NORTH AMERICAN
PUBLIC SECTOR

Where the Rubber Meets the Code – Static Code Analysis for Software Assurance in the Acquisition Life Cycle

Paul R. Croll
Fellow
CSC
pcroll@csc.com

Outline

NORTH AMERICAN
PUBLIC SECTOR

Setting the Stage for Static Code Analysis

What is Static Code Analysis?

- Static code analysis is the process of evaluating a system or component based on its form, structure, content, or documentation. From a software assurance perspective, static analysis addresses weaknesses in program code that might lead to vulnerabilities
- Such analysis may be manual, as in code inspections, or automated through the use of one or more tools
- Automated static code analyzers typically check source code but there is a smaller set of analyzers that check byte code and binary code, especially useful when source code in not available (e.g for COTS components).

The Scope of The Problem

Size in FP	Civilian Government Projects	Military Projects	*Average*
1	1	1	*1*
10	5	4	*5*
100	29	14	*24*
1,000	155	55	*120*
10,000	832	209	*600*
100,000	4,467	794	*3,031*
1,000,000	23,988	3,020	*15,412*
Average	*4,211*	*585*	*2,742*

Figure 1. Estimated Number of Security Vulnerabilities in Software Applications. Source: Capers Jones © 2008

Size in FP	Civilian Government Projects	Military Projects	*Average*
1	25.00%	5.00%	*11.29%*
10	35.00%	15.00%	*26.00%*
100	45.00%	20.00%	*33.57%*
1,000	62.00%	30.00%	*54.57%*
10,000	80.00%	35.00%	*74.00%*
100,000	87.00%	40.00%	*80.14%*
1,000,000	92.00%	45.00%	*86.29%*
Average	*60.86%*	*27.14%*	*52.27%*

Figure 2. Probability of Serious Security Vulnerabilities in Software Applications. Source: Capers Jones © 2008

- For military projects, as one approaches systems the size of typical large combat systems (expressed as function points), the estimated number of security vulnerabilities rises to above 3000 and the probability of serious vulnerabilities rises to over 45%
- The statistics are much worse for civilian systems. As we move more and more into COTS and open source software for our combat systems, one might expect that the true extent of vulnerabilities in our systems would lie somewhere between those of military and civilian systems.

COTS and Open Source Exacerbate the Problem

- Reifer and Bryant [2] studied 100 packages were selected at random from 50 public Open-Source, COTS, and GOTS libraries
  - Spanned a full range of applications and sites like SourceForge
  - Over 30% of Open Source and GOTS (Government Off the Shelf) packages analyzed had dead code
  - Over 20% of the Open Source, COTS, and GOTS packages had suspected malware
  - Over 30% of the COTS packages analyzed had behavioral problems
- Reifer and Bryant conclude that the potential for malicious code in applications software is large as more and more packages are used in developing a system.

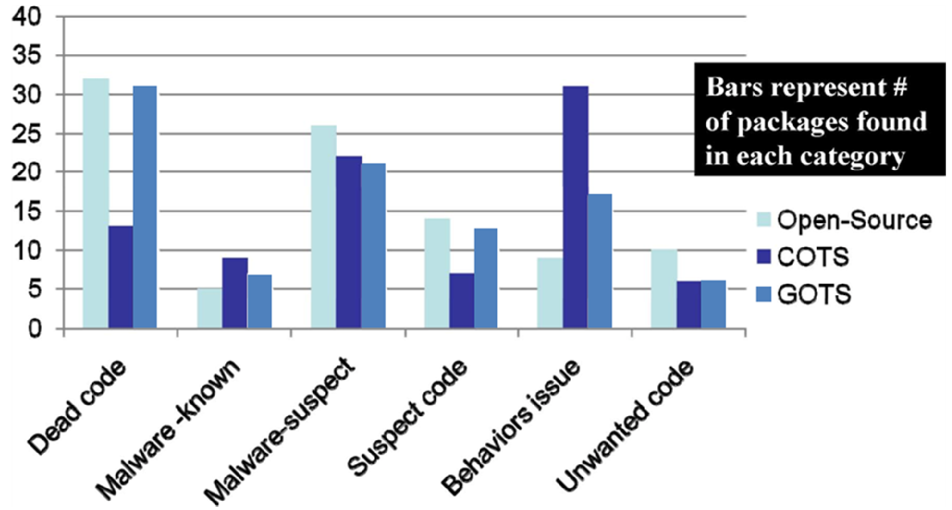


Figure 3. COTS Study Findings. *Source: D. Reifer and E. Bryant, Software Assurance in COTS and Open Source Packages, DHS Software Assurance Forum, October 2008*

DoD Clarifying Guidance Regarding Open Source Software (OSS) – October 16, 2009

2. GUIDANCE

a. In almost all cases, OSS meets the definition of "commercial computer software" and shall be given appropriate statutory preference in accordance with 10 USC 2377 (reference (b)) (see also FAR 2.101(b), 12.000, 12.101 (reference (c)); and DFARS 212.212, and 252.227-7014(a)(1) (reference (d))).

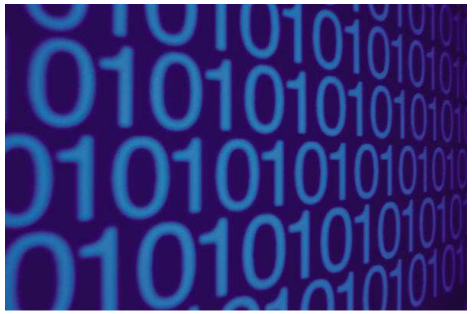
c. DoD Instruction 8500.2, "Information Assurance (IA) Implementation," (reference (g)) includes an Information Assurance Control, "DCPD-1 Public Domain Software Controls," which limits the use of "binary or machine-executable public domain software or other software products with limited or no warranty," on the grounds that these items are difficult or impossible to review, repair, or extend, given that the Government does not have access to the original source code and there is no owner who could make such repairs on behalf of the government. This control should not be interpreted as forbidding the use of OSS, as the source code is available for review, repair and extension by the government and its contractors.

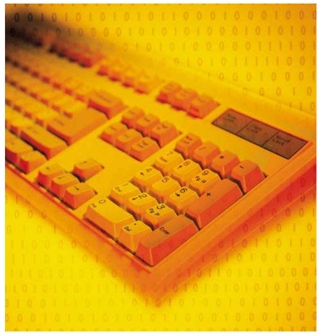
d. The use of any software without appropriate maintenance and support presents an information assurance risk. Before approving the use of software (including OSS), system/program managers, and ultimately Designated Approving Authorities (DAAs), must ensure that the plan for software support (e.g., commercial or Government program office support) is adequate for mission need.

Source: DoD Chief Information Officer (CIO) Memorandum, "Clarifying Guidance Regarding Open Source Software (OSS) in the Department of Defense (DoD)," October 16, 2009

Testing vs. Static Code Analysis

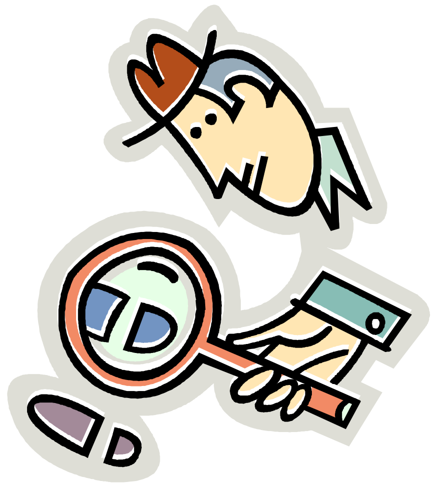
- Testing requires code that is relatively complete
- Static analysis can be performed on modules or unfinished code [4]
- A static analysis tool is a program written to analyze other programs for flaws
  - Such analyzers typically check source code
  - A smaller set of analyzers can check byte code and binary code
- Manual analysis, or code inspection, can be very time-consuming, and inspection teams must know what security vulnerabilities look like in order to effectively examine the code
- Static analysis tools are faster and don't require the tool operator to have the same level of security expertise as a code inspector [5]





What Code Do You Analyze?

