Carnegie Mellon University
Software Engineering Institute

Exploring the Use of Metrics for Software Assurance

Carol Woody, Ph.D.
Robert Ellison, Ph.D.
Charlie Ryan

December 2018

TECHNICAL NOTE
CMU/SEI-2018-TN-004

CERT Division

http://www.sei.cmu.edu

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

DM18-1366

Table of Contents

List of Figures

List of Tables

Acknowledgments

The development of this report was sponsored by the Department of Defense Cruise Missile Defense Systems (CMDS) Project Office. We thank James Wessel and Eileen Wrubel from the Carnegie Mellon University Software Engineering Institute (SEI) for coordinating the funding that allowed us to assemble and publish our research in this area and for their input as reviewers.

In addition, we thank William Richard Nichols and Timothy Chick for their support and insights as reviewers. Also, we thank Barbara White and Sandy Shrum for providing editing support.

Abstract

The Software Assurance Framework (SAF) is a collection of cybersecurity practices that programs can apply across the acquisition lifecycle and supply chain. The SAF can be used to assess an acquisition program's current cybersecurity practices and chart a course for improvement, ultimately reducing the cybersecurity risk of deployed software-reliant systems.

This report proposes measurements for each SAF practice that a program can select to monitor and manage the progress it's making toward software assurance. Metrics are needed to determine how effectively a practice is performed and how well software assurance is addressed. This report presents an approach for determining which SAF practices should be measured and how. It provides acquirers, program managers, and contractors with an approach for using metrics to establish confidence that the systems they plan to field will have sufficient software assurance.

1 Introduction to Software Assurance

There is always uncertainty about a software system's behavior. Rather than performing exactly the same steps repeatedly, most software components function in a highly complex networked and interconnected system of systems that changes constantly. Measuring the design and implementation yields confidence that the delivered system will behave as specified. Determining that level of confidence is the objective of software assurance, which is defined by the Committee on National Security Systems (CNSS 2010)[1] as

> *Implementing software with a level of confidence that the software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software, throughout the lifecycle.*

Measuring the software assurance of a product as it is developed and delivered to function in a specific system context involves assembling carefully chosen metrics that demonstrate a range of behaviors to confirm confidence that the product functions as intended and is free of vulnerabilities. Measuring software assurance is challenging, since it is a complex and difficult problem with no readily available solutions.

The first challenge is evaluating whether a product's assembled requirements define the appropriate behavior. The second challenge is to confirm that the completed product, as built, fully satisfies the specifications for use under realistic conditions.

Determining assurance for the second challenge is an incremental process applied across the lifecycle. There are many lifecycle approaches, but, in a broad sense, some form of requirements, design, construction, and test is performed to define what is wanted, enable its construction, and confirm its completion. Many metrics are used to evaluate parts of these activities in isolation, but establishing confidence for software assurance requires considering the fully integrated solution to establish overall sufficiency.

1.1 Examples of Product and Process Confidence

As an example of the complexity in establishing confidence, consider one aspect of product performance. When used, the product must meet some level of performance (e.g., sub-second response time). Assurance includes tests to confirm that the final product meets the requirements. Best practices start with building a computational model during design and using simulations to demonstrate assurance using engineering analysis. Assurance continues into the implementation. For example, unit testing provides assurance that a component behaves as specified by the model. If necessary, corrective action can be taken during the design and implementation phases.

An additional complexity for software assurance is recognizing that software is never defect free, and up to 5% of the unaddressed defects are vulnerabilities [Ellison 2014]. According to Jones

1 This same definition is applied in the National Defense Authorization Act (NDAA) of 2013 [PL 112-239, Sec. 933(2)].

and Bonsignour, the average defect level in the U.S. is 0.75 defects per function point or 6,000 per million lines of code (MLOC) for a high-level language [Jones 2011]. Very good levels would be 600 to 1,000 defects per MLOC, and exceptional levels would be below 600 defects per MLOC.

