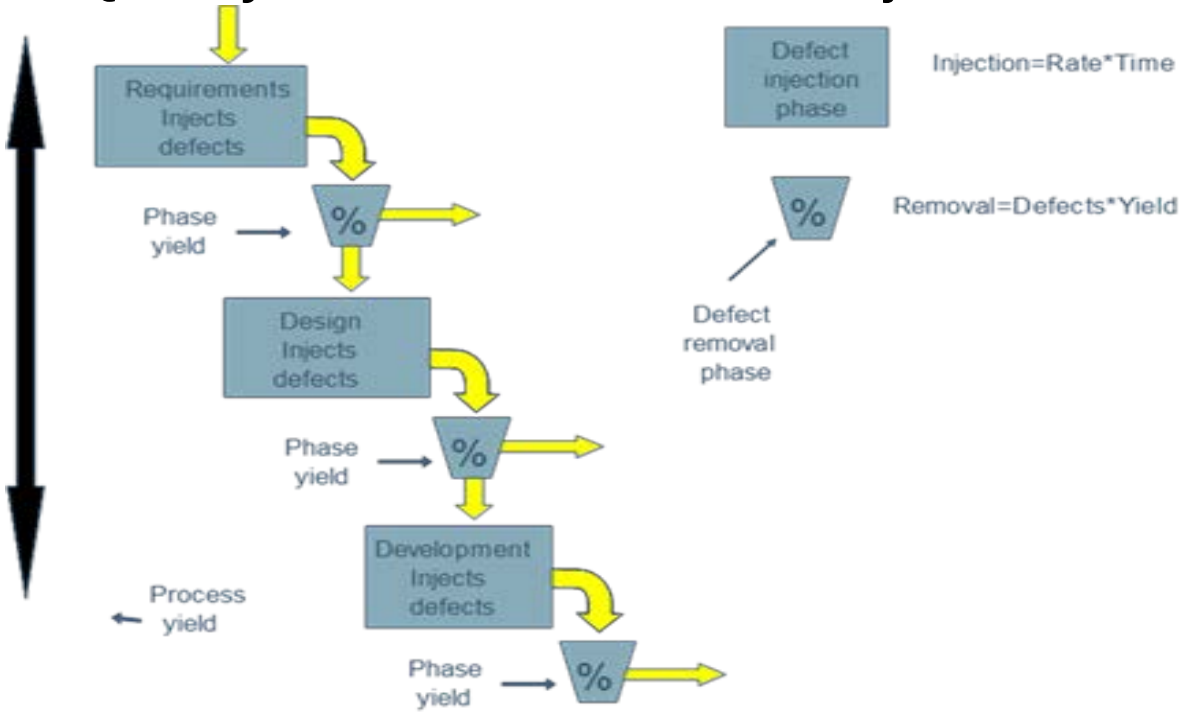


Less than 19% coordinate their security practices in various stages of the development lifecycle.

Source: Forrester Consulting, “State of Application Security,” January 2011

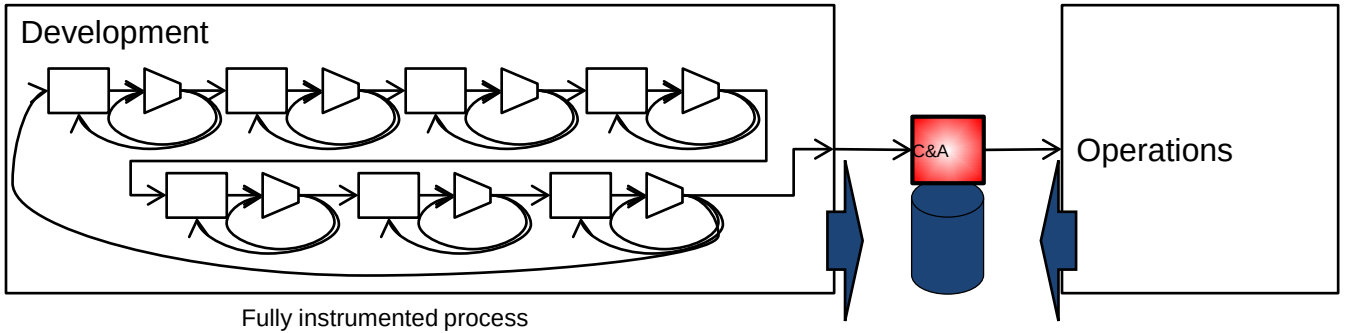
Applying Critical Cybersecurity Requirements

Emphasize Quality - Focus on Defect Injection and Removal



Poor quality predicts poor security

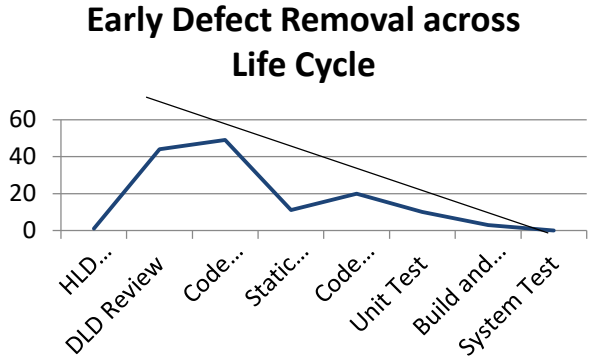
Approach for Predicting Security with Quality



Fully instrumented process

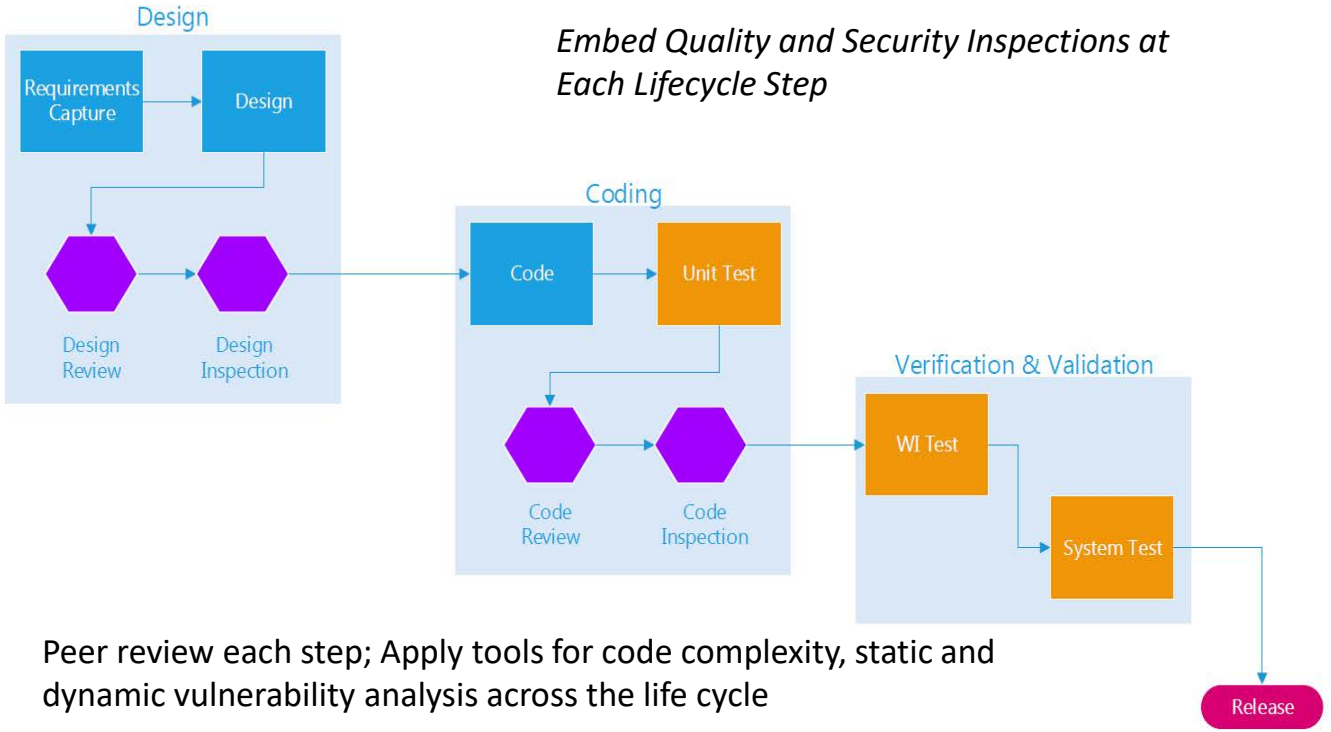
Early Life Cycle Indications for software assurance risk based on quality processes and quality control