- How do you prioritize a code review effort when you have thousands of lines of source code, and perhaps object code to review?
- From a software assurance perspective, looking at attack surfaces is not a bad place to start [6]
  - A system's attack surface can be thought of as the set of ways in which an adversary can enter the system and potentially cause damage
  - The larger the attack surface, the more insecure the system [7]
  - Higher attack surface software requires deeper review than code in lower attack surface components.



Heuristics For Code Review – 1

- Howard proposes the following heuristics as an aid to determining code review priority [8]:
  - **Old code**
    - Older code may have more vulnerabilities than new code because newer code often reflects a better understanding of security issues
    - Code considered "legacy" code should be reviewed in depth.
  - **Code that runs by default**
    - Attackers often go after installed code that runs by default
    - Such code should be reviewed earlier and deeper than code that doesn't execute by default
    - Code running by default increases an application's attack surface
  - **Code that runs in elevated context**
    - Code that runs in elevated identities, e.g. root in *nix, for example, also requires earlier and deeper review because code identity is another component of attack surface.
  - **Anonymously accessible code**
    - Code that anonymous users can access should be reviewed in greater depth than code that only valid users and administrators can access
  - **Code listening on a globally accessible network interface**
    - Code that listens by default on a network, especially uncontrolled networks like the Internet, is open to substantial risk and must be reviewed in depth for security vulnerabilities

Heuristics For Code Review – 2

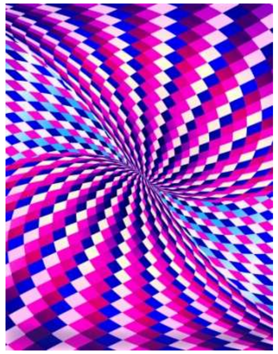
- **Code listening on a globally accessible network interface**
  - Code that listens by default on a network, especially uncontrolled networks like the Internet, is open to substantial risk and must be reviewed in depth for security vulnerabilities.
- **Code written in C/C++/assembly language**
  - Because these languages have direct access to memory, buffer-manipulation vulnerabilities within the code can lead to buffer overflows, which often lead to malicious code execution
  - Code written in these languages should be analyzed in depth for buffer-overflow vulnerabilities
- **Code with a history of vulnerabilities**
  - Code that's had a number past security vulnerabilities should be suspect, unless it can be demonstrated that those vulnerabilities have been effectively removed.
- **Code that handles sensitive data**
  - Code that handles sensitive data to should be analyzed to ensure that weaknesses in the code do not disclose such data to untrusted users.
- **Complex code**
  - Complex code has a higher bug probability, is more difficult to understand, and may likely have more security vulnerabilities.
- **Code that changes frequently**
  - Frequently changing code often results in new bugs being introduced
  - Not all of these bugs will be security vulnerabilities, but compared with a stable set of code that's updated only infrequently, code that is less stable will probably have more vulnerabilities in it

A Three-Phase Code Analysis Process – Phase 1

- Howard [6] also suggests a notional three-phase code analysis process that optimizes the use of static analysis tools.
- **Phase 1 – Run all available code-analysis tools**
  - Multiple tools should be used to offset tool biases and minimize false positives and false negatives
  - Analysts should pay attention to every warning or error
    - Warnings from multiple tools may indicate that the code that needs closer scrutiny (e.g. manual analysis).
  - Code should be evaluated early, preferable with each build, and re-evaluated at every milestone.

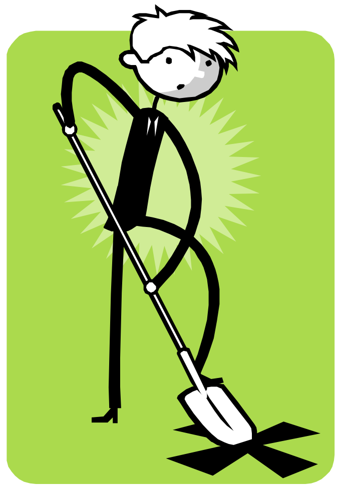
A Three-Phase Code Analysis Process – Phase 2

- **Phase 2 – Look for common vulnerability patterns**
  - Analysts should make sure that code reviews cover the most common vulnerabilities and weaknesses, such as integer arithmetic issues, buffer overruns, SQL injection, and cross-site scripting (XSS)
  - Sources for such common vulnerabilities and weaknesses include the Common Vulnerabilities and Exposures (CVE) and Common Weaknesses Enumeration (CWE) databases, maintained by the MITRE Corporation and accessible at: **http://cve.mitre.org/cve/** and **http://cwe.mitre.org/**
  - MITRE, in cooperation with the SANS Institute, also maintain a list of the "Top 25 Most Dangerous Programming Errors" (**http://cwe.mitre.org/top25/index.html**) that can lead to serious vulnerabilities
  - Static code analysis tool and manual techniques should at a minimum, address these Top 25



A Three-Phase Code Analysis Process – Phase 3

- **Phase 3 – Dig deep into risky code**
  - Analysts should also use manual analysis (e.g. code inspection) to more thoroughly evaluate any risky code that has been identified based on the attack surface, or based on the heuristics on Slides 10 and 11
  - Such code review should start at the entry point for each module under review and should trace data flow though the system, evaluating the data, how it's used, and if security objectives might be compromised

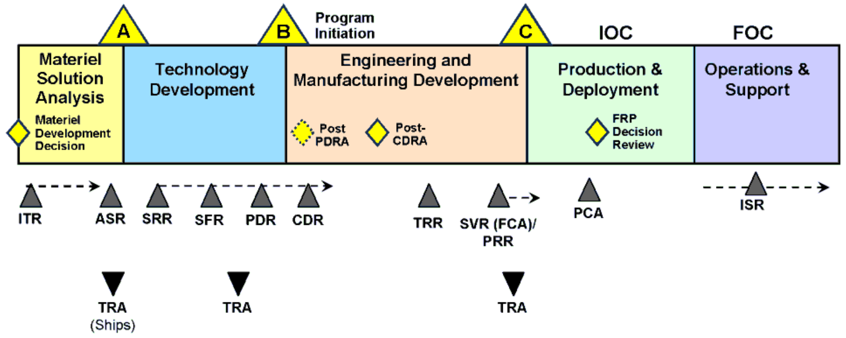


The Assurance Case – Capturing the Results of Static Code Analysis as Evidence for Assurance Claims

- An Assurance Case is a set of structured assurance claims, supported by evidence and reasoning that demonstrates how assurance needs have been satisfied [9]
  - It shows compliance with assurance objectives
  - It provides an argument for the safety and security of the product or service.
  - It is built, collected, and maintained throughout the life cycle
  - It is derived from multiple sources
- The Sub-parts of an assurance case include:
  - A high level summary
  - Justification that product or service is acceptably safe, secure, or dependable
  - Rationale for claiming a specified level of safety and security
  - Conformance with relevant standards and regulatory requirements
  - The configuration baseline
  - Identified hazards and threats and residual risk of each hazard and threat
  - Operational and support assumptions
- An Assurance Case should be part of every acquisition in which there is concern for IT security
  - Should be prepared by the supplier
  - Should describe
    - The assurance-related claims for the software being delivered,
    - The arguments backing up those claims,
    - The hard evidence supporting those arguments, **including static code analysis results**

