

# Practical Software and Systems Measurement Continuous Iterative Development Measurement Framework

Part 3: Software Assurance and Technical Debt

Version 2.1  
April 15, 2021

## Developed and Published by Members of:

Practical Software &  
Systems Measurement



Product No.  
PSM-2021-03-001

National Defense Industrial  
Association



International Council on  
Systems Engineering



Product No.  
INCOSE-TP-2020-001-06

## Editors:

**Cheryl L. Jones**

US Army

[cheryl.l.jones128.civ@mail.mil](mailto:cheryl.l.jones128.civ@mail.mil)

**Bill Golaz**

Lockheed Martin

[william.h.golaz@lmco.com](mailto:william.h.golaz@lmco.com)

**Geoff Draper**

L3Harris Technologies

[geoff.draper@l3harris.com](mailto:geoff.draper@l3harris.com)

**Paul Janusz**

US Army

[paul.e.janusz.civ@mail.mil](mailto:paul.e.janusz.civ@mail.mil)

Unclassified: Distribution Statement A: Approved for Public Release; Distribution is Unlimited



PSM Product Number: PSM-2020-06-001

INCOSE Product Number: INCOSE-TP-2020-001-06

## Copyright Notice:

For this document, each of the collaborative organizations listed on the cover page is the sole manager of their products and services and are the only parties authorized to modify them. Since this is a collaborative product, modifications are managed through the participation of all parties.

General Use: Permission to reproduce, use this document or parts thereof, and to prepare derivative works from this document is granted, with attribution to PSM, NDIA, and INCOSE, and the original author(s), provided this copyright notice is included with all reproductions and derivative works.

Supplemental Materials: Additional materials may be added for tailoring or supplemental purposes if the material developed separately is clearly indicated. A courtesy copy of additional materials shall be forwarded to PSM ([psm@psmsc.com](mailto:psm@psmsc.com), attention: Cheryl Jones). The supplemental materials will remain the property of the author(s) and will not be distributed, but will be coordinated with the other collaboration parties.

Author Use: Authors have full rights to use their contributions with credit to the technical source.

Supplemental Notice from INCOSE: This work is an Affiliate Product per INCOSE Policy TEC-107 INCOSE Technical Product Development & Commercialization (26 October 2018). It is a technical product developed outside the INCOSE product development process and was made by INCOSE members in cooperation with PSM and NDIA; then approved by INCOSE to be distributed from INCOSE central channels. The authors own the copyright and take primary responsibility for proper branding, intellectual property, content quality and appropriate citations with INCOSE oversight based on this policy & related procedure.



## CONTENTS

<b>EXECUTIVE SUMMARY</b> .....	<b>1</b>
CONTRIBUTORS .....	2
<b>10. SOFTWARE ASSURANCE</b> .....	<b>3</b>
10.1 SOFTWARE ASSURANCE TERMINOLOGY.....	4
10.2 IMPLEMENTATION CONSIDERATIONS FOR ASSURING SECURE RESILIENT PRODUCTS .....	5
10.3 SOFTWARE ASSURANCE MEASUREMENT .....	6
10.3.1 <i>Software Assurance Measures</i> .....	8
<b>11. TECHNICAL DEBT</b> .....	<b>10</b>
11.1 TECHNICAL DEBT TERMINOLOGY .....	10
11.2 INFORMATION NEEDS.....	11
11.3 MEASURES FOR TECHNICAL DEBT .....	11
11.4 APPLYING THE PSM CID MEASUREMENT FRAMEWORK TO MANAGE TECHNICAL DEBT .....	12
11.5 TOOLS/METHODS .....	14
<b>12. ICM TABLE</b> .....	<b>15</b>
<b>BIBLIOGRAPHY</b> .....	<b>23</b>

## LIST OF FIGURES

Figure 1: Alignment of PSM Software Assurance Measures with ISO/IEC 25000 standards for quality characteristics.....	7
--	---

## LIST OF TABLES

Table 1: PSM CID Measurement Framework Editors.....	2
Table 2: Part 3 Core Team Contributors and their Organization.....	2
Table 3: Software Assurance Terms and Definitions .....	5
Table 4: Recommendations for Initial Software Assurance Measures .....	9
Table 5: Applying PSM CID Measures to Manage Technical Debt .....	12
Table 6: Software Assurance Issues, Categories, and Measures .....	15



## EXECUTIVE SUMMARY

This report provides recommendations for the measurement of continuous iterative developments (CID). It includes a Practical Software and Systems Measurement (PSM) CID measurement framework detailing common information needs and measures that are effective for evaluating CID approaches. The information needs address the team, product, and enterprise perspectives to provide insight and drive decision-making. The framework also identifies and specifies an initial set of measures that have been identified as being practical measures to address these information needs.

This guidance is intended to be used by team, program, and enterprise personnel who are implementing CID approaches, as a reference for common, practical measures that can be utilized. The measures a program or enterprise chooses to implement and collect will be tailored based on alignment with its information needs and objectives, so they may differ from those described here. The measures presented are intended to be tailored and adapted to the development approach and environment.

Version 1.05 detailed potential information needs and measures that are common to CID approaches, and an initial set of ten measurement specifications that were prioritized by user surveys as highest value. This Version 2.1 includes added material that has been researched and developed by the CID working group. The new materials include information on measuring:

- Product value (Part 2, section 8.11)
- Enterprise measurement (Part 2, section 9)
- Software assurance (Part 3, section 10)
- Technical debt (Part 3, section 11)

Part 1 of this report includes a series of diagrams and an ontology to describe the development approaches and terminology used. It also includes an “Information Category-Measurable Concept-Measures” (ICM) Table detailing potential information needs and measures for CID developments. Additional potential measures will be added in future releases, as described in Section 6, Next Steps.

For the highest priority measures, sample measurement specifications have been developed that detail the identified measures. These are included in a separate Part 2 of the paper, along with a discussion of how to use these measures for enterprise decision making. This addendum, Part 3 of the paper, separately extends the main CID paper with information and guidance on Software Assurance and Technical Debt.

We invite your comments on this material, and your participation in future updates addressing additional measures and guidance.

This report is intended to be methodology and approach-agnostic and is written so that it may be adapted to organizational needs. Different methodologies and tools may use different terminology than defined in this report.