- Fine-grained quality gates and tracking
- Models for predicting phase, increment, release, and operations quality



The Systems Sciences Institute at IBM has reported that "the cost to fix an error found after product release was four to five times as much as one uncovered during design, and up to 100 times more than one identified in the maintenance phase." [2015]

Successful Projects Inject Security Expertise Continuously



Embed Quality and Security Inspections at Each Lifecycle Step

Peer review each step; Apply tools for code complexity, static and dynamic vulnerability analysis across the life cycle

Plan for Effective Cybersecurity

Establish a Secure Software Development Lifecycle (SDLC) to ensure that cybersecurity activities are an integral part of the development effort to address:

- Software security as a continuous concern across the life cycle
- Early detection and removal of software weaknesses
- Awareness of the impact of software assurance choices on program protection

RMF Controls Addressed

SA-3 System Development Life Cycle

SA-4(3) Acquisition Process | Development Methods/Techniques/Practices

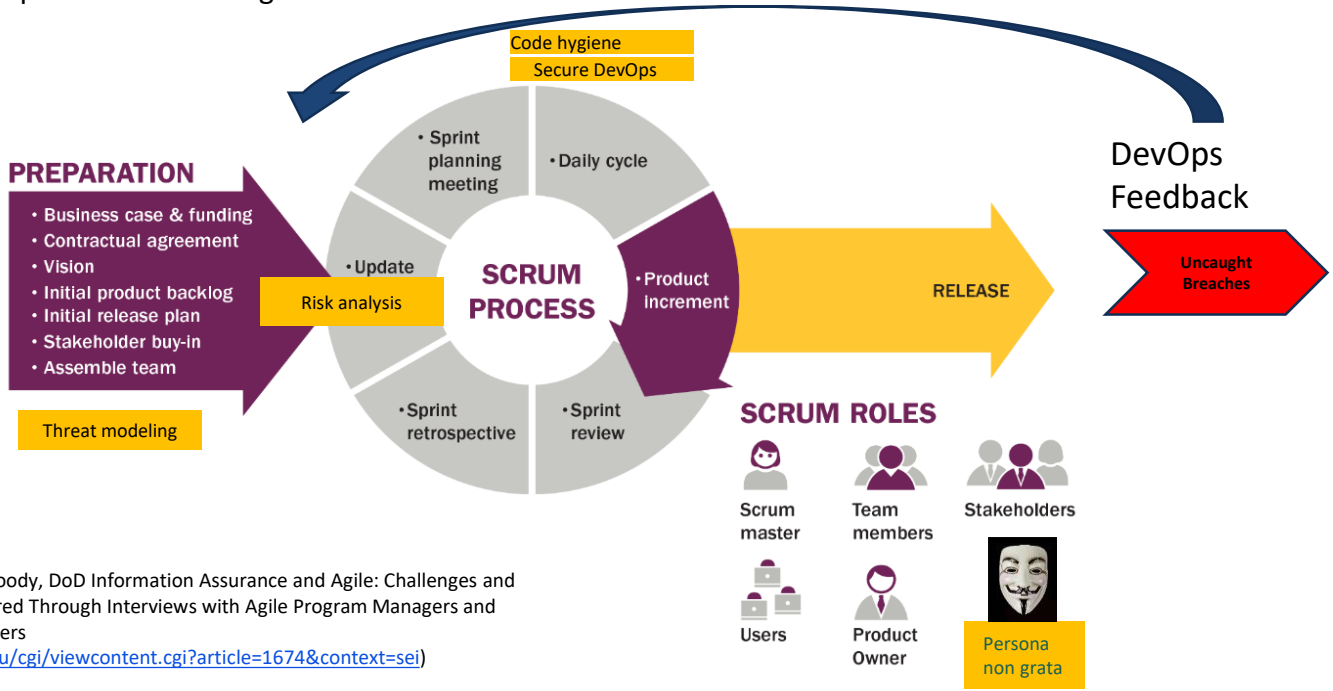
Secure Software Development Life Cycle

Key Considerations:

- Secure development practices must apply to all software (including firmware, embedded, safety-critical, and third-party)
- Processes and practices should be clearly identified as addressing cybersecurity and integrated into each delivery cycle
- Third party participation includes confirming they are following secure development practices or compensating for potential gaps by applying them as part of integration
- Incident response, breach notification, and data recovery should be part of the software life cycle practices (direct link to DevOps) with a feedback mechanism to remove the root causes

Secure Software Development Life Cycle

1. Code hygiene – introduce secure coding
2. Secure DevOps – include security tools
3. Threat modeling – represent a new role
4. Risk analysis – prioritize in backlog



(See also: Bellomo and Woody, DoD Information Assurance and Agile: Challenges and Recommendations Gathered Through Interviews with Agile Program Managers and DoD Accreditation Reviewers (http://repository.cmu.edu/cgi/viewcontent.cgi?article=1674&context=sei)

Focus on Software Vulnerabilities & Weaknesses

	Critical Software Cybersecurity Requirements
1	Secure System/Software Development Lifecycle
2	Software Development Process, Standards, and Tools
3	Software Security Requirements
4	Software Security Architecture and Design
5	Software Configuration Management
6	Developer Security Testing and Evaluation
7	Static Code Analysis
8	Dynamic Code Analysis
9	Manual Code Reviews
10	Attack Surface Reviews
11	Software Threat Analysis
12	Penetration Testing/Analysis
13	Verifying Scope of Testing and Evaluation
14	Independent Verification of Assessment Plans/Evidence
15	Software Flaw Remediation
16	Malicious Code Protection
17	Software and Firmware Integrity
18	Software Supply Chain Protection

For each software requirement we will address the following:

- Description/value
- RMF Controls
- Examples
- Key Considerations
- Evidence

2. Software Development Process, Standards, and Tools

Description

Repeated consistent delivery for of effective cybersecurity for all software development requires the use of processes, standards, and tools that address software security within the development life cycle for the each specific type of software to be delivered. Choices should be based on criticality of software, who must use them, options available for each specific software type and when they are used in the software development life cycle.