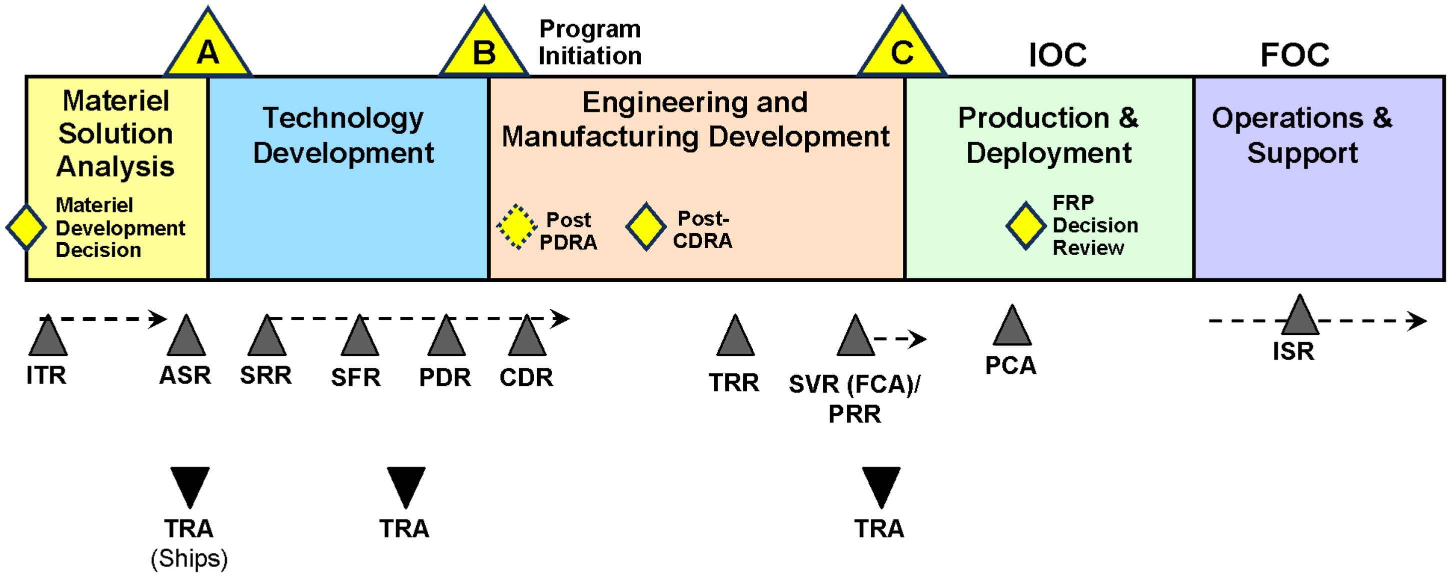
NORTH AMERICAN
PUBLIC SECTOR

Static Code Analysis in the Acquisition Life Cycle



System Engineering Technical Review Process (SETR)

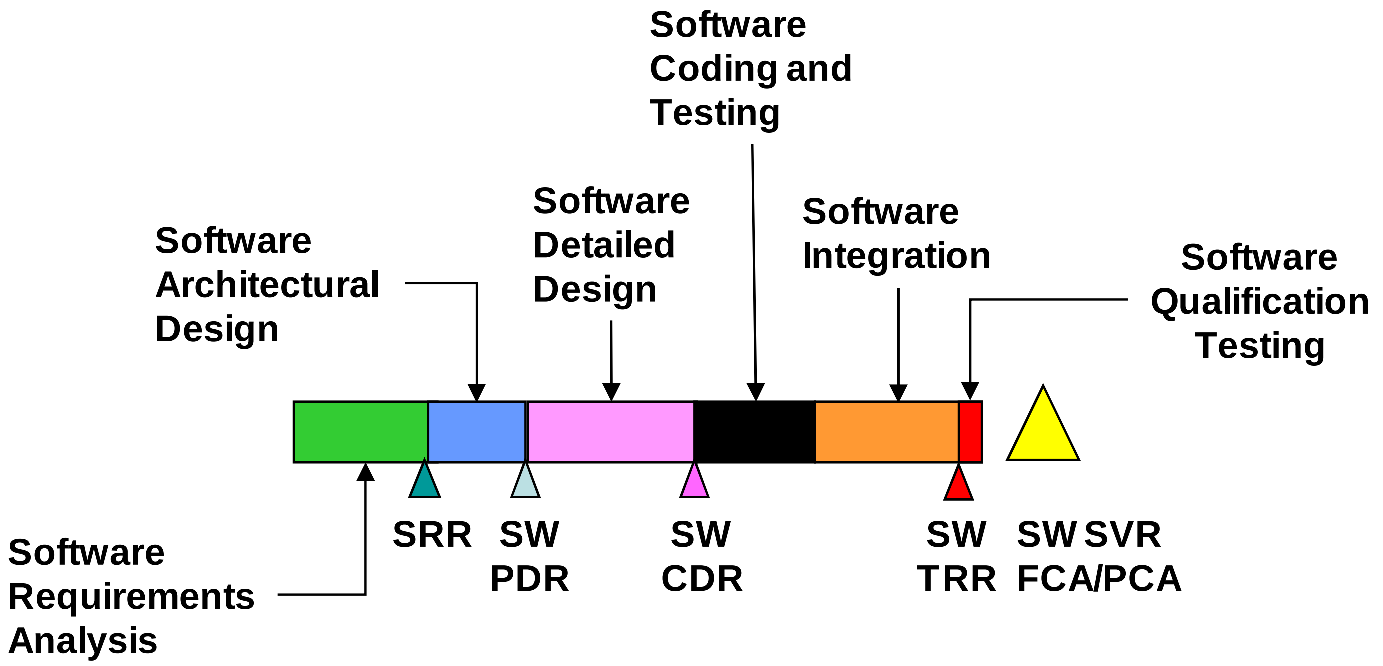
- DoDI 5000.02, Operation of the Defense Acquisition System [10], describes the System Engineering Technical Review (SETR) process associated with the system acquisition life cycle.



- Initial Technical Review (ITR)
- Alternative Systems Review (ASR)
- Systems Requirements Review (SRR)
- System Functional Review (SFR)
- Preliminary Design Review (PDR)
- Critical Design Review (CDR)
- Post-PDR Assessment (Post-PDRA)
- Post-CDR Assessment (PCDRA)
- Test Readiness Review (TRR)
- System Verification Review (SVR)
- Functional Configuration Audit (FCA)
- Production Readiness Review (PDR)
- Operational Test Readiness Review (OTRR)
- Physical Configuration Audit (PCA)
- Technology Readiness Assessment (TRA)
- In-Service Review (ISR)