## CONTRIBUTORS

**Table 1: PSM CID Measurement Framework Editors**

<b>Editors</b>	<b>Organization</b>
Cheryl Jones	Army Futures Command – CCDC Armament Center
Geoff Draper	L3Harris Technologies / NDIA Systems Engineering Division
Bill Golaz	Lockheed Martin Corporation
Paul Janusz	Army Futures Command – CCDC Armament Center

**Table 2: Part 3 Core Team Contributors and their Organization**

<b>Core Team</b>	<b>Organization</b>
Mark Cornwell	OUSD R&E
Holly Dunlap	Raytheon Technologies
William Hayes	Software Engineering Institute
Ronda Henning	L3Harris Technologies
Stephen Henry	Defense Acquisition University (retired)
Joe Jarzombek	Synopsys
Jason McDonald	L3Harris Technologies
William J. Nichols	Software Engineering Institute
Cory Ocker	Raytheon Technologies
Carmela Rice	OUSD (A&S)
David Rosenfeld	L3Harris Technologies
Larri Rosser	Raytheon Technologies
Forrest Shull	Software Engineering Institute
Robin Yeman	Lockheed Martin
Carol Woody	Software Engineering Institute / CERT



## 10. SOFTWARE ASSURANCE

The rapid delivery of secure, resilient systems that meet mission needs is a business and national security imperative. The timely mitigation and resolution of security vulnerabilities and weaknesses is a business-critical concern that affects the system security posture and the speed and cycle time at which new capabilities are deployed and securely maintained. Department of Defense (DoD) systems are software-intensive, pushing software assurance into a key role in systems' system security posture. Software Assurance is the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner. Because software systems may consist of open source, Commercial Off the Shelf (COTS) products, and unique applications combined to address mission requirements, software assurance must be addressed throughout the entire system development and acquisition lifecycle. This method of system development introduces security concerns that must be addressed throughout the system product life cycle, in both development and operations (as emphasized in DevSecOps).

As part of the DoD Adaptive Acquisition Framework (AAF), multiple DoD functional policies drive software assurance requirements to Program Managers, Science & Technology (S&T) managers, and systems engineers. DoD Instruction (DoDI) 5000.90 requires program managers to address cybersecurity responsibilities from the earliest exploratory phase throughout all stages of the acquisition. Potential breaches and their consequences must be documented in the Cyber Security Strategy annex to the Program Protection Plan and must include network, enabling systems, and supply chain risks. Additionally, per DoDI 5000.83 Technology and Program Protection, software assurance methods and practices, a critical part of program protection in design, test, manufacture and sustainment, ensures that systems function as intended, mitigating risks associated with known and exploitable software vulnerabilities to provide a level of assurance commensurate with technology, program, system and mission objectives. It also directs the establishment of Technology Area Protection Plans (TAPP) for each S&T modernization priority area to reduce compromise or loss of critical technologies and protect against unwanted technology transfer. Key elements of TAPPs include:

- Critical technology areas and technical threshold levels to protect
- Contract, grant, and cooperative agreement clauses
- Protection efforts for contractor, contracts, and universities researchers to focus on
- Thresholds for international collaboration and sales.

Measures such as those described here help programs manage the implementation and effectiveness of their Cyber Security Strategy across the program life cycle. Data from the TAPP informs all of the Government protection effort for critical DoD technologies.

The integration of software assurance measurement into the development and acquisition life cycle is emphasized in the Defense Innovation Board (DIB) Software Acquisition and Practice (SWAP) study and new DoD Adaptive Acquisition Framework policies. The strategies, visionary concepts, and best practices relevant to successfully achieving software assurance measurement include:

- Create and use automatically generated, continuously available measures that emphasize speed, cycle time, security, and code quality.



- Make security a first-order consideration for all software-intensive systems, recognizing that security-at-the-perimeter is not enough.
- Create, implement, support, and require a fully automatable approach to test and evaluation, including security, which allows high-confidence distribution of software to the field on an iterative basis.
- Shift from certification of executables, to certification of code, to certification of development, integration, and deployment toolchain, with the goal of enabling rapid fielding of mission-critical code at high levels of information assurance.
- Establish and maintain a digital infrastructure within each Service or Agency that enables rapid deployment of secure software to the field and incentivize its use by contractors.

The DIB SWAP study also recommended measures for software development, including these measures related to system security:

- Time to field high priority functions (specification -> operations) or fix newly found security hole (finding -> operations)
- Time required for full regression test (automated) and cybersecurity audit/penetration testing
- Structure of the code base (software architecture).

These strategies can only be achieved by integrating security considerations early in the entire engineering life cycle, starting with system concepts, systems engineering, architecture and design. These enablers are described in the Information Needs and Potential Measures described in the ICM table in section 12, a summary of which includes:

- Identifying and resolving software vulnerabilities, weaknesses and defects from system deliveries
- Characterizing the size (lines of code) and trends of software (use of open source, agile development, etc.) relative to the system attack surface
- Ensuring adequate planning for resources and execution of verification and validation of software assurance requirements

These security measurement concepts are further described in the following sections:

- *Software Assurance Terminology* – introduction to key terms, concepts and descriptions used as a basis for this document
- *Implementation Considerations for Assuring Secure Resilient Products* – best practices for design and development practices and tools to help assure an appropriate security posture in deliverable products and systems
- *Software Assurance Measurement* – recommendations on a prioritized set of consensus measures for software assurance

## 10.1 SOFTWARE ASSURANCE TERMINOLOGY

The following terms, concepts and definitions are applicable for the software assurance practices and measures in this document.



**Table 3: Software Assurance Terms and Definitions**

Term	Description
<b>Software Assurance</b>	Software Assurance (SwA) is the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner [CNSS 4009]. This ideal of no exploitable vulnerabilities is usually unachievable in practice, so programs must perform risk management to reduce the probability and impact of vulnerabilities and related weaknesses to acceptable levels.  <a href="https://www.cnss.gov/CNSS/issuances/Instructions.cfm">https://www.cnss.gov/CNSS/issuances/Instructions.cfm</a>
<b>Common Weakness Enumeration (CWE)</b>	Common Weakness Enumeration (CWE™) is a community-developed list of software and hardware weakness types. It serves as a common language, a measuring stick for security tools, and as a baseline for weakness identification, mitigation, and prevention effort.  <a href="https://cwe.mitre.org">https://cwe.mitre.org</a>
<b>Common Vulnerabilities and Exposures (CVE)</b>	Common Vulnerabilities and Exposures (CVE®) is a list of entries – each containing an identification number, a description, and at least one public reference – for publicly known cybersecurity vulnerabilities. CVE Entries are used in numerous cybersecurity products and services from around the world, including the U.S. National Vulnerability Database (NVD).  <a href="https://cve.mitre.org/">https://cve.mitre.org/</a>
<b>Common Attack Pattern Enumeration and Classification System (CAPEC)</b>	Common Attack Pattern Enumeration and Classification (CAPEC™) is a community resource for identifying and understanding attacks. Understanding how the adversary operates is essential to cyber security. CAPEC helps by providing a comprehensive dictionary of known patterns of attack employed by adversaries to exploit known weaknesses in cyber-enabled capabilities. It can be used by analysts, developers, testers, and educators to advance community understanding and enhance defenses.  <a href="https://capec.mitre.org/">https://capec.mitre.org/</a>
<b>Common Weakness Scoring System (CWSS)</b>	The Common Weakness Scoring System (CWSS™) provides a mechanism for prioritizing software weaknesses in a consistent, flexible, open manner. It is a collaborative, community-based effort that is addressing the needs of its stakeholders across government, academia, and industry.  <a href="https://cwe.mitre.org/cwss/">https://cwe.mitre.org/cwss/</a>
<b>Common Vulnerability Scoring System (CVSS)</b>	The Common Vulnerability Scoring System (CVSS) provides a way to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity. The numerical score can then be translated into a qualitative representation (such as low, medium, high, and critical) to help organizations properly assess and prioritize their vulnerability management processes.  <a href="https://first.org/cvss/">https://first.org/cvss/</a>

## 10.2 IMPLEMENTATION CONSIDERATIONS FOR ASSURING SECURE RESILIENT PRODUCTS

Software assurance is an integral part of software quality. Research has shown that reduced defects will also reduce security weaknesses. Defects and security weaknesses are introduced throughout design and development requiring active identification and removal efforts.