RMF Controls Addressed

SA-15 Development Process, Standards, and Tools

2. Software Development Process, Standards, and Tools

Examples

Processes for Security Requirements: Threat modeling, Security Engineering Risk Analysis (SERA), Software vulnerability assessments, Supply Chain Risk Management (SCRM)

Standards: NIST Cybersecurity Framework (CSF) focuses on critical infrastructure, ISO/IEC 27034 (2011) an international standard for application security which is life cycle agnostic, CERT Coding Standards (C, C++, Java), Open Trusted Provider Technology Standard (O-TTPS) that certifies conformance to best practices to mitigate the risk of tainted & counterfeit products

Tools: Integrated Development Environments (IDE) incorporate software vulnerability tracking into software development activities; static and dynamic analysis tools, code compliance checkers

2. Software Development Process, Standards, and Tools

Key Considerations:

Coverage of the many types of software is a major concern

- Will legacy code and supplier software be covered?
- Are all coding languages covered?
- Are multiple tools used?

Evidence:

Tools, standards, and processes should be clearly visible and monitored for effectiveness (e.g. # uncaught breaches and reasons for them should be reduced)

Developers tend to repeat their mistakes – how is their training improved by the identification and handling vulnerabilities and uncaught breaches

3. Software Security Requirements (User Stories for Misuse and Abuse)

Description

It is well recognized in industry that requirements engineering is critical to the success of any major development project. Several authoritative studies have shown that requirements engineering defects cost 10 to 200 times more to correct once fielded than if they were detected during requirements development.

For Agile development building starts with good user stories that consider not only desired functionality but ways in which the system should properly handle unacceptable behaviors (misuse and abuse).

RMF Controls Addressed

SA-4 Acquisition Process

SA-4(1) Acquisition Process | Functional Properties of Security Controls

3. Software Security Requirements

Examples

- Attack trees

  Ellison, R. & Moore, A. Trustworthy Refinement Through Intrusion-Aware Design (CMU/SEI-2003-TR-002, ADA414865). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.

- Use, Misuse, and Abuse Cases

  Alexander, Ian. “Misuse Cases: Use Cases with Hostile Intent.” IEEE Software 20, 1 (January-February 2003): 58-66.

- Security Quality Requirements Engineering (SQUARE)

  Mead, N.R. ; Hough, E.; & Stehney, T. Security Quality Requirements Engineering (SQUARE) Methodology (CMU/SEI-2005-TR-009). Software Engineering Institute, 2005. http://www.sei.cmu.edu/library/abstracts/reports/05tr009.cfm

- Security Patterns

  Mellardo, D., Fernandez-Medina, E., Paittini, M., Applying a Security Requirements Engineering Process. Computer Security – ESORICS 2006: 11th European Symposium on Research in computer Security, Hamburg, Germany, September 18-20, 2006. Proceedings (pp.192-206)

3. Software Security Requirements

Key Considerations:

Common software security requirements problems:

- stated as specific security solutions (practices) or compliance mandates and not real requirements which must be testable
- too narrowly focused on security in a particular component (e.g. use SSL for Web communication) and not the whole system
- external feeds from other systems are trusted without verification
- no stakeholders knowledgeable enough about security impacts to effectively state software security requirements

Evidence:

User stories clearly describe how the system should and should not perform

Test plans include verification that the system does not do what is not allowed

4. Software Security Architecture and Design

Description

Architecture and design establish how the system will function and choices are made among the various system qualities (performance, safety, reliability, security, etc.). Security architecture is a unified security design that addresses when and where to apply security controls and how these controls relate to the overall system design. The architecture should be analyzed to ensure it is structured to meet desired security needs.

RMF Controls Addressed

SA-17 Developer Security Architecture and Design

SA-4(2) Acquisition Process | Design/Implementation Information for Security Controls

4. Software Security Architecture and Design

Examples

Secure Architecture Design

- Industrial Control Systems diagrams: https://ics-cert.us-cert.gov/Secure-Architecture-Design
- Case study development of an information system security architecture https://www.sans.org/reading-room/whitepapers/auditing/information-systems-security-architecture-approach-layered-protection-1532

Architecture Analysis Approaches

- Quality Attribute Workshop (QAW) - facilitated method that engages system stakeholders early in the lifecycle to discover the driving quality attribute requirements of a software-reliant system
- Architecture Tradeoff Analysis Method (ATAM) - method for evaluating software architectures relative to quality attribute goals; provides insight into how those quality goals interact with each other—how they trade off

4. Software Security Architecture and Design

Key Considerations:

- Design choices must be made since qualities are frequently in conflict. The Secure Software Development Lifecycle must include ways for making trade-off choices to meet risk tolerances by decision makers knowledgeable about software security risk.
- Design weaknesses allow attackers to bypass security controls; these cannot be patched later and will require redesign to address if inappropriate risk choices are made
- Architects and designers must be knowledgeable in the ways in which design weaknesses occur and can impact system and software security

Evidence:

Software architecture is clearly described and the qualities that the architecture emphasizes (prioritizing requirements) are clearly defined

Security risk is considered within the architecture decision making

5. Software Configuration Management

Description

Software configuration management is the task of tracking and controlling changes in the software, part of the larger cross-disciplinary field of configuration management. Poor configuration management practices are a major source of software problems (e.g. code added from inappropriate libraries with known malware).

RMF Controls Addressed

SA-10 Developer Configuration

SA-10 (1) Developer Configuration Management | Software/Firmware Integrity Verification

5. Software Configuration Management

Guidance for Configuration Management

Top 10 Best Practices in Configuration Management (2007)

https://www.cmcrossroads.com/article/top-10-best-practices-configuration-management

Software Configuration Management Best Practices
https://www.microfocus.com/media/white-paper/software_configuration_management_best_practices_wp.pdf

Introduction to Configuration Management Best Practices: Practical Methods that Work in the Real World (2010) http://www.informit.com/articles/article.aspx?p=1622259

Examples: Open source security issues

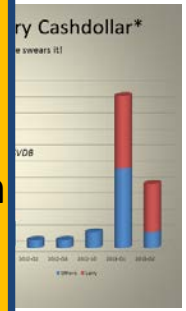
Heartbleed and Shellshock were found by exploitation

Other open source software illustrates vulnerabilities from cursory inspection

1.8 billion vulnerable open source components downloaded in 2015

26% of the most common open source components have high risk vulnerabilities

On average, applications have 22.5 open source vulnerabilities



Sources: Steve Christey (MITRE) & Brian Martin (OSF), Buying Into the Bias: Why Vulnerability Statistics Suck, https://media.blackhat.com/us-13/US-13-Martin-Buying-Into-The-Bias-Why-Vulnerability-Statistics-Suck-Slides.pdf; Sonatype, Sonatype Open Source Development and Application Security Survey; Sonatype, 2016 State of the Software Supply Chain; Aspect Software "The Unfortunate Reality of Insecure Libraries," March 2012, Mike Pittenger, Black Duck, "Open Source Security Analysis," 2016

5. Software Configuration Management

Key Considerations:

- Software configuration management (and effective tools) must be consistently applied to all software assets and integrated into the system and software development lifecycle
- Contractors should explain how software developed in the supply chain will be integrated into their configuration management
- Information about security fixes are too frequently buried within software feature updates and not effectively validated in updates

Evidence:

Each software build clearly tracks each element to the configuration management environment.