Software CI Reviews

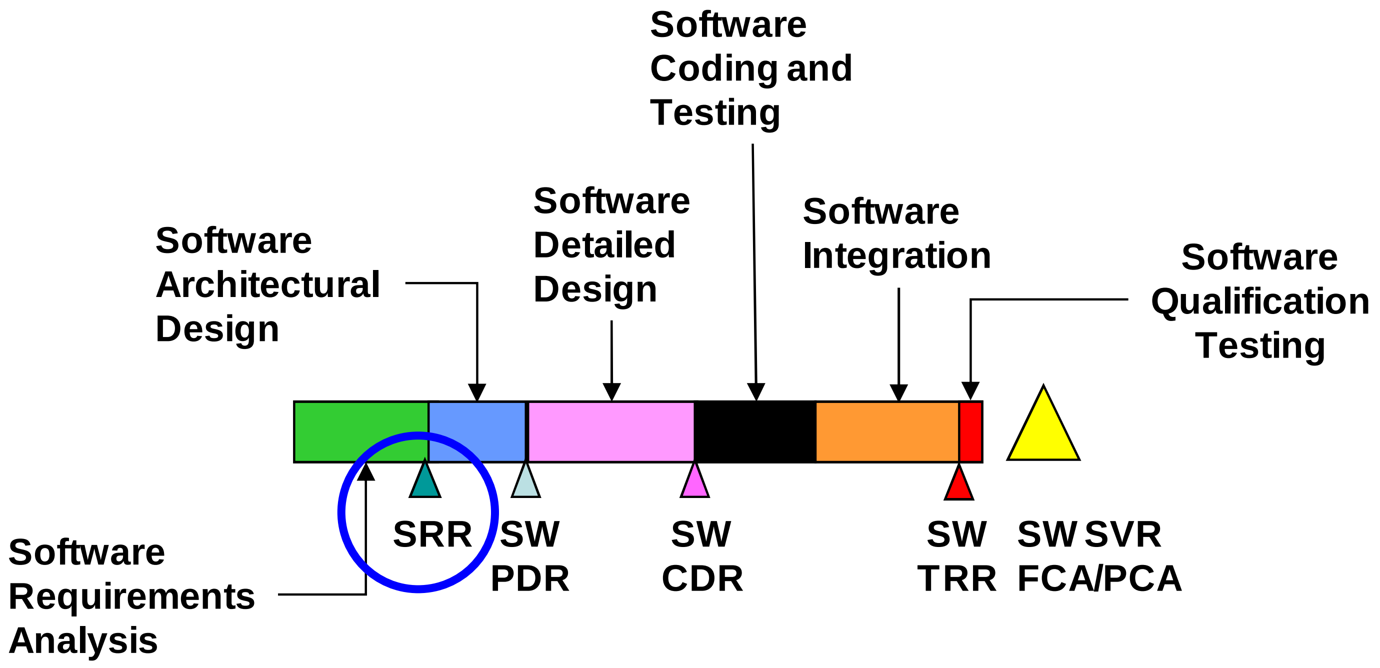
- The reviews typically associated with software are shown below [11]


Software Coding and Testing
Software Detailed Design
Software Integration
Software Architectural Design
Software Qualification Testing
Software Requirements Analysis
SRR
SW PDR
SW CDR
SW TRR
SW FCA/PCA
SVR

Source: PEO IWS Technical Review Manual (TRM), December 2008

System Requirements Review (SRR) Objectives

- The SRR helps the PM understand the scope of the software assurance landscape (assurance requirements, elements to be protected, the threat environment) in which context static code analysis should be applied.

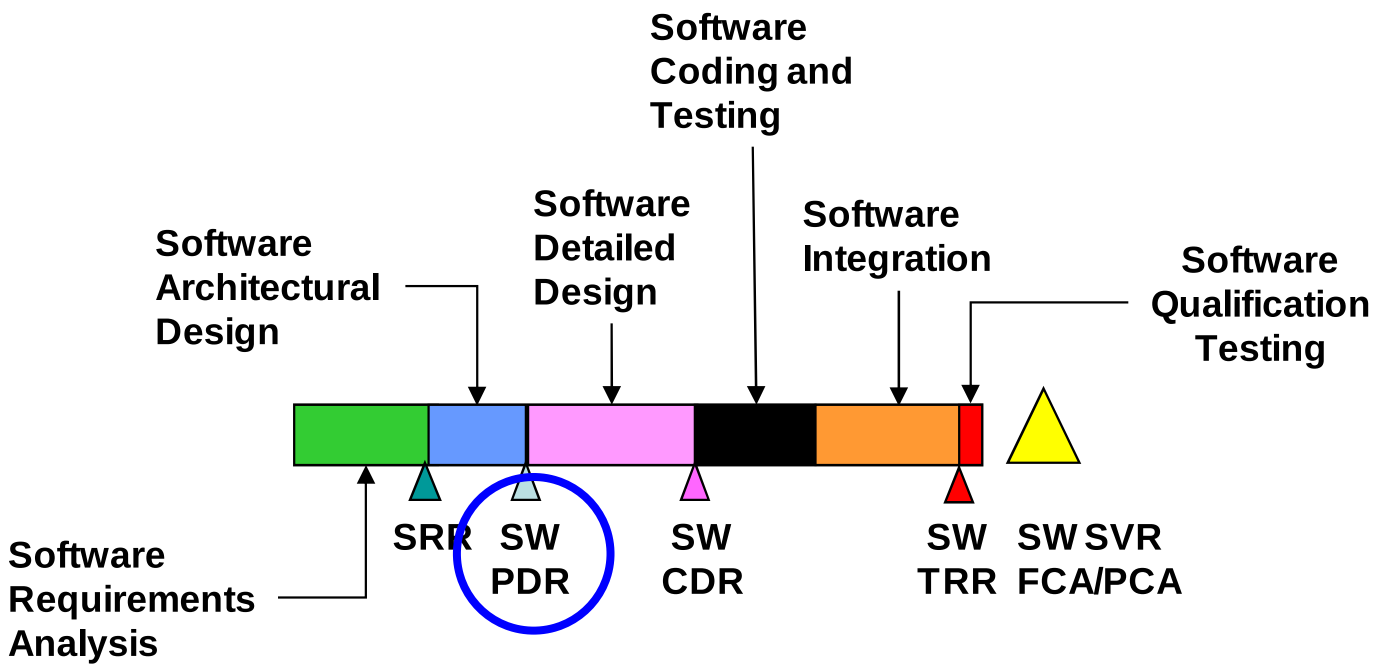

Software Coding and Testing
Software Detailed Design
Software Integration
Software Architectural Design
Software Qualification Testing
Software Requirements Analysis
SRR
SW PDR
SW CDR
SW TRR
SW SVR FCA/PCA

System Requirements Review (SRR) Outcomes

- Establishment of the System Assurance Case
  - Specification of the top-level system assurance claims that address identified threats to the mission.
  - Identification of the approach for developing the system assurance case.
- Identification of all critical elements to be protected
  - Identification of all relevant system assurance threats and their potential impact on critical system assets.
  - Identification of high-level potential weaknesses in the system
  - Determination and derivation of system assurance requirements (as a subset of the system requirements).
- Test and Evaluation Master Plan (TEMP) addressing system assurance
  - Examine the TEMP to ensure testing processes are sufficient for system assurance. **This may include planning for static code analysis**
- Support and Maintenance Concepts
  - Documentation of the support and maintenance concepts including a description of how assurance will be maintained.
  - **Description of what static code analysis tools will be used post deployment and how and when they will be applied**

Preliminary Design Review (PDR) Objectives

- The PDR is a multi-disciplined technical review to ensure that the system under review can proceed into detailed design, and can meet the stated performance requirements within cost (program budget), schedule (program schedule), risk, and specific assurance requirements and constraints.


Software Coding and Testing
Software Detailed Design
Software Integration
Software Architectural Design
Software Qualification Testing
SRR
SW PDR
SW CDR
SW TRR
SW SVR
SW FCA/PCA
Software Requirements Analysis

Preliminary Design Review (PDR) Outcomes – 1

- Information security technology evaluation of all critical COTS/GOTS elements
  - Performed as part of the analysis of alternatives.
  - Includes an updated assurance case based on the design, and new weaknesses and vulnerabilities identified.
  - **Results of static code analyses performed of GOTS/COTS components.**
    - **Which tools were used?**
    - **What weaknesses and vulnerabilities were discovered**
- **Specification of assurance-specific static analysis**
  - **Specification of assurance-specific static analysis and assurance-specific criteria to be examined during code reviews**
    - **Code reviews performed during implementation**
    - Documented in the System Engineering Plan (SEP) and Software Development Plan (SDP)
    - **Plan for training to use static analysis tools and for manual analysis**
- Configuration management
  - For Assurance, the preliminary configuration management plan must support traceability and protection of each configuration item, including requirements and architectural elements.
    - **At what stages of the configuration management process will static code analysis be applied?**
    - **What configuration change events will trigger code analysis?**
    - **What components will be analyzed?**
    - **How will the results of the analyses be documented?**