Considerable research and experience have led to well-documented best practices for assuring the delivery of secure resilient products, spanning the development cycle. These software





assurance-related topics are beyond the scope of this measurement-focused document but are mentioned as a key element of assuring product quality and secure systems. Examples of proven security best practices include:

- Security as a key component integrated into the systems engineering process across the life cycle, including early requirements, architecture, and design activities. Effective software assurance must be planned and monitored throughout the systems engineering process to ensure desired operational outcomes: it cannot not be added after the fact.
- Establishing an organizational culture and emphasis on software assurance, quality, and proven best practices.
- Adopting secure coding guidelines and practices. Train, deploy, and institutionalize their consistent use across the organization as an integral part of design, development, and verification processes.
- Treating security weaknesses as quality defects. Identify, prioritize, and remediate software assurance defects like other software deficiencies.
- Secure coding practices as part of code reviews and success criteria.
- Incorporating static and dynamic analysis testing tools into the development process, with regular automated scans of the codebase to identify security weaknesses and vulnerabilities.
- Incorporating software composition analysis to create a bill of materials.
- Ensuring testing of all components, including open source software and external libraries, on which applications rely for operations.
- Executing security test cases as part of the automated verification suite integrated into the software development pipeline and toolchain.

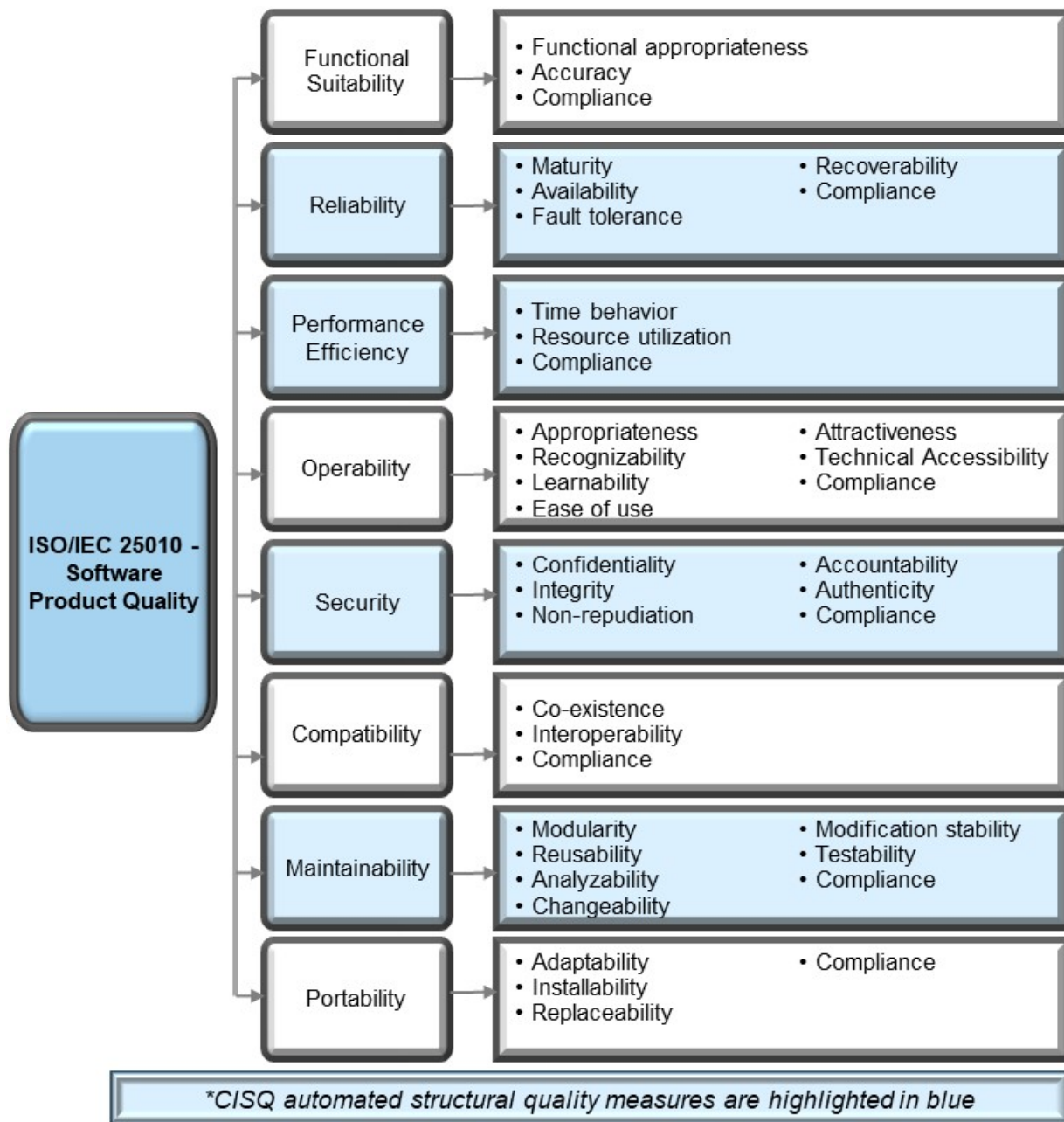
There are many tools and methods for evaluating different aspects of software assurance, software structure, and code quality. Tools and methods for increasing the assurance of software and security quality attributes include static and dynamic code analysis tools, defect or configuration management systems, test suites, and secure development environments. Since each tool focuses on only a perspective of potential weaknesses and vulnerabilities, plans must include the use of multiple tools for adequate coverage.

### 10.3 SOFTWARE ASSURANCE MEASUREMENT

The prior sections set the context for definition of objectives and measures to manage the assured integrity of software products. The ICM table in section 12 describes information needs and potential measures for software assurance. These fall generally in the following areas:

- Measures about identifying, mitigating and resolving security vulnerabilities and weaknesses in developed or non-developmental (reused) software
- Measures about managing security defects and technical debt relative to the system software attack surface
- Measures of effectiveness for security controls and testing
- Measures assessing the effectiveness of program protection planning and compliance with the mission-critical Risk Management Framework (RMF)
- Measures of performance on conducting timely security audit/penetration testing and obtaining Interim Authorization to Test (IATT) and/or Authorization to Operate (ATO)
- Measures of timeliness on recovering from system compromise to a full security posture.