6. Developer Security Testing and Evaluation

Description

Security testing involves a range of tools and specialized skills that are not part of the standard software testing process. The Security Testing and Evaluation plan needs to define what tools will be used, who will be using the tools (skill levels), tool coverage across the code and plans for addressing gaps, when the testing will be done and how results will be reported. A good test plan establishes confidence that software security requirements are appropriately addressed.

RMF Controls Addressed

SA-11 Developer Security Testing and Evaluation

6. Developer Security Testing and Evaluation

Guidelines

- Testing Guide Introduction, which includes a section on security testing integrated into development https://www.owasp.org/index.php/Testing_Guide_Introduction#Deriving_Security_Test_Requirements

- Writing Software Security Test Cases http://www.qasec.com/2007/01/writing-software-security-test-cases.html

Security Testing - Government Tool Resources

NIST Software Assurance Metrics and Tool Evaluation (SAMATE) (http://samate.nist.gov/index.php/Main_Page)

Evaluates available tools against standard vulnerabilities to objectively demonstrating their use on real software

Software Assurance Marketplace (https://continuousassurance.org/)

Provides free ready-to-use computing platform and tools or the SWAMP-in-a-Box (SiB) open-source distribution for code security testing

SOAR

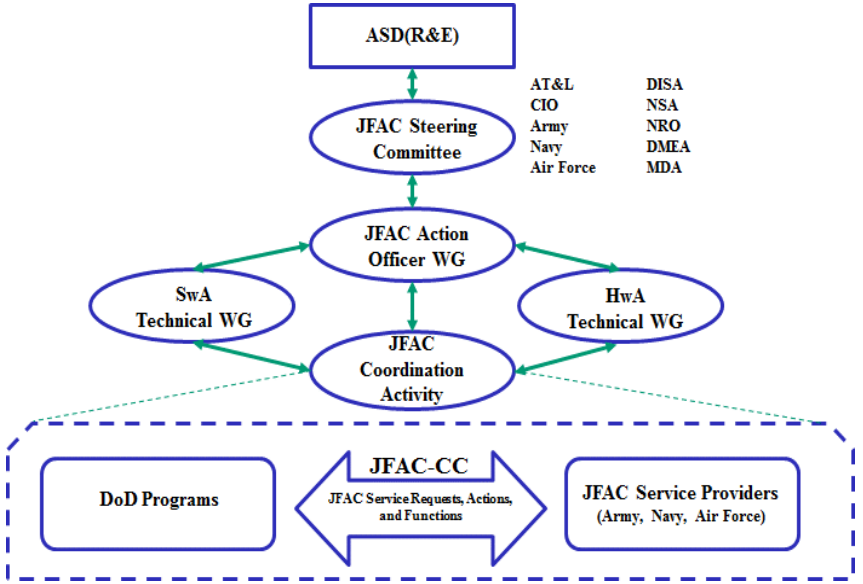
State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation (http://www.acq.osd.mil/se/docs/P-8005-SOAR-2016.pdf)

IDA supports this to assist those making effective software assurance (SwA) and supply chain risk management (SCRM) decisions

DoD Software Assurance Resources

DoD Joint Federated Assurance Center (JFAC)

- **Service providers** designated by each DoD Service Component (Army, Navy, Air Force) and available to DoD program offices
- **Managed** by the JFAC Coordination Center (JFAC-CC)



6. Developer Security Testing and Evaluation

Key Considerations:

- Security test cases should:
  - be built as the system is designed to ensure full confirmation of security requirements
  - validate that security is properly blocking inappropriate behaviors as well as allowing proper behaviors
- Security testing should include identification and prioritization of software vulnerabilities
- Security testing should focus on confirming the software is built to function as intended and does not exhibit unstable behaviors that can be compromised

Evidence:

- Reduction in uncaught breaches after implementation
- Improved quality metrics (reduced defects)

7. Static Code Analysis

Description

Static code analysis tools evaluate code for specific software weaknesses that could lead to software vulnerabilities without actually executing the code; these tools are specific to a development environments, coding language, code parts (e.g. source, bytecode, or binary) and identify structures that are likely candidates for problems.

RMF Controls Addressed

SA-11 (1) Developer Security Testing and Evaluation | Static Code Analysis

7. Static Code Analysis

Guidance and Related Information

- State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation (http://www.acq.osd.mil/se/docs/P-8005-SOAR-2016.pdf) describes the various types of static code analysis that should be addressed by the contractor.
- Many tools exist with varying effectiveness. NIST SAMATE project periodically evaluates available static analysis tools against a set of code (Juliet suite) that was written to include 11 of the top 25 CWEs to objectively demonstrate the capabilities of each tool. https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html
- Integrated Developer Environments (IDEs) that include analysis tools allow coders to check their code as they write it (e.g. FindBugs (Java) can be integrated into Eclipse and Jdeveloper)

7. Static Code Analysis

Key Considerations:

- Broad code coverage is needed along with multiple tools to ensure breadth of CWE coverage (described in the STP)
- There are many types of static analysis (e.g., source analysis, binary analysis, origin analysis) each requires different source material, uses different tools, and addresses different kinds of software weaknesses
- Tools must be effectively tuned to identify critical software problems without identifying too many false positives that frustrate the developers who may ignore them, turn off flags or skip tool use. Tool set up and monitoring should be handled by knowledgeable resources.

Evidence:

Developers are trained to avoid repeating the same types of mistakes

Improved code quality has shown a reduction in false positives by static analysis tools

8. Dynamic Code Analysis

Description

Dynamic code analysis is an evaluation of the software as it executes on a real or virtual processor using sufficient test inputs to evaluate a wide range of possible behaviors.

These tools should generate runtime vulnerability scenarios through the following functions: perform file corruption, resource fault injection, network fault injection, system fault injection, and user interface fault injection attacks.

Types of dynamic code analysis include network scanners, network sniffers, network vulnerability scanners, host-based vulnerability scanners, and host application interface scanners.

RMF Controls Addressed

SA-11 (8) Developer Security Testing and Evaluation | Dynamic Code Analysis

8. Dynamic Code Analysis

Guidance and Tools

- State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation (http://www.acq.osd.mil/se/docs/P-8005-SOAR-2016.pdf) describes the various types of dynamic code analysis that should be addressed by the contractor.
- Open source or free tools list (June 2016) https://www.peerlyst.com/posts/resource-a-list-of-dynamic-analysis-tools-for-software-susan-parker
- Another dynamic code analysis tool list is available (September 2017) at https://en.wikipedia.org/wiki/Dynamic_program_analysis

8. Dynamic Code Analysis

Key Considerations:

- Use of measures such as code coverage help ensure that an adequate slice of the software's set of possible behaviors has been observed.
- Tools are specific to operating system platforms and types of analysis; multiple tools are needed to identify a sufficient range of problem software behaviors.