Preliminary Design Review (PDR) Outcomes – 2

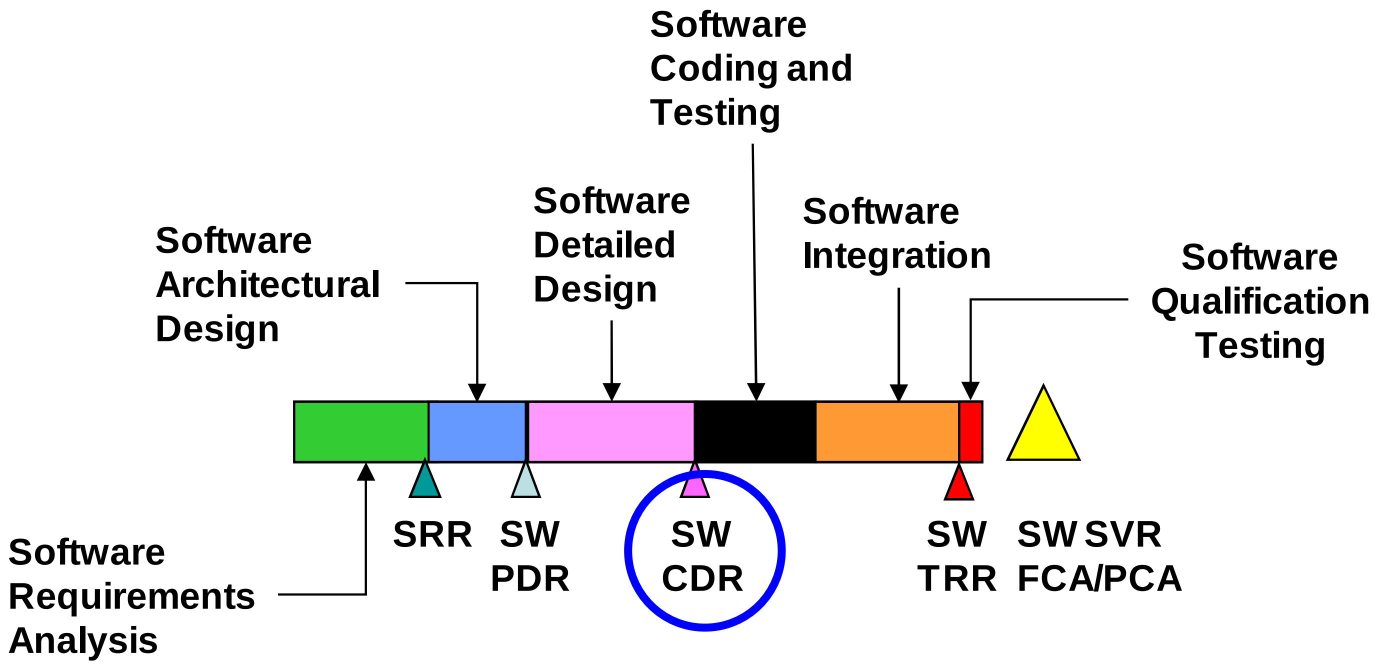
- Supply Chain Assurance
  - For all critical elements being considered for procurement, an analysis of the supplier and its processes should be performed
    - **Will the supplier perform static code analysis as part of its code development and/or code integration processes?**
    - **Which components will be analyzed? Which will not?**
    - **What tools do they plan to use?**
    - **What are the details of their code inspection process for manual security analysis?**
    - How will they mitigated any discovered vulnerabilities or weaknesses?
- Assurance Case
  - Updating of the assurance case with relevant evidence

Additional Preliminary Design Review (PDR) Considerations

- COTS source code is rarely available to the acquirer for independent code review
  - PMs should request COTS vendors provide Assurance Cases for their COTS products detailing both the vendor's secure coding practices and **the results of internal static code analysis** or third party assessment (e.g. Common Criteria certification)
  - In cases where such information is unavailable, and there is still a desire to use the COTS component, **the PM should consider binary code analysis**
  - Such analysis could be performed either as part of the system integrator's life cycle process, or independently by an IV&V agent
- **Ensure that a party other than the developer (such as a peer) will independently perform static analysis and test**, and that the element being reviewed will be the element that will be delivered.

Critical Design Review (CDR) Objectives

- The CDR is a multi-disciplined technical review to ensure that the system under review can proceed into system fabrication, demonstration, and test, and can meet the stated performance requirements within cost (program budget), schedule (program schedule), risk, and specific assurance requirements and constraints



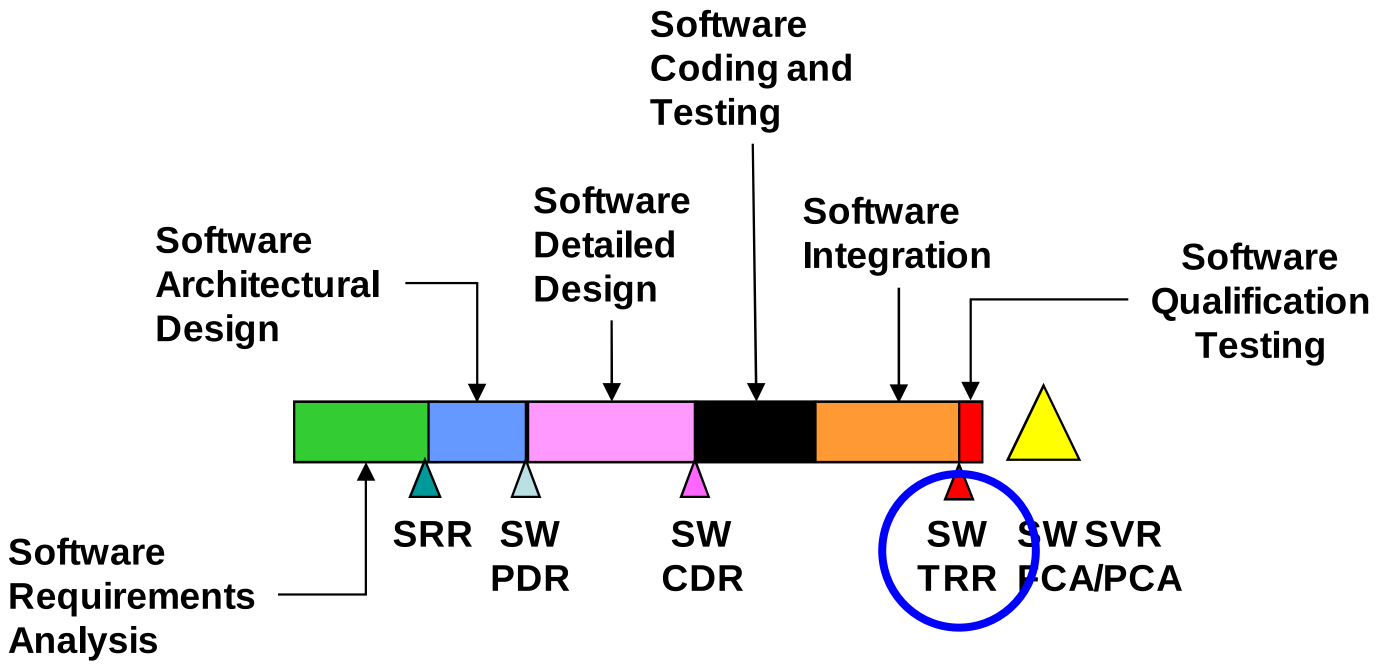
Software Coding and Testing
Software Detailed Design
Software Architectural Design
Software Integration
Software Qualification Testing
Software Requirements Analysis
SRR
SW PDR
SW CDR
SW TRR
SW FCA/PCA
SW SVR

Critical Design Review (CDR) Outcomes

- **Identification and use of selected source code analysis tools**
  - Selection of additional development tools and guidelines to counter weaknesses and vulnerabilities in the system elements and development environment(s)
    - ***These include static analysis tools for source code evaluation***.
  - **Definition and selection of assurance-specific static analyses** and assurance-specific criteria to be examined during peer reviews performed during implementation.
    - Documented in the SEP and Software Development Plan (SDP).
  - **Planning for training for assurance-unique static analysis tools** and peer reviews.
  - **Ensuring that another party (such as a peer) will independently perform static analysis and test**, and that the element being reviewed will be the element that will be delivered
    - This counteracts the risk of a developer intentionally subverting analysis and test, as well as aiding against unintentional errors.
- Assurance Case
  - Updating of the assurance case with relevant evidence.