ISO/IEC 25010 software quality includes eight software quality attributes with multiple options for measurement. As noted in **Figure 1**, security measures can address confidentiality, integrity, non-repudiation, accountability, authenticity, and compliance. The Consortium for Information & Software Quality™ (CISQ™) has developed a standard for the automated static analysis measurement of Technical Debt that is designed to predict corrective maintenance costs and related factors to guide IT decisions and resource allocations. Similarly, the Software Engineering Institute has an extensive collection of best practices for software assurance.



**Figure 1: Alignment of PSM Software Assurance Measures with ISO/IEC 25000 standards for quality characteristics**



Other qualities such as reliability, maintainability, and performance efficiency align to several of these same qualities and several measures are already in place that can be leveraged. Existing measures related to defect tracking, test coverage, problem reports and burndown rates, meantime to restore, minimum viable product (MVP) and minimum viable capability release (MVCR) can be useful if applied to security defects, security tests, security problems, and security requirements. With the implementation of appropriate tagging of security related outputs across the lifecycle (requirements, defect reports, change requests, etc.), a wide range of existing measures can be refocused to support tracking of the schedule and progress of security development, product quality (including security), and several aspects of process performance as related to software assurance.

Research has shown that an estimated 5% of defects are vulnerabilities (Woody et al, 2014). Even without specific security designations for defect reports and change requests, this provides a high-level estimate to help management gauge the level of security risk that should be expected as the product matures. Measures of defect resolution can use this same relationship to estimate product security improvement.

With an effective tagging of security related requirements (including those related to supply chain risk management, insider threat, infrastructure and recovery) and complete traceability as these flow into architecture, detail design, coding, testing, and implementation the monitoring of product security can be implemented at each step of the lifecycle. By relating requirements to the appropriate qualities and tracing them as they flow to components and the development pipeline, verification of the end product for a critical quality such as security can be greatly facilitated.

At a minimum, the tracking of known (n-day) weaknesses and vulnerabilities from identification (typically as the output of static and dynamic analysis tools), prioritization and remediation (typically using an existing scoring system), and determination of those remaining in the implemented product should be closely monitored. Additionally, the tracking of the time it takes for unknown (0-day) weaknesses or vulnerabilities from identification (usually after the system has been exploited) to mitigation should be closely monitored. This tracking will show the level of resources applied to security and the remaining risk that the operational environment will need to address. This can be augmented with third party product information from the National Vulnerability Database (NVD). Also, suppliers should already be tracking this information and contracts should be updated to request this level of reporting. Guidance on contracting for software assurance in DOD contracts is provided in the whitepaper “Incorporating Software Assurance into Department of Defense Acquisition Contracts” produced by the DoD SwA Community of Practice Working Group. This can all be handled through existing defect reporting with the addition of appropriate data tagging.

### 10.3.1 Software Assurance Measures

The intent of the descriptions of software assurance and measurement in this PSM CID document is to:

- Increase awareness of security and assurance-related issues across disciplines so they can be considered early in development life cycle and integrated into system architecture, design, and implementation processes
- Identify potential measures to identify, mitigate and manage software vulnerabilities and weaknesses to assure the security and integrity of deliverable software products



The ICM table in section 12 identifies potential measures for software assurance aligned with product and enterprise information needs. Of these, the PSM working group recommends prioritizing initial software assurance measures as described in Table 4.

**Table 4: Recommendations for Initial Software Assurance Measures**

Measurement Concepts	Recommended Software Assurance Measures (Initial Priorities)
Identification and resolution of vulnerabilities and weaknesses	<ul style="list-style-type: none"> <li>• Identification of vulnerabilities (CVEs) and weaknesses (CWEs)</li> <li>• Resolution of CVEs and CWEs</li> <li>• Patches delivered to burn down and close vulnerabilities</li> </ul>
Security defect tracking	<ul style="list-style-type: none"> <li>• Counts of security defects (open, closed)</li> <li>• Security defect attributes (e.g., severity, criticality)</li> <li>• Security defect containment (saves vs. escapes)</li> </ul>
Quality and security testing coverage	<ul style="list-style-type: none"> <li>• Percentage of code base screened for vulnerabilities and weaknesses (developed code and non-developmental items)</li> <li>• Security test coverage (code base, security controls)</li> <li>• Security test case status (passed, failed)</li> <li>• Coverage and trends in size of the attack surface</li> </ul>

Pending community review and feedback, specifications and indicators for prioritized software assurance measures will be developed and published in future iterations of this guidance.

Many of the other CID measures in this document are also relevant and useful when applied to software assurance in an overall system context, such as:

- *Cycle Time* – turnaround time for releasing security patches to resolve vulnerabilities
- *MTTR / MTTD* – mean time to restore (mitigate) and detect security defects
- *Automated Test Coverage* – extent of capability and code covered by security test cases
- *Defect Detection* – containment of security defects and minimizing escapes to operations
- *Defect Resolution* – the extent of identified security defects that have been resolved
- *Burndown* – progress toward completing resolution of security vulnerabilities
- *Release Frequency* – how often security patches and releases are deployed

The PSM working group and security subteam welcomes feedback on this initial set of measures to advance the state of the practice for software assurance, and experiences from applying them.



## 11. TECHNICAL DEBT

One of the characteristics of continuous iterative development (CID) is its ability to implement new capabilities and release them to one's users at a much faster pace than traditional development. Program managers recognize that doing so involves making informed decisions and trade-offs amongst inter-related quality, cost, and performance components/objectives. If not managed properly, these trade-offs can come at a significant cost or "debt", which may have consequences in the future. One of the goals of the new DoD Adaptive Acquisition Framework (AAF) policy is to deploy software faster, but not at the expense of compromising quality (including security). To that end, the policy also requires organizations to measure and manage their "technical debt".

### 11.1 TECHNICAL DEBT TERMINOLOGY

Technical debt consists of design or implementation constructs that are expedient in the short term, but that set up a technical context that can make a future change more costly or impossible. Technical debt may result from having implementation issues related to architecture, design, structure, duplication, test coverage, comments and documentation, potential defects, complexity, or coding practices. The metaphor of "debt" communicates that technical debt items incur extra costs for the program in the future, in the form of increased costs of change during lifecycle evolution and sustainment. Addressing technical debt requires effort to be spent on activities to improve the quality of the software architecture, code, documentation, etc., or otherwise "pay down" the debt to the system, beyond just adding new capability.