Evidence:

Developers are trained to avoid repeating the same types of mistakes

Uncaught vulnerabilities should be reducing over time

9. Manual Code Reviews

Description

Code review is a systematic examination (sometimes referred to as peer review) of computer source code. It is intended to find mistakes overlooked in software development, improving the overall quality of software.

Tools miss a lot of issues lurking in code and need to be backstopped by manual review undertaken by developers with security expertise who can overcome the limitations of these tools.

Reviews can vary in coverage and intensity: spot checks, specific reviews, IEEE 1028 formal inspections, and generated code inspections.

RMF Controls Addressed

SA-11 (4) Developer Security Testing and Evaluation | Manual Code Reviews

9. Manual Code Reviews

Guidance

- 5 Best Practices for Perfect Secure Code Review https://www.checkmarx.com/2016/02/05/5-best-practices-perfect-secure-code-review/
- OWASP Code Review Guide https://www.owasp.org/images/2/2e/OWASP_Code_Review_Guide-V1_1.pdf
- Code Review Checklist http://www.evoketechnologies.com/blog/code-review-checklist-perform-effective-code-reviews/
- 22 Point Code Review Checklist http://www.fromdev.com/2015/02/code-review-checklist.html

9. Manual Code Reviews

Key Considerations:

- Tools are not available for all programming languages and manual review is the only option for some software. The STP should identify specific review issues that require manual review.
- These reviews need to be conducted during development with resources familiar with the security requirements and the software language in use.
- Sufficient time that matches resource availability needs to be allowed in the schedule to perform manual reviews.

Evidence:

All code is reviewed at some point in the lifecycle (including legacy, third party, firmware)

Planning clearly notes what approach will be used for each type of code and how success will be measured.

10. Attack Surface Reviews

Description

The software attack surface includes all of the ways an unauthorized user (attacker) could reach the software to exploit vulnerabilities or extract data; this includes paths from access points external to the system, linked system components including third-party products such as operating systems, and network connections. Reducing the number of attack paths has been shown to improve software protection.

RMF Controls Addressed

SA-11 (6) Developer Security Testing and Evaluation | Attack Surface Reviews

10. Attack Surface Reviews

Guidance and tools

- Common Attack Pattern Enumeration and Classification (CAPEC) attack patterns (https://capec.mitre.org/) provides a list of typical attack patterns that can be used to evaluate the software attack surface
- An Attack Surface Analysis Cheat Sheet is available from Open Web Application Security Project (OWASP) https://www.owasp.org/index.php/Attack_Surface_Analysis_Cheat_Sheet
- Microsoft (MS) Attack Surface Analyzer is a tool created for the analysis of changes made to the attack surface of the MS operating systems since Windows Vista and beyond. It is available for public use at https://www.microsoft.com/en-us/download/details.aspx?id=24487

10. Attack Surface Reviews

Key Considerations:

- Limiting the attack surface must be a requirement for the overall system not just software components
- This analysis should begin this type of analysis early in the life cycle to allow for the identification and use of attack reduction opportunities in architecture and design decisions
- Externally developed software can increase the attack surface and should be part of the analysis - – choices in components can change the structure of the system increasing attack risk
- Testing should verify that the planned attack surface matches the actual build

Evidence:

Attack surface considerations influence user stories and are included in evaluation of design and development options.

11. Software Threat Analysis

Description

Software threat modeling is an analysis practice for evaluating the expected software security by analyzing the context in which the software operates, the threat sources and potential vulnerabilities. Appropriate countermeasures must be defined, as needed, to prevent or mitigate the effects of high risk threats to the software.

RMF Controls Addressed

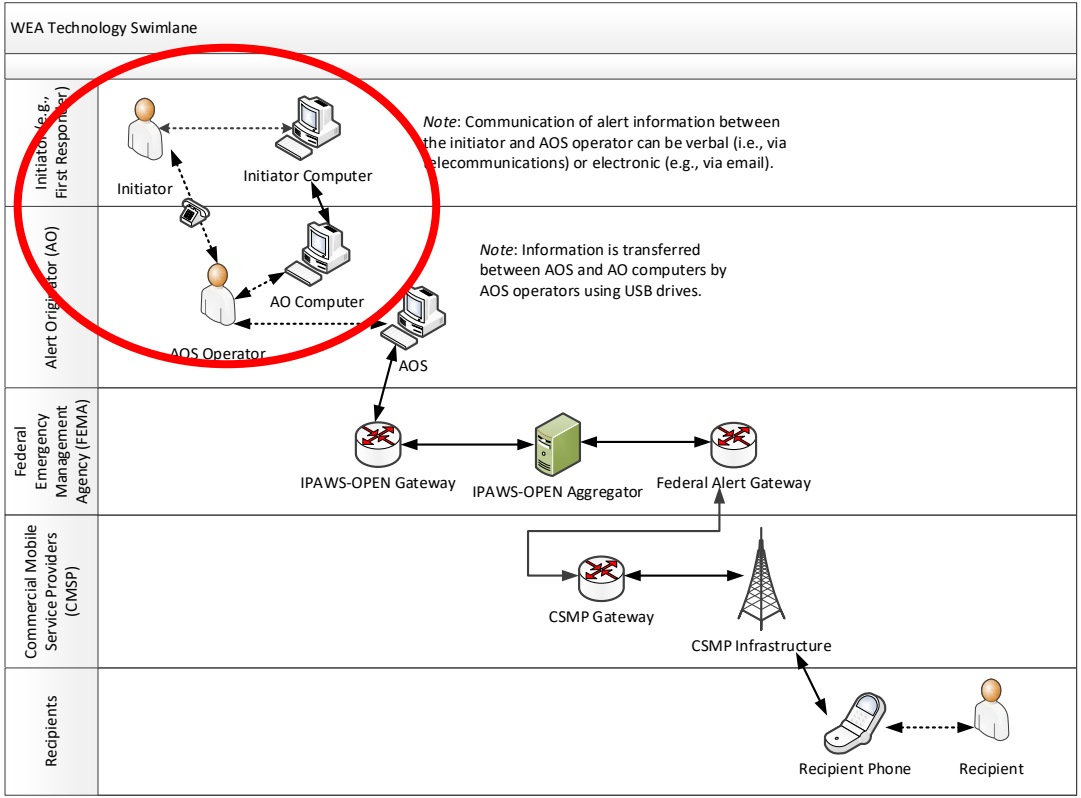
SA-11(2) Developer Security Testing and Evaluation | Threat and Vulnerability Analysis