Test Readiness Review (TRR) Objectives

- The TRR is a multi-disciplined technical review to ensure that the subsystem or system under review is ready to proceed into formal test

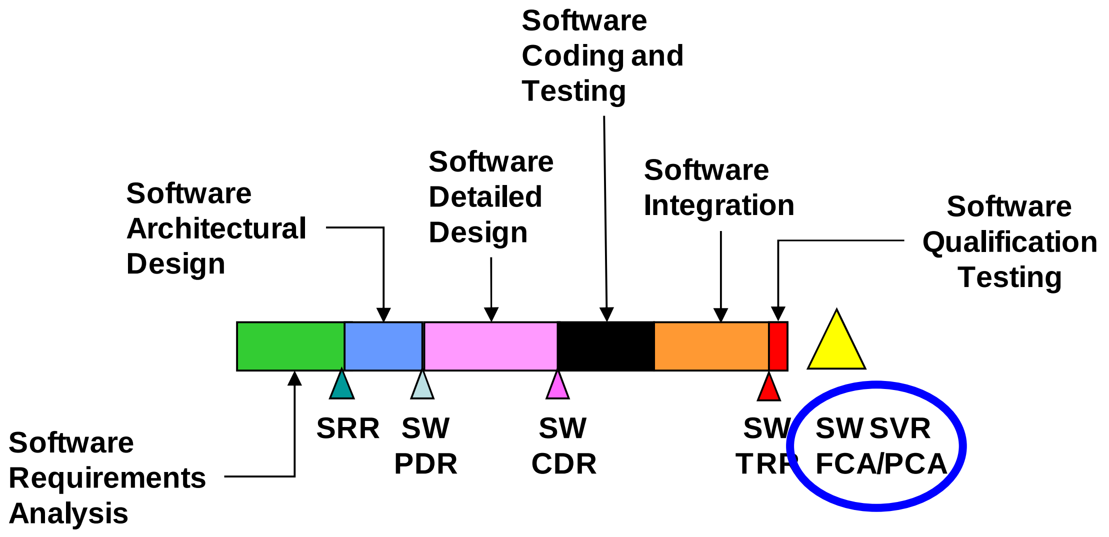

Software Coding and Testing
Software Detailed Design
Software Integration
Software Architectural Design
Software Qualification Testing
SRR
SW PDR
SW CDR
SW TRR
SW FCA/PCA
SVR
Software Requirements Analysis

Test Readiness Review (TRR) Outcomes

- **Verification regarding static code analysis**
  - **Verification that assurance-specific static analysis** and peer reviews of assurance criteria **have been completed**
  - **Verification that another party (such as a peer) performed static analysis and peer review**
  - **Selection of any additional static analysis tools to identify or verify weaknesses and vulnerabilities in the system elements and development environment(s)**
  - Completion and verification of an information security technology evaluation for all critical COTS/GOTS elements
- Open source verification
  - Identification of industry tools and test cases to be used for the testing of any binary or machine-executable open source software products with no warranty and no source code
  - **Documentation of evidence that static analysis has been performed (both source and binary) to identify weaknesses and vulnerabilities such as buffer overruns and cross-site scripting issues**
- Assurance Case
  - Updating of the assurance case with relevant evidence

System Verification Review SVR/Production Readiness Review (PRR) Objectives

- The SVR is a multi-disciplined product and process assessment to ensure that the system under review can proceed into low-rate initial production (LRIP) and full-rate production (FRP) within cost (program budget), schedule (program schedule), risk, and other system constraints
- The PRR examines a program to determine if the design is ready for production and if the producer has accomplished adequate production planning
- The primary difference between PRR and TRR is that the system test results are available prior to PRR
  - If changes are made to the system in response to test results, it will be necessary to revisit TRR tasks
  - Any evidence provided by system test results should be incorporated into the assurance case prior to PRR

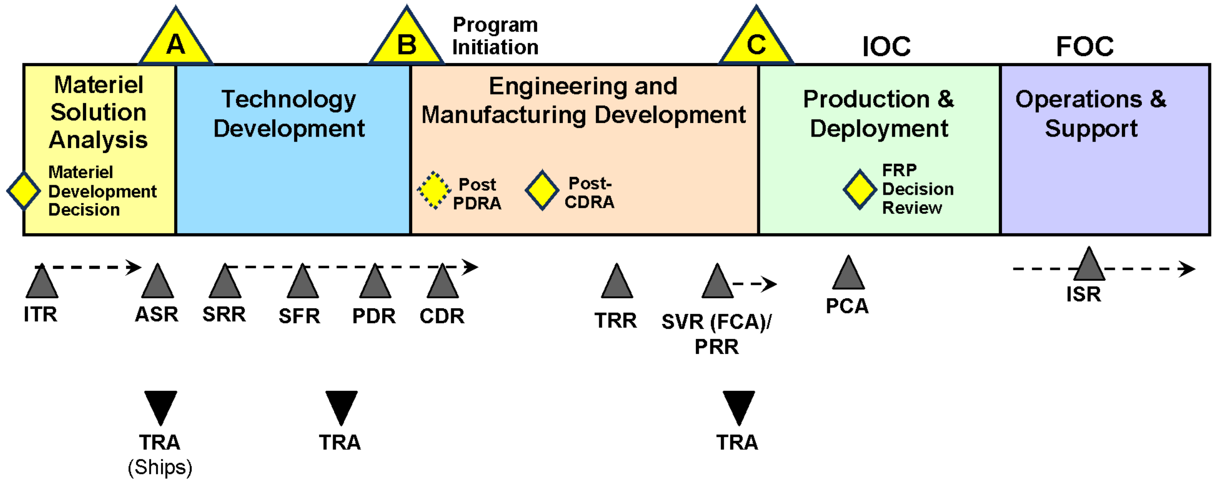

Software Coding and Testing
Software Detailed Design
Software Integration
Software Architectural Design
Software Qualification Testing
Software Requirements Analysis
SRR
SW PDR
SW CDR
SW TRR
SW SVR FCA/PCA

System Verification Review SVR/Production Readiness Review (PRR) Outcomes

- **Verification regarding static code analysis**
  - **Verification that assurance-specific static analysis** and peer reviews of **assurance criteria have been completed**
  - **Verification that another party (such as a peer) performed static analysis and peer review.**
  - **Selection of any additional static analysis tools to identify or verify weaknesses and vulnerabilities in the system elements and development environment(s)**
  - Completion and verification of an information security technology evaluation for all critical COTS/GOTS elements.
- Open source verification
  - Identification of industry tools and test cases to be used for the testing of any binary or machine-executable open source software products with no warranty and no source code.
  - **Documentation of evidence that static analysis has been performed (both source and binary) to identify weaknesses and vulnerabilities such as buffer overruns and cross-site scripting issues**
- Assurance Case
  - Updating of the assurance case with relevant evidence.