#### Discussion of Definition

There is considerable academic research and industry practice focused on addressing technical debt, but no consistent definitions or measures are in widespread use, due in part to differing goals. Some in the system and software engineering community take a somewhat narrow definition that restricts the domain of technical debt to sub-optimal design decisions. This approach primarily affects maintainability issues such as changeability and scalability, but often excludes areas such as missing features, functional defects, or most structural flaws. In contrast, other definitions of technical debt tend toward broader measures that aim to include the future costs of corrective maintenance and other software quality-related outcomes.

For our purposes in this report, technical debt will be considered from three primary sources:

- Debt from weaknesses and vulnerabilities in code constructs
- Debt from design and architecture flaws
- Debt from missing information items such as documentation shortfalls, missing information, IP issues.

System security risk is a form of technical debt, considered when determining program plans and mitigation priorities. Software flaws, weaknesses, and vulnerabilities should be considered defects, adjudicated and resourced on an equal footing relative to other program priorities. Defects are categorized using standard registries and taxonomies. Security-related defects are worked off during the development process for subsequent system releases.



Some definitions of technical debt also include mission debt such as functional deficiencies, required capabilities or features that should have already been implemented but are missing (distinguished from a backlog of features not yet prioritized for implementation), or missing functionality or performance issues in COTS for a COTS-intensive system. However, mission debt is not included as a part of technical debt as defined in this paper.

A buildup of technical debt items may make it more difficult to continue to add more features in a timely way or otherwise prevent the team from dealing with necessary enhancements. While the definition discusses technical debt that is "expedient in the short term", not all technical debt arises from conscious decisions. Some technical debt is known; it is the result of risks accepted, or business decisions made to ensure progress towards near term goals. Other technical debt may be inadvertent and not known or identified until later. The real question is how a program can effectively identify and track all these needs together. With only finite resources available, a program must balance resolving defects and mission debt, implementing new capabilities, and addressing technical debt. One possible strategy for addressing the burndown and reducing the system's overall technical debt may be to leverage existing plans for upgrades or related tasks; e.g., a supplier could fix a legacy static analysis finding the next time that code file is touched. Another strategy would be to have iterations specifically dedicated to the burndown of technical debt items. This strategy would be up to the teams to decide which course of action is best.

When discussing technical debt, it may be important to distinguish between root cause of technical debt (e.g. problematic code constructs, code not tested, lack of peer reviews), counts of technical debt (e.g. comments, CVEs), and the impacts of technical debt (e.g. how hard is it to make changes, cost/effort/time to resolve technical debt).

## 11.2 INFORMATION NEEDS

Information needs related to technical debt include:

- How easy/difficult is it to update or refactor the design and code?
- Can the system architecture be expanded as the system continues to be developed and revised?
- When does it become too costly or take too long to maintain the design or architecture?
- How many defects are identified as technical debt (versus mission debt)?
- Are there areas of code that have a high frequency of defects?
- Is the documentation current, sufficient for user needs, and sustainable throughout the lifecycle?
- When should identified technical debt be resolved, parts of the system replaced, or a new system started?
- What is the impact of this technical debt? Is it worth the investment and schedule to resolve it?

## 11.3 MEASURES FOR TECHNICAL DEBT

Measures addressing the breadth of Technical Debt is worthy of a separate dedicated effort; indeed, many are addressed by extensive research literature referenced herein. Notably, these issues apply broadly to development domains well beyond just continuous iterative development, which is the emphasis of this document. The PSM working group has chosen to describe how the

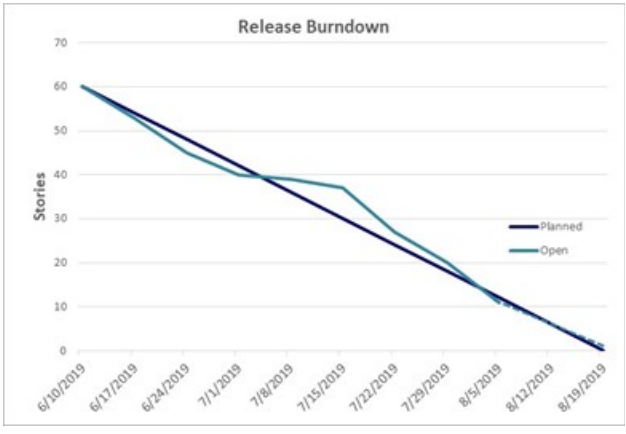



CID measures described in Part 2, Section 8 of this framework can be applied to identify and mitigate certain factors of Technical Debt.

## 11.4 APPLYING THE PSM CID MEASUREMENT FRAMEWORK TO MANAGE TECHNICAL DEBT

Technical debt is not measured with "one" measure but will likely utilize several measures. Some of the example measures in the PSM CID Measurement Framework can be used for multiple purposes, including technical debt. As part of this working group effort, other practical and newer measures were identified to provide feedback on technical debt specific information needs. These are identified in the ICM Table in Part 1 section 7, described in the measurement specifications in Part 2 section 8, and some examples discussed in Table 5 below.

**Table 5: Applying PSM CID Measures to Manage Technical Debt**

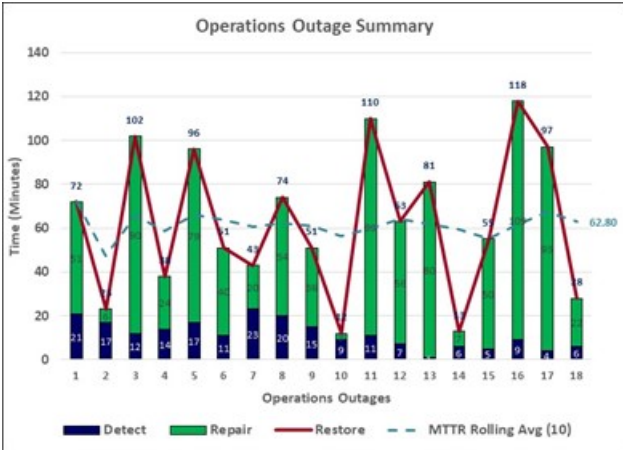
PSM Measure	Example Indicator Example Applicability to Managing Technical Debt
Burndown Chart (8.2)	 <p>Planned band shows the plan rate of closing of defects that are related to technical debt, while open band shows how many identified technical debt defects remain open. Used to manage the closure of identified technical debt through development. Requires that technical debt is identified in defect tracking system.</p>
Committed vs. Completed (8.3)	 <p>If committed stories outpace completed stories, there will be technical debt if resourcing or scheduling is not adjusted.</p>