11. Software Threat Analysis

Examples

- MS STRIDE is a structured approach (with a tool) for evaluating software for a typical set of software threats: spoofing identity, tampering data, repudiation, information disclosure, denial of service, and elevation of privilege.
- DREAD is a structure for quantifying, comparing and prioritizing the amount of risk presented by each evaluated threat based on the following: damage, reproducibility, exploitability, affected users, and discoverability
- SEI SERA (Security Engineering Risk Analysis) is currently in use at MDA BMDS to identify potential system threats that could be triggered by software
- System Theory Process Analysis for Security (STPA-Sec) is an approach that evaluates redesign options to address security threats

Software Context Must Match Security Concerns: *Wireless Emergency Alerting System*



Recent Hawaii incident involved sending a inaccurate public notice of a missile attack – the software did not require multiple confirming authorizations and a single bad actor created international havoc

11. Software Threat Analysis

Key Considerations:

- Incomplete or inaccurate operational context for the software will result in missed or inappropriate threat considerations
- Analysis must cover software within the context of all system components.
- The impact of interfaces from external systems cannot be ignored as they represent critical context threat vectors

Evidence:

User stories include consideration of expected threats

Monitoring in DevOps includes a focus on expected threats

Software threat monitoring includes consideration of the operational context, physical environment, and all external interfaces and influences

12. Penetration and Fuzz Testing

Description

Penetration testing (aka ethical hacking) is an authorized effort to evaluate the software security by attempting to exploit software vulnerabilities in a controlled environment.

Fuzz testing (fuzzing) involves inputting large volumes of invalid and random data to see how well the software can handle the unexpected. A wide range of refinement capabilities are available to make fuzzing more useful beyond just crashing the software. Framework-based fuzzers include instrumentation to guide the data generation to improve relevancy.

RMF Controls Addressed

SA-11(5) Developer Security Testing and Evaluation | Penetration Testing/Analysis

12. Penetration and Fuzz Testing

Guidance and Training

- Penetration testing involves the use of many tools and requires resources trained in effectively applying the available tools; boot camps for training are available such as: https://www.infosecinstitute.com/courses/ethical-hacking-boot-camp?gclid=EAIaIQobChMI-4SvqdCb2AIVkoF-Ch0IAg6YEAMYASAAEgLW2PD_BwE
- Fuzz testing tutorial is available such as https://www.guru99.com/fuzz-testing.html Synopsys Defensics shows up frequently in fuzzing tool searches

12. Penetration and Fuzz Testing

Key Considerations:

These need to be executed as part of the standard software lifecycle activities and performed consistently for all software development and as part of the regression testing for all updates.

Resources who are knowledgeable in the use of the tools are required for the results to be of value

Evidence:

Testing includes consideration of a wide range of unexpected data

13. Verifying Scope of Testing and Evaluation

Description

Software testing is an investigation conducted to provide information about the quality of the software product or service under evaluation. Software security testing needs to provide a view of the software in operation to appreciate and understand the risks of the software implementation. Test techniques also need to include the process of executing the software with the intent of finding software vulnerabilities and verifying that the software product is fit for use.

RMF Controls Addressed

SA-11(7) Developer Security Testing and Evaluation | Verify Scope of Testing/Evaluation

13. Verifying Scope of Testing and Evaluation

Key Considerations:

- Attackers do not distinguish between arbitrary boundaries for software, software security, and systems so it is important that the testing ensure full coverage within a system and with external interfaces.
- The plan for software testing and the evaluation of its sufficiency should be part of the system test planning.
- Actual software security testing can be extremely complex and difficult to execute; the process needs to be evaluated closely for sufficiency to confirm security requirements; the closer the test environment is to the live environment the greater the confidence.

Evidence:

Code coverage includes considerations of which tools are handle each type of issue

Evaluation of uncaught breaches identifies which tools were incomplete or inconsistently applied to discover the problems earlier

14. Independent Verification of Assessment Plans/Evidence

Description

Software test planning and test execution (including software security) requires independent verification in the same manner as system testing, but those performing the independent assessment must have appropriate skills in the use of the tools and analysis techniques selected to generate the evidence.

RMF Controls Addressed

SA-11(3) Developer Security Testing and Evaluation | Independent Verification of Assessment Plans/Evidence

14. Independent Verification of Assessment Plans/Evidence

Key Considerations:

- The context of independent verification should be as close to actual operational context as feasible to ensure effective confirmation of software security.
- Verifiers must be knowledgeable about software and the ways in which security vulnerabilities are attacked.
- The program must provide sufficient time and resources for this independent review.

Evidence:

Operational support resources should be part of the review of development results to ensure they will not break (or easily attacked) after implementation

15. Software Flaw Remediation

Description

All software contains defects and vulnerabilities. As these are discovered and addressed, corrective actions will be needed both during and after system development. A flaw remediation process must include all flaw handling for all types of software and needs to interface with the configuration management process. Security-relevant software updates including patches, service packs, hot fixes, and anti-virus signatures need to be planned for and addressed as criticality warrants. Organizations must also address flaws and vulnerabilities discovered during security assessments, incident response activities, and system error handling.

RMF Controls Addressed

SI-2 Flaw Remediation
SI-2(1) Flaw Remediation | Central Management
SI-2(2) Flaw Remediation | Automated Flaw Remediation Status
SI-2(3) Flaw Remediation | Time to Remediate Flaws/Benchmarks for Corrective Actions
SI-2(6) Flaw Remediation | Removal of Previous Versions of Software/Firmware

15. Software Flaw Remediation

Key Considerations:

- Consistency in planning for and addressing flaws can increase confidence that the contractor understands how to effectively handle software security.
- Remediation should include a consistent prioritization and tracking of unaddressed flaws and the risk this represents.
- If the contractor is not also handling software sustainment, a means of informing those assuming this responsibility of the residual software risk will be needed.
- The cost of remediation will be less the closer it is performed to the flaw creation.

Evidence:

All types of defects are identified, tracked, and addressed not just those identified as high priority at the moment – attacker capabilities are continually increasing and today's low priority weaknesses become tomorrows incidents

16. Malicious Code Protection

Description

Malicious code insertions occur through the exploitation of software and system vulnerabilities. In addition, external access capabilities and trusted interfaces with software from external systems can provide a mechanism for malicious code insertion. Protection mechanisms include the reduction of the attack surface, removal of software weaknesses, good configuration management, and operational code validity controls (e.g. code signatures).

RMF Controls Addressed

SI-3 Malicious Code Protection
SI-3(1) Malicious Code Protection | Central Management
SI-3(2) Malicious Code Protection | Automatic Updates
SI-3(10) Malicious Code Protection | Malicious Code Analysis

16. Malicious Code Protection

Guidance

- What is Malicious Code? https://www.veracode.com/security/malicious-code; http://www.informit.com/articles/article.aspx?p=31782&seqNum=3
- Malicious Code – What Should We Do? https://www.sans.org/reading-room/whitepapers/malicious/malicious-code-do-1290
- How to Prevent Malicious Code https://www.checkmarx.com/glossary/how-to-prevent-malicious-code/

16. Malicious Code Protection

Key Considerations:

- Mechanisms for ensuring the integrity of software code (including protection from malicious code insertion) should be part of the configuration management and structured within the practices used in the Secure Software Development Life Cycle for establishing and maintaining development and deployment environments.
- Verification should be in place for acceptance of software from suppliers to ensure no malicious code is accepted.