Operational Test Readiness Review (OTRR) Objectives

- The OTRR is a multi-disciplined product and process assessment to ensure that the “production configuration” system can proceed into Initial Operational Test and Evaluation with a high probability of successfully completing the operational testing
- Successful performance during operational test generally indicates that the system is suitable and effective for service introduction



- Initial Technical Review (ITR)
- Alternative Systems Review (ASR)
- Systems Requirements Review (SRR)
- System Functional Review (SFR)
- Preliminary Design Review (PDR)
- Critical Design Review (CDR)
- Post-PDR Assessment (Post-PDRA)

- Post-CDR Assessment (PCDRA)
- Test Readiness Review (TRR)
- System Verification Review (SVR)
- Functional Configuration Audit (FCA)
- Production Readiness Review (PDR)
- Operational Test Readiness Review (OTRR)
- Physical Configuration Audit (PCA)

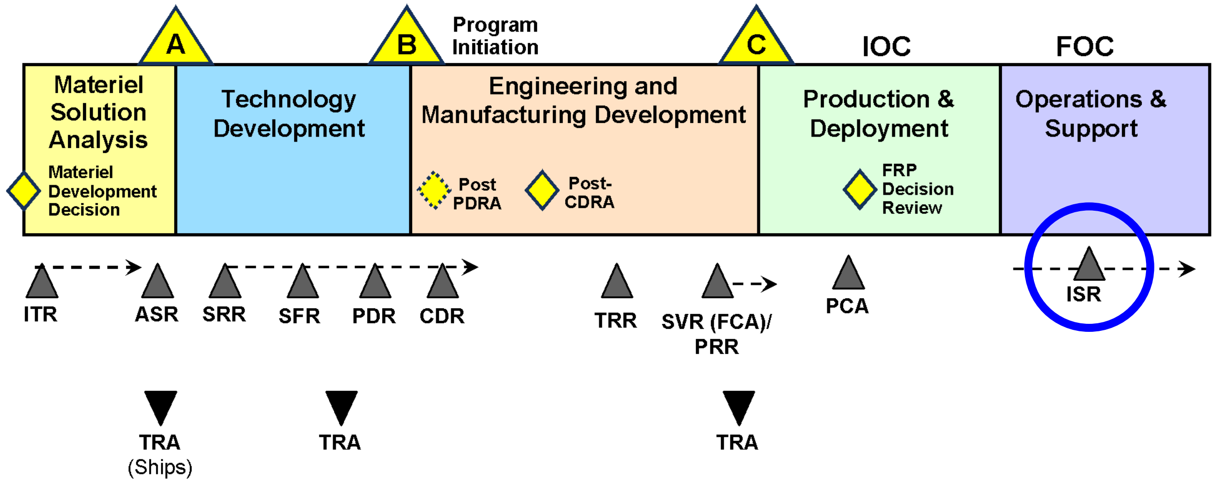
- Technology Readiness Assessment (TRA)
- In-Service Review (ISR)

Operational Test Readiness Review (OTRR)

- **Verification regarding static code analysis**
  - **Re-verification that assurance-specific static analysis** and peer reviews of assurance criteria **have been completed**.
    - **Source code static analysis is typically not performed again for OTRR, but binary analysis is performed, if appropriate.**
  - **Re-verification that another party (such as a peer) performed static analysis** and peer review.
  - Completion and verification of an information security technology evaluation for all critical COTS/GOTS elements.
- Weaknesses and vulnerabilities evaluation
  - **Documentation of evidence that the system has been analyzed for weakness and vulnerabilities using static (binary) analysis tools to identify such flaws as buffer overruns and cross-site scripting issues**
- Assurance Case
  - Updating of the assurance case with relevant evidence

In-Service Review (ISR) Objectives

- The ISR is a multi-disciplined product and process assessment to ensure that the system under review is operationally employed with well-understood and managed risk. This review is intended to characterize the in-service technical and operational health of the deployed system. It provides an assessment of risk, readiness, technical status, and trends in a measurable form



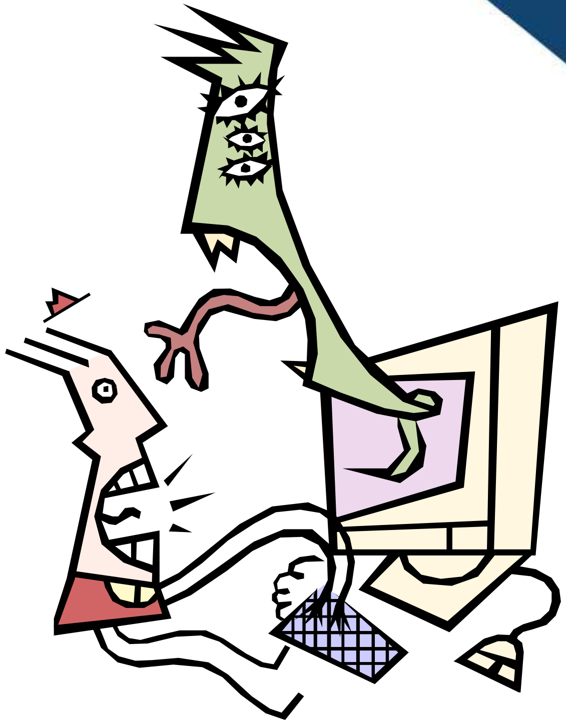
- Initial Technical Review (ITR)
- Alternative Systems Review (ASR)
- Systems Requirements Review (SRR)
- System Functional Review (SFR)
- Preliminary Design Review (PDR)
- Critical Design Review (CDR)
- Post-PDR Assessment (Post-PDRA)
- Post-CDR Assessment (PCDRA)
- Test Readiness Review (TRR)
- System Verification Review (SVR)
- Functional Configuration Audit (FCA)
- Production Readiness Review (PDR)
- Operational Test Readiness Review (OTRR)
- Physical Configuration Audit (PCA)
- Technology Readiness Assessment (TRA)
- In-Service Review (ISR)

In-Service Review (ISR) Outcomes

- Configuration Management
  - Review of the configuration management process, to determine that it remains adequate with respect to analysis of code changes, and being followed
- Weaknesses and vulnerabilities evaluation
  - **Documentation of evidence that any changes to the software throughout its service life have been analyzed for weakness and vulnerabilities using static (source or binary) analysis tools to identify such flaws as buffer overruns and cross-site scripting issues**
- Assurance Case
  - Updating of the assurance case with relevant evidence

NORTH AMERICAN
PUBLIC SECTOR

Challenges to Effective Static Code Analysis



Challenge – Procurement and Maintenance of Tools

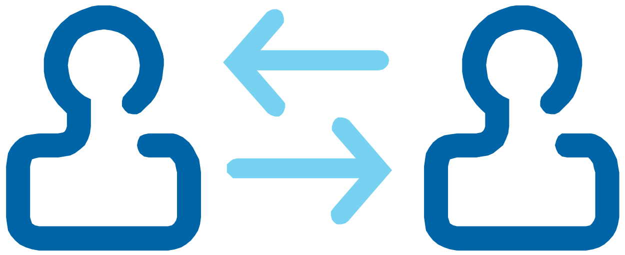
- The better static code analysis tools are expensive
  - Use multiple tools used to offset tool biases and minimize false positives and false negatives can quickly become cost prohibitive for a single program
  - In addition, maintenance agreements to ensure a tool is up to date with respect to the spectrum of threats, weaknesses, and vulnerabilities add long term costs
- Buy it once, use it often provides the most bang for the buck
- Pooled-resources analysis labs may make economic sense.