PSM Measure	Example Indicator Example Applicability to Managing Technical Debt	
<p>Cumulative Flow (8.4)</p>		<p>Top band of CFD shows rate of arrival of new work that has been committed to. Height shows the depth of the queue of work To Do. A widening band show work commitments that are not being worked off at the same rate as arrivals and represents a potential growth in technical debt.</p> <p>Watch for increase in technical debt represented by growth "in progress" or "to do" queues, which indicates backlogs and work not being completed as committed.</p>
<p>Defect Resolution (8.7)</p>		<p>For technical or management reasons, some detected defects may be knowingly accepted with resolution deferred to a future iteration or release. It may be a reasoned cost vs. benefit decision to defer resolution of known defects. This is one kind of technical debt.</p> <p>Open defects (the difference between defects resolved and defects detected) may be technical or mission debt. These defects are on the backlog and may not yet be allocated to an increment or release. Escapes to later iterations/releases indicative of rework that could be more costly. Failure to detect defects is hidden future rework (technical debt).</p>





PSM Measure	Example Indicator Example Applicability to Managing Technical Debt
Mean Time to Restore (MTTR) / Mean Time to Detect (MTTD) (8.8)	<div style="display: flex; align-items: center;">  <div style="margin-left: 20px;"> <p>Operational services can be impacted if defects escape from development to operations. MTTR (restore) and MTTD (detect) are indicators of how quickly full service can be restored.</p> <p>Availability and reliability of deployed services, and impacts to operations, are key factors in assessing</p> </div> </div> <p>technical debt.</p>

## 11.5 TOOLS/METHODS

Automated tools may be required to collect the data for the measures. There are many tools and methods for measuring and evaluating different aspects of technical debt including:

- Code/Product- include static and dynamic code analysis tools that check for compliance with coding standards, code scanning tools that check for vulnerabilities and weaknesses, and code quality tools that track and classify defects, test suites monitor test coverage, etc.
- Development Processes - DevSecOps or Continuous Integration / Continuous Delivery (CI/CD) pipelines for automated analysis of newly introduced code and evaluation of newly identified CWE/CVE
- Design – MBSE tools, engineering tools, structural quality tools that evaluate architecture maintainability, cohesion, coupling, flexibility, scalability, etc.
- Missing information – tools that help with configuration management systems, development environments, technical manuals

A few of the more widely used tools and methods are noted in the references in the bibliography.

Before selection by a program, any tool should be evaluated based on the needs of the program and the strength of the tool. Needs should include the information needs to be addressed, and the measures selected to address those information needs.

Adopters of the PSM CID framework are encouraged to seek additional guidance on identifying and managing Technical Debt by leveraging resources from CISQ, Object Management Group (OMG), INCOSE, the Software Engineering Institute (SEI), and other researchers. Additionally, indicators of Technical Debt applicable to traditional development can also be applied in CID environments, such as those contained in these measurement references:

- Practical Software and Systems Measurement: A Foundation for Objective Management, v. 4.0. <http://www.psmc.com/PSMGuide.asp>
- Systems Engineering Leading Indicators Guide, Version 2.0. INCOSE-TP-2005-001-03

# PSM Continuous Iterative Development Measurement Framework - Part 3

Developed and Published by Members of:



**12. ICM TABLE** Part 1 contains an ICM table for continuous iterative development. The information needs and measures in this section (Table 6) are specific to software assurance and technical debt.

**Table 6: Software Assurance Issues, Categories, and Measures**

Row	Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures**	Notes	Category*
1	Schedule and Progress Product Quality	Work Unit Progress Security	Are patches delivered as committed?	Are known (n-day) vulnerabilities and weaknesses resolved as committed? Are previously unknown (o-day) vulnerabilities being mitigated after being identified?	What features / capabilities remain vulnerable and are unresolved?	<b>Patches Delivered</b> <b>Vulnerabilities, Weaknesses Resolved</b> <b>Features/Capabilities Resolved</b> <b>Burndown of Vulnerabilities, Weaknesses</b> <b>Time from vulnerability identification to mitigation</b>		SwA-High
2	Schedule and Progress	Work Backlog		How many software assurance defects contribute to technical debt? Mission debt? How many software assurance defects are going to be resolved in the next release? Future releases?		Software Assurance Defects Unresolved Backlog Completed iterations in release	Criticality is key -- Requires a process to address zero day vulnerabilities and rapid remediation. Program would have to designate defects as software assurance.	SwA-Medium
3	Schedule & Progress	Work Backlog	How many iterations does it take to resolve outstanding technical debt actions?	How many releases/months does it take to resolve outstanding technical debt actions?	How many external releases/months does it take to resolve outstanding technical debt actions?	Cycle Time Aging of Tasks Defect Resolution, Defect Lag Time Mean Time to Restore (MTTR) <b>Technical Debt Actions (Written, Committed, Completed)</b> <b>Burndown of Technical Debt Items from Backlog</b>	Source: Design, Product, or Info Items Programs would have to designate these items as technical debt.	TD-High

# PSM Continuous Iterative Development Measurement Framework - Part 3

Developed and Published by Members of:



Row	Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures**	Notes	Category*
4	Resources and Cost	Financial Performance	What is the level of effort needed to address critical weaknesses prior to release?	What is the level of effort needed to contain and recover from a realized risk or vulnerability (to an acceptable level of risk)?	What is the level of effort needed to contain and recover from a realized risk or vulnerability (to an acceptable level of risk)?	Cost to Fix Vulnerabilities	Specific to one realized risk or vulnerability, or a set. Need to include financials to link it to program risk.	SwA-Medium
5	Resources & Cost [Schedule & Progress Process Performance]	Facilities and Support Resources		When does it become too costly or take too long to evolve or maintain the architecture, design, or component? When should a replacement system or component be considered? Is the product taking on acceptable risk?	When does it become too costly or take too long to evolve or maintain the architecture, design, or component? Is this system or component incompatible for use or obsolete? Is there another system or component in the portfolio that can do the job better? As an enterprise, are we taking on acceptable risk?	Cost and Schedule (Time) to Resolve Technical Debt Actions Cost of Delay Rework Cost/Effort Total Lifecycle Cost, Total Ownership Cost Replacement Cost and Schedule (Time) Risk Burndown	Source: Design, Product	TD-Medium
6	Resources & Cost [Schedule & Progress Product Quality Customer Satisfaction]	Facilities and Support Resources		What is the impact of this technical debt? Is it worth the investment and schedule (time) to resolve it?	What is the impact of this technical debt? Is it worth the investment and schedule (time) to resolve it?	Technical Debt Costs Replacement Costs Total Lifecycle Costs, Total Ownership Costs		TD-Medium

# PSM Continuous Iterative Development Measurement Framework - Part 3

Developed and Published by Members of:



Row	Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures**	Notes	Category*
7	Product Quality	Functional Correctness	How many software assurance defects have been identified and adjudicated? How many new software assurance defects have been identified since the last assessment?	How many software assurance defects have been identified and adjudicated? How many new software assurance defects have been identified since the last assessment? How big/what is the size of the system's attack surface? Is the attack surface increasing, decreasing, or staying the same?	How big/what is the size of the system's attack surface? Is the attack surface increasing, decreasing, or staying the same?	<b>Common Vulnerabilities Enumeration (CVEs)</b> <b>Common Weaknesses Exposure (CWEs)</b> Software Assurance Vulnerabilities and weaknesses Detected / Resolved Software Assurance Defects Detected / Resolved Trend of Software Assurance Defects Detected / Resolved over time Size of Attack Surface Defect Density (Defects / size)	Programs would need to define what constitutes a SwA defect and categorize defects. Defects and vulnerabilities may be categorized by priority, criticality, design, product, info items, etc. Existing vulnerabilities and weaknesses need to be fixed to reduce the attack surface. Focus also on resolving vulnerabilities and weaknesses earlier in the lifecycle.	SwA-High
8	Product Quality	Security		What impacts to system performance and/or integrity will incur by refactoring or replacing components?	How vulnerable is the system to attack? How many vulnerabilities and weaknesses have been mitigated?	Percentage of Code Base Available for Screening Percentage of Code Base Screened for Vulnerabilities and Weaknesses Percentage of Code Requiring Binary Analysis (no source code available)	Activity: Testing of Reuse/Supply Chain. The code base would include code from legacy, 3rd party, open source, subcontractors, and COTS.	SwA-High

# PSM Continuous Iterative Development Measurement Framework - Part 3

Developed and Published by Members of:



Row	Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures**	Notes	Category*
9	Product Quality	Security	What percentage of code from suppliers (legacy, 3rd party, subcontractors, COTS) is screened for vulnerabilities and weaknesses?	<p>What percentage of code from suppliers (legacy, 3rd party, subcontractors, COTS) is screened for vulnerabilities and weaknesses?</p> <p>What is the quality / vulnerability / supportability of legacy and third party code?</p> <p>Did my system inherit a vulnerability from another system?</p> <p>How many vulnerabilities and weaknesses were inherited from COTS?</p> <p>How many have been mitigated? How many have been reported to the National Vulnerability Database (NVD)?</p> <p>How secure is the product?</p>	<p>What percentage of the code from suppliers (legacy, 3rd party, subcontractors, COTS) is screened for vulnerabilities and weaknesses?</p> <p>What is the quality / vulnerability / supportability of legacy and third party code?</p> <p>Did my system inherit a vulnerability from another system?</p>	Vulnerabilities and Weaknesses Inherited from COTS	<p>Analysis of the COTS would include looking at the age of the COTS the last update.</p> <p>Older versions may be unsupported and need to be replaced. Newer versions have to go through the accreditation process.</p> <p>Address and identify the vulnerabilities and weaknesses -- isolate or sandbox.</p>	SwA-High

# PSM Continuous Iterative Development Measurement Framework - Part 3

Developed and Published by Members of:



Row	Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures**	Notes	Category*
10	Product Quality	Security	<p>What percentage of software assurance controls are covered by testing?</p> <p>How many of the vulnerabilities have been identified in testing to determine the extent of impacted areas?</p> <p>Is a patch working sufficiently for the time being?</p>	<p>What percentage of software assurance controls are covered by testing?</p> <p>How many relevant attack patterns have been covered by test cases?</p> <p>Has the system been sufficiently tested for software assurance vulnerabilities and weaknesses?</p>		<p>Code Test Coverage (impacted areas, patched areas)</p> <p>Automated SwA Test Coverage</p> <p>Vulnerabilities, Attack Pattern Test Coverage</p> <p>Test Cases Developed, Verified per Attack Pattern</p> <p>Failed Tests Due to Vulnerabilities</p> <p>Misuse/Abuse Cases</p> <p>Opportunities</p> <p><b>Vulnerabilities that are Identified in Testing, but not Fixed in Coding</b></p>	<p>Activity: Testing Code - Have the software assurance controls been tested?</p> <p>Activity: Testing Vulnerabilities and Weaknesses by phase</p> <p>Analysis would include assessing whether vulnerabilities are causing downtime or denial of service, or result in the data being compromised.</p>	SwA-Medium
11	Product Quality	Security	<p>Are we preventing vulnerabilities from releasing to operation?</p>	<p>Is the attack surface increasing, decreasing, or staying the same?</p>	<p>How much do these unpatched vulnerabilities contribute to the overall system risk or software assurance posture?</p> <p>What is the risk to the system mission if this product is released with these weaknesses and vulnerabilities?</p>	<p>Trends in size of attack surface</p> <p>Likely Impact of Residual Weaknesses &amp; Vulnerabilities</p> <p>Trends in findings of code scans over time from testing to fielding</p>	<p>Analysis would include assessing whether the vulnerabilities were causing downtime or denial of service, or resulted in the data being compromised.</p>	SwA-Medium

## PSM Continuous Iterative Development Measurement Framework - Part 3

Developed and Published by Members of:



Row	Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures**	Notes	Category*
12	Product Quality	Dependability - Reliability	Is there a need to design the software to be modular, replaceable, or proprietary/open source?	Can the system architecture be expanded as the system continues to be developed and revised? How easy/difficult is it to update or refactor the design and product? When do obsolete components need to be replaced?		Cohesion Coupling Design and/or Code Complexity Interfaces Affected Effort/Cost to De-Couple or Refactor System	Source: Design, Product	TD-Medium
13	Product Quality	Dependability - Reliability		Is the documentation sufficient for user needs and for sustainability? Is the technical data package complete and current?	Is the documentation sufficient for user needs and for sustainability? Is the technical data package complete and current?	Documentation Actions or Defects on Documents Needed for Technical Data Package (TDP)	Source: Information	TD-Medium
14	Product Quality Process Performance	Security Process Effectiveness	What Risk Management Framework (RMF) Controls need to be implemented/adhered to?	What RMF Controls need to be implemented/adhered to? What is the (overall) compliance with the mission-critical Risk Management Framework (RMF) controls established for a program?	What is the (overall) compliance with the mission-critical Risk Management Framework (RMF) controls established for a program?	RMF Controls		SwA-Medium

## PSM Continuous Iterative Development Measurement Framework - Part 3

Developed and Published by Members of:



Row	Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures**	Notes	Category*
15	Product Quality Process Performance	Security Process Effectiveness		How quickly can a software assurance event or vulnerability be detected? (Monitor, Detect) How quickly can the team respond to a software assurance event? (Resolve, Deploy) How well has the system been designed to recover?	How rapid can the system recover to a known, secure state after an attack (Resiliency)? Is the system cyber-resilient? (Remove, recover)	Mean Time to Restore (MTTR) Mean Time to Detect (MTTD) Response Time Time to Patch Vulnerability Software assurance Vulnerability Lead or Cycle time	Also a schedule and quality issue. Dependent on identified vulnerabilities (COTS issues or built-in). May be dependent on release process (e.g. assessments & authorizations).	SwA-Medium
16	Product Quality	Security			Is the program protection planning adequate? Is there a software assurance strategy that maps to the Program Protection Plan?	Vulnerabilities Covered by Program Protection Vulnerabilities Removed Prior to Testing Code Passing Peer Review		SwA-Medium
17	Product Quality [Customer Satisfaction Resources & Cost]	Security		How much technical debt does the system have? What will it take to remove this technical debt? How is technical debt prioritized?	How much technical debt does the enterprise have? What will it take to remove this technical debt?	Technical Debt Actions (Written, Committed, Completed) Effort/Cost to Resolve Technical Debt (Plan, Actual)		TD-High
18	Product Quality [Process Performance]	Security	How many defects are identified as technical debt?	How many defects are identified as technical debt?		Technical Debt Defects Coding Standard Violations	Program would have to designate defects as technical debt.	TD-Medium
19	Process Performance Product Quality	Process Efficiency Security		How long does it take to successfully complete software assurance audit/penetration testing? How much of the SwA testing is automated?	How long does it take to successfully complete software assurance audit/penetration testing? How much of the SwA testing is automated?	<b>Software Assurance Test Duration</b> Automated SwA Test Coverage	Critical path, time required for full regression testing/audit. Relates to continuous ATO process.	SwA-High



## PSM Continuous Iterative Development Measurement Framework - Part 3

Developed and Published by Members of:



Row	Information Categories	Measurable Concept	Team Information Need	Product Information Need	Enterprise Information Need	Potential Measures**	Notes	Category*
20	Process Performance [Schedule and Progress]	Process Efficiency	How often has the baseline changed? Is the baseline stable?	How long does it take to get an authorization (IATT/ATO) for new releases? How long does it take to prepare the authorization Package? Is the Time to authorization quick enough to meet the criteria of a Continuous ATO? How many critical software assurance defects are holding up/present a roadblock to the authorization process?	How long does it take to get an authorization (IATT/ATO) for new releases? How fast can the system deploy new secure capabilities to the user? Can the system be released (Go/No Go Decision)?	<b>Time to Authorization (IATT/ATO)</b> Time to Prepare the authorization Package Authorization (IATT/ATO) Status Frequency of Baseline Changes Unresolved Critical Software Assurance Defects	Use of a hardened DoD cloud may speed authorizations for deployment. Time to authorization in relationship to the release date. Time to Prepare the authorization package may be broken out by acquirer, supplier, and joint efforts.	SwA-High

\*\* Measures in **Bold** were identified as High Priority and will be addressed in the next release of this paper.

\* Category includes Sw Assurance (SwA) or Technical Debt (Tech Debt) and Priority



## BIBLIOGRAPHY

### Policy and Study Board Reports

Department of Defense. *DOD Instruction 5000.87, Operation of the Software Acquisition Pathway*. (2020, October 2)  
[https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500087p.PDF?ver=virAfQj4v\\_LgN1JxpB\\_dpA%3D%3D](https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500087p.PDF?ver=virAfQj4v_LgN1JxpB_dpA%3D%3D)

Defense Innovation Board (DIB), *Software Is Never Done: Refactoring the Acquisition Code for Competitive Advantage*, 2019, Software Acquisition and Practices (SWAP)  
[https://media.defense.gov/2019/Apr/30/2002124828/-1/-1/0/SOFTWAREISNEVERDONE\\_REFACTORINGTHEACQUISITIONCODEFORCOMPETITIVEADVANTAGE\\_FINAL.SWAP.REPORT.PDF](https://media.defense.gov/2019/Apr/30/2002124828/-1/-1/0/SOFTWAREISNEVERDONE_REFACTORINGTHEACQUISITIONCODEFORCOMPETITIVEADVANTAGE_FINAL.SWAP.REPORT.PDF)

Defense Science Board (DSB), *Design and Acquisition of Software for Defense Systems*, Defense Science Board (DSB) Task Force on Design and Acquisition of Software for Defense Systems, 2018  
[https://dsb.cto.mil/reports/2010s/DSB\\_SWA\\_Report\\_FINALdelivered2-21-2018.pdf](https://dsb.cto.mil/reports/2010s/DSB_SWA_Report_FINALdelivered2-21-2018.pdf)

### General

John McGarry (Author), D. C. (2001). *Practical Software Measurement: Objective Information for Decision Makers*. Addison-Wesley Professional.

Vacanti, D. S. (2015). *Actionable Agile Metrics for Predictability: An Introduction*. Daniel S. Vacanti, Inc.

### Software Assurance

ISO/IEC 25010, Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and software Quality Models:

<https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>

Dr. William R. Nichols, J. D. (2018). *DoD Developer's Guidebook for Software Assurance*. Retrieved from Software Engineering Institute:  
[https://resources.sei.cmu.edu/asset\\_files/SpecialReport/2018\\_003\\_001\\_538761.pdf](https://resources.sei.cmu.edu/asset_files/SpecialReport/2018_003_001_538761.pdf)

K. Nidiffer, C. W. (2018). *Program Manager's Guidebook for Software Assurance*. Retrieved from Software Engineering Institute: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=538771>

MITRE. (n.d.). *Common Attack Pattern Enumeration and Classification*. Retrieved from CAPEC: <https://capec.mitre.org/>

Mitre. (n.d.). *Common Weakness Scoring System (CWSS™)*. Retrieved from Common Weakness Enumeration: [https://cwe.mitre.org/cwss/cwss\\_v1.0.1.html](https://cwe.mitre.org/cwss/cwss_v1.0.1.html)

NVD. (n.d.). *Common Vulnerabilities and Exposures*. Retrieved from CVE: <https://cve.mitre.org/>



- NVD. (n.d.). *Common Vulnerability Scoring System (CVSS)*. Retrieved from National Vulnerability Database: <https://nvd.nist.gov/vuln-metrics/cvss>
- SEI. (n.d.). *Predicting Software Assurance Using Quality and Reliability Measures*. Retrieved from Software Engineering Institute: <https://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=428589>
- Woody, E. R. (n.d.). *Exploring the Use of Metrics for Software Assurance*. Retrieved from Software Engineering Institute: <https://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=540881>
- Marien, John R et. al. (2017). Department of Defense (DoD) Software Assurance (SwA) Community of Practice (CoP) Contract Language Working Group Working Paper. Retrieved from Department of Defense Research and Engineering Enterprise: <https://rt.cto.mil/wp-content/uploads/2019/06/Incorporating-SwA-Contracts-2017-11-15.pdf>

## Technical Debt

- Managing Technical Debt: Reducing Friction in Software Development by Philippe Kruchten, Robert Nord, Ipek Ozkaya, ISBN-13: 978-0135645932, ISBN-10: 013564593X
- An OMG ® Automated Technical Debt Measure Publication Automated Technical Debt Measure. (2018).  
<https://www.omg.org/spec/ATDM/NormativeMachineConsumableFiles>  
<https://www.omg.org/spec/ATDM/20170303/AutomatedTechnicalDebtMeasure.xmi>