Evidence:

Development environments are supported for security in the same manner as operational environments (patched and monitored)

Configuration management covers all types of software in every context to ensure bad code is not picked up anywhere

17. Software and Firmware Integrity

Description

Software and firmware integrity involves verification between the current file state and a known, good baseline that typically involves calculating a known cryptographic checksum of the baseline and comparing with the calculated checksum of the current state of the file.

RMF Controls Addressed

SI-7 Software, Firmware, and Information Integrity

SI-7(1) Software, Firmware, and Information Integrity | Integrity Checks

17. Software and Firmware Integrity

Guidance and tools

- Firmware Integrity, Verification, and Monitoring Tool with Mapping to NIST Guidelines https://csrc.nist.gov/Presentations/2015/Firmware-Integrity-Verification,-Monitoring-and-Re
- 10 Tools to Verify File Integrity Using MD5 and SHA1 Hashes https://www.raymond.cc/blog/7-tools-verify-file-integrity-using-md5-sha1-hashes/
- File Integrity Monitoring https://en.wikipedia.org/wiki/File_integrity_monitoring

17. Software and Firmware Integrity

Key Considerations:

- Practices for integrity confirmation need to be part of the planned Secure Software Development Life Cycle.
- Software acceptance practices for third party software needs to include integrity confirmation.
- Integrity confirmation should be coordinated with configuration management processes.

Evidence:

Software integrity is verified with the baseline any time it is implemented or changed.

18. Software Supply Chain Protection

Description

Ensure that appropriate software assurance practices are implemented in the software supply chains. A supply chain can include components, code libraries, code generation tools, COTS, firmware, and open source products

RMF Controls Addressed

SA-12 Supply Chain Protection

SA-12(1) Supply Chain Protection | Acquisition Strategies/Tools/Methods

SA-12(5) Supply Chain Protection | Limitation of Harm

SA-12(8) Supply Chain Protection | Use of All-Source Intelligence

SA-12(9) Supply Chain Protection | Operations Security

SA-12(11) Supply Chain Protection | Penetration Testing/Analysis of Elements, Processes, and Actors

SA-22 Unsupported System Components

18. Software Supply Chain Protection

Key Considerations:

- Contractors may have existing supplier contracts that do not reflect effective software security and compensations will need to be implemented as part of software acceptance.
- Mechanisms for verifying that software supply chain protections are applied in practice are needed (e.g. sampling)

Evidence:

Transparency as to how software is selected and integrated at each level in the supply chain will increase confidence that software supply chain protections are in place.

Case Studies & Summary

Example 1: Federal Acquisition Focused Cybersecurity

19 new contract requirements will be added to all new contracts issued or updated after April 2018

- Selected from Federal recommended sources: NIST 800-53, NIST 800-53A, and CNSSI No. 1253
- Addressing 37 key security RMF controls

Requirements for nine existing system deliverables have been expanded to add software information.

Four new contract deliverables have been added to report specific information about threats and vulnerabilities.

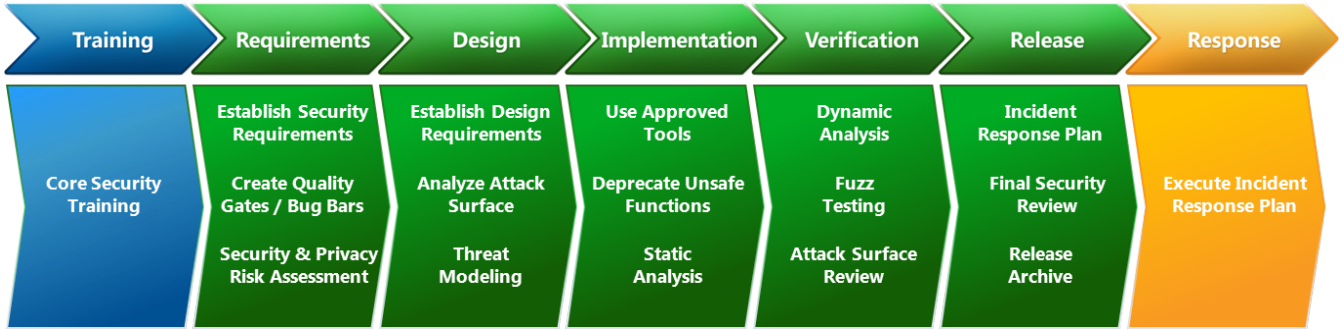
Engineering technical assessment criteria have been updated for:

- System Requirements Review (SRR)
- Preliminary Design Review (PDR)
- Critical Design Review (CDR)
- Test Readiness Review (TRR)

Lifecycle choices are not specified

Example 2: Microsoft Security Development Lifecycle (MS SDL)

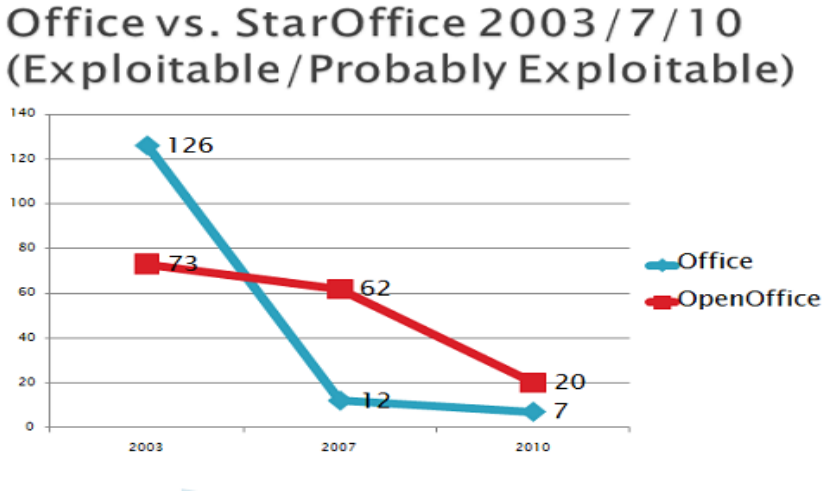
The Security Development Lifecycle (SDL) is a software development security assurance process consisting of security practices grouped by seven phases.