Challenge – Training

- Static code analysis is not for sissies, although it may be for CISSPs (Certified Information System Security Professionals)
  - This tongue-in-cheek statement belies the difficulty in using static code analysis tools to their best advantage
  - Chandra, Chess, and Steven [12] point out that when static code analysis tools are employed by a trained team of code analysts, false positives are less of a concern; the analysts become skilled with the tools very quickly; and greater overall audit capacity results.
- In order to determine the validity of static code analysis results, it is important for PMs to understand
  - The level of training that code analysts have had with the tools employed for static code analysis
  - Their understanding of code weaknesses and vulnerabilities

Useful Links

- NIST SAMATE Static Analysis Tool Survey
  - The National Institutes for Science and Technology (NIST), Software Assurance Metrics and Tool Evaluation (SAMATE) project, provides tables describing current static code analysis tools for source, byte, and binary code analysis
  - More information on SAMATE can be found at **http://samate.nist.gov/**
- DHS Build Security In Web Site
  - A wealth of software and information assurance information, including white papers on static code analysis tools
  - More information on Build Security In can be found at **https://buildsecurityin.us-cert.gov/daisy/bsi/home.html**



NIST SAMATE – Source Code Analysis Tools
http://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html

Tool	Language(s)	Avail.	Finds or Checks for	------Date------
C++test .TEST Jtest	C++ C#, VB.NET, MC++ Java	Parasoft	"defects, poor constructs, potentially malicious code and other elements"	4 Apr 2006
cadvise	C, C++	HP	many lint-like checks plus memory leak, potential null pointer dereference, tainted data for file paths, and many others	11 Mar 2009
CodeCenter	C	CenterLine Systems	incorrect pointer values, illegal array indices, bad function arguments, type mismatches, and uninitialized variables	28 Oct 2005
CodeScan	ASP Classic, PHP, ASP.Net	CodeScan Labs	specialise in inspecting web source code for security holes and source code issues.	14 Jul 2008
CodeSecure	PHP, Java (ASP.NET soon)	Armorize Technologies	XSS, SQL Injection, Command Injection, tainted data flow, etc.	16 Mar 2007
K7	C, C++, and Java	Klocwork	Access problems, buffer overflow, injection flaws, insecure storage, unvalidated input, etc.	6 July 2005
Ounce	C, C++, Java, JSP, ASP.NET, VB.NET, C#	Ounce Labs	coding errors, security vulnerabilities, design flaws, policy violations and offers remediation	19 Apr 2007
PLSQLScanner 2008	PLSQL	Red-Database-Security	SQL Injection, hardcoded passwords, Cross-site scripting (XSS), etc.	23 Jun 2008
PolySpace	Ada, C, C++	PolySpace Technologies	run-time errors, unreachable code	25 Feb 2005
PREfix and PREfast	C, C++	Microsoft proprietary		10 Feb 2006
Prevent	C, C++	Coverity	flaws and security vulnerabilities - reduces false positives while minimizing the likelihood of false negatives.	11 Mar 2005

NIST SAMATE – Byte Code Analysis Tools http://samate.nist.gov/index.php/Byte_Code_Scanners.html

Tool	Lan-guage	Avail.	Finds or Checks for	Date
AspectCheck	Java and .NET applications, including ASP.NET, C#, and VB.NET	Aspect Security proprietary	security critical calls	24 Nov 2004
FindBugs™	Java class files	free	null pointer deferences, synchronization errors, vulnerabilities to malicious code, etc. It can be linked to Java source code to highlight the problem in the source.	23 June 2005
FxCop	.NET managed code assemblies	free	checks for conformance to the Microsoft .NET Framework Design Guidelines: more than 200 defects in: Library design, Globalization, Naming conventions, Performance, Interoperability and portability, Security, and Usage.	16 May 2008
Gendarme	.NET Applications	free	extensible rule-based tool to find problems in .NET applications and libraries.	30 Oct 2008
Moonwalker	.NET Applications	free	find deadlocks and assertion violations in .NET programs	14 Nov 2008
Smokey	.NET or Mono assemblies	jesjo...@mindspring.com	correctness, design, security, performance and other rules	13 Nov 2008
SoftCheck Inspector	Java	SofCheck	creates assertions for each module, tries to prove the system obeys assertions and the absence of runtime errors.	8 Jun 2006
XSSDetect BETA	compiled managed assemblies (C#, Visual Basic .NET, J#)	free	Visual Studio plugin to help find Cross-Site Scripting vulnerabilities (CWE 79). Ignores paths with proper encoding or filtering	10 Jul 2008

NIST SAMATE – Binary Code Analysis Tools http://samate.nist.gov/index.php/Binary_Code_Scanners.html

Tool	Lan-guage	Avail.	Finds or Checks for	- - Date - -
BugScam	app binaries .EXE or .DLL files	SourceForge	This a package of IDC scripts for IDA Pro to look for common programming flaws.	8 May 2003
CodeSurfer/x86	x86 executables	Grammatech	A prototype system from joint research by the University of Wisconsin and GrammaTech to provide a platform for an analyst to understand the workings of COTS components, plugins, mobile code, and DLLs, as well as memory snapshots. CodeSurfer is a source code anaylzer.	2005
IDA Pro	Window/Linux excutables	DataRescue	A disassembler/debugger that can be used to analyze security issues in binary code.	31 Jan 2008
Logiscan	J2EE, MIPS and SPARC binaries, as well as existing Intel x86 support	LogicLab	Weaknesses such as buffer overflows, SQL injection and cross-site scripting can be discovered . It also offers suggestions for appropriate security remediation via its built-in training for secure coding. Formerly BugScan.	2005
SecurityReview	Excutable of *C, C++, C#, JAVA	Veracode	Automated static binary and dynamic web application analyses to identify software flaws and vulnerabilities, absence of security features, and malcode including backdoors and other unintended functionality. SecurityReview is a security testing service provided by Veracode.	24 May 2007
Vine	x86 executables	BitBlaze	Vine is a component of UC Berkeley s research project BitBlaze. It provides an intermediate language (ILA) that x86 code can be translated to. It also provides analysis on the ILA, such as abstract interpretation, dependency analysis, and logical analysis via interfaces with theorem provers.	20 Jan 2008
CAT.NET	x86 executables	Microsoft	A binary code analysis tool that helps identify common variants of certain prevailing vulnerabilities that can give rise to common attack vectors such as Cross-Site Scripting (XSS), SQL Injection and XPath Injection.	30 Dec 2009

References

[1] Jones, Capers. Overview of the United States Software Industry Results Circa 2008, June 20, 2008.

[2] Reifer, D, and Bryant, E. Software Assurance in COTS and Open Source Packages, Proceedings of the DHS Software Assurance Forum, October 14-16, 2008.

[3] DoD Chief Information Officer (CIO) Memorandum, Clarifying Guidance Regarding Open Source Software (OSS) in the Department of Defense (DoD), October 16, 2009

[4] Black, P. Static Analyzers in Software Engineering, CrossTalk, The Journal of Defense Software Engineering, pp. 16-17, March-April 2009.

[5] McGraw, G. Automated Code Review Tools for Security, Computer, vol. 41, no. 12, pp. 108-111, Dec. 2008.

[6] Howard, M. Mitigate Security Risks by Minimizing the Code You Expose to Untrusted Users, http://msdn.microsoft.com/msdnmag/issues/04/11/AttackSurface, November, 2004.

[7] Manadhata, P., Tan, K, Maxion, R, and Wing, J. An Approach to Measuring a System's Attack Surface, CMU-CS-07-146, Carnegie Mellon University, August 2007.

[8] Howard, M. A Process for Performing Security Code Reviews, IEEE Security & Privacy, pp. 74-79, July-August 2006.

[9] ISO/IEC/IEEE CD 15026-2.3, Systems and software engineering — Systems and software assurance — Part 2: Assurance case, February 12, 2009.

[10] DoDI 5000.02, Operation of the Defense Acquisition System, December 8, 2008.

[11] Program Executive Office (PEO) Integrated Warfare Systems (IWS) Technical Review Manual (TRM) (Draft), Department of the Navy, Naval Sea Systems Command, Program Executive Office, Integrated Warfare Systems, December 2008.

[12] Chandra, P., Chess, B., and Steven, J. Putting the Tools to Work: How to Succeed with Source Code Analysis, IEEE Security & Privacy, pp. 80-83, May-June 2006.

For More Information . . .

Paul R. Croll
CSC
10721 Combs Drive
King George, VA 22485-5824

Phone: +1 540.644.6224

Fax: +1 540.663.0276

e-mail: pcroll@csc.com