Reference: http://www.microsoft.com/security/sdl/learn/measurable.aspx

CERT Secure Practices Mapped to MS SDL http://www.cert.org/archive/pdf/MS_CERT_SDL.pdf

MS SDL as Practiced at Microsoft



Dan Kaminsky - Fuzzmarking: Towards Hard Security Metrics For Software Quality?

http://dankaminsky.com/2011/03/11/fuzzmark/

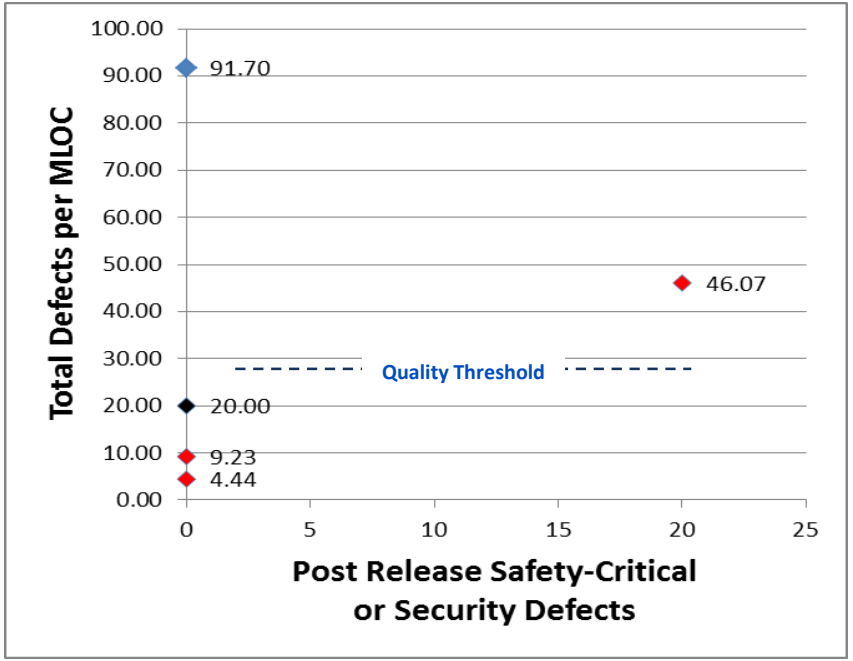
Microsoft mandatory development policy since 2004

- Designed to reduce the number and severity of vulnerabilities in Microsoft software
- Specifically tailored to Microsoft development practices and business drivers
- Designed for enterprise scale software development
- Longitudinal 3rd party studies show definite impact (see graphic)

Example 3: Quality Informs Security Risk Predictions

Org.	Project	Type	Secure or Safety Critical Defects	Defect Density	Size
D	D1	Safety Critical	20	46.07	2.8 MLOC
D	D2	Safety Critical	0	4.44	.9 MLOC
D	D3	Safety Critical	0	9.23	1.3 MLOC
A	A1	Secure	0	91.70	.6 MLOC
T	T1	Secure	0	20.00	.1 MLOC

Data from five projects with low defect density in system testing reported very low or zero safety critical and security defects in production use.



Woody, Carol et al. *Predicting Software Assurance Using Quality and Reliability Measures.* CMU/SEI-2014-TN-026. Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=428589

Example 3: Critical Metrics Tracked

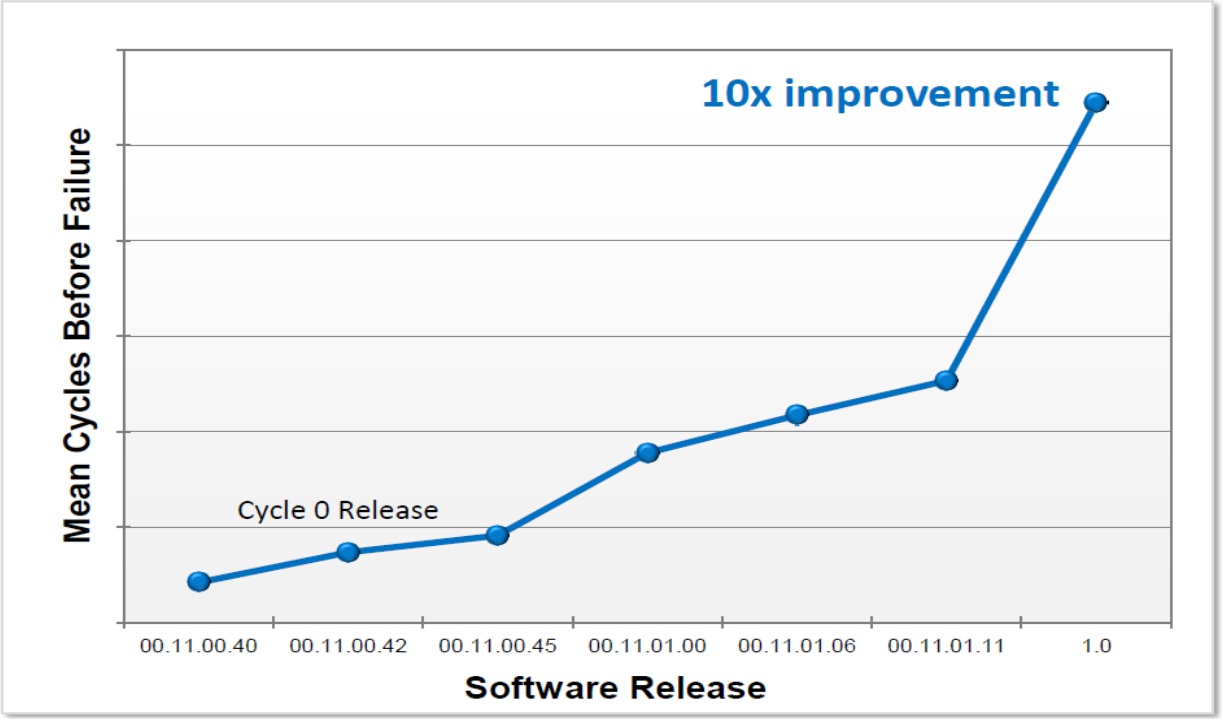
Development Metrics

- Incoming/week
- Triage rate
- % closed
- Development work for cycle
- Software change request per developer (SCR/Dev) per week
- # developers
- Protocol work
- Software change request per safety verifier & validator (SCR/SVV) per week
- # verification persons

Software Change Metrics

- Fixed work per cycle
- Deferred planned work per cycle

Example 3: Reliability Tracking from Operations



Summary

Software quality matters for security

- Any project could evaluate their security relative to quality using current data sources with a calibrated quality model.

Consist implementation and monitoring of security practices matters (automate and monitor)

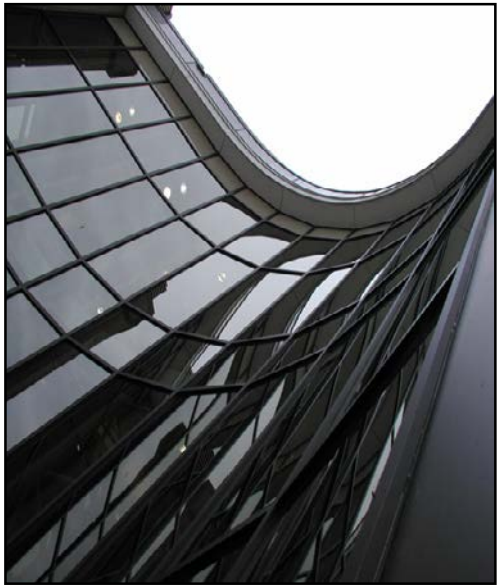
Knowledgeable use of security tools matters

Operational experience must inform development (automate with DevOps)

- Augment development quality evidence with sustainment reliability data

Assembling quality and reliability data across multiple projects and cycles within projects supports a baseline from which organizational standards of sufficiency can be confirmed

Contact Information



Carol Woody, Ph.D.
cwoody@cert.org

Web Resources (CERT/SEI)

http://www.sei.cmu.edu/