Thus, software cannot always function perfectly as intended. How can confidence be established? One option is to use measures that establish reasonable confidence that security is sufficient for the operational context. Assurance measures are not absolutes, but information can be collected that indicates whether key aspects of security have been sufficiently addressed throughout the lifecycle to establish confidence that assurance is sufficient for operational needs.

At the start of development, much about the operational context remains undefined, and there is a general knowledge of the operational and security risks that might arise as well as the security behavior that is desired when the system is deployed. This vision provides only a limited basis for establishing confidence in the behavior of the delivered system.

Over the development lifecycle, as the details of the software and operational context incrementally take shape, it is possible, with well-selected measurements, to incrementally increase confidence and eventually confirm that the delivered system will achieve the level of software assurance desired. When acquiring a product, if it is not possible to conduct measurement directly, the vendor should be contacted to provide data that shows product and process confidence. Independent verification and validation should also be performed to confirm the vendor's information.

A comparison of software and hardware reliability provides some insight into challenges for managing software assurance. An evaluation of hardware reliability uses statistical measures, such as the mean time between failures (MTBF) since hardware failures are often associated with wear and other errors that are frequently eliminated over time. A low number of hardware failures increases our confidence in a device's reliability.

The differences between software and hardware reliability are reflected in their associated failure-distribution curves shown in Figure 1. A *bathtub curve*, shown in the left graph, describes the typical failure distribution for hardware. The bathtub curve consists of three parts: a decreasing failure rate (of early failures), a constant failure rate (of random failures), and an increasing failure rate (of wear-out failures), as wear increases the risk of failure. Software defects exist when a system is deployed. Software's failure distribution curve, shown in the right graph of Figure 1, reflects changes in operational conditions that exercise those defects as well as new faults introduced by upgrades. The reduction of errors between updates can lead system engineers to make reliability predictions for a system based on a false assumption that software is perfectible over time. Complex software systems are never as error free as described above.

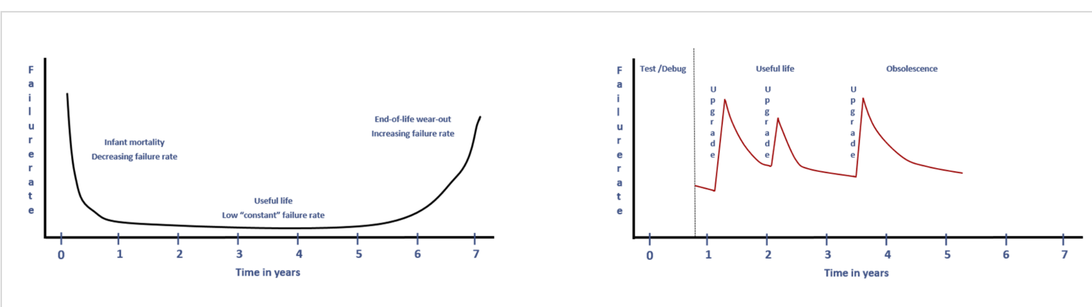

Figure 1: Failure Distribution Curves

As noted in the *2005 Department of Defense Guide for Achieving Reliability, Availability, and Maintainability (RAM)*,[2] a lack of observed software defects is not necessarily a predictor for improved operational software reliability. Defects are inserted into the software before it is deployed, and operational failure results from environmental conditions that were not considered during testing. Too little reliability engineering was a key reason for the reliability failures described in the DoD RAM guide. This lack of reliability engineering was exhibited by failure to design-in reliability early in the development process and the reliance on predictions (i.e., using reliability defect models) instead of conducting engineering design analysis.

The same problem applies to software assurance. Software assurance must be engineered into the design of a software-intensive system. Designing in software assurance requires going beyond identifying defects and security vulnerabilities towards the end of the lifecycle (reacting) and extending to evaluating how system requirements and the engineering decisions made during design contribute to vulnerabilities. Many known attacks are the result of poor acquisition and development practices.

This approach to software assurance depends on establishing measures for managing software faults across the full acquisition lifecycle. It also requires increased attention to earlier lifecycle steps, which anticipate results and consider the verification side as shown in Figure 2. Many of these steps can be performed iteratively with opportunities in each cycle to identify assurance limitations and confirm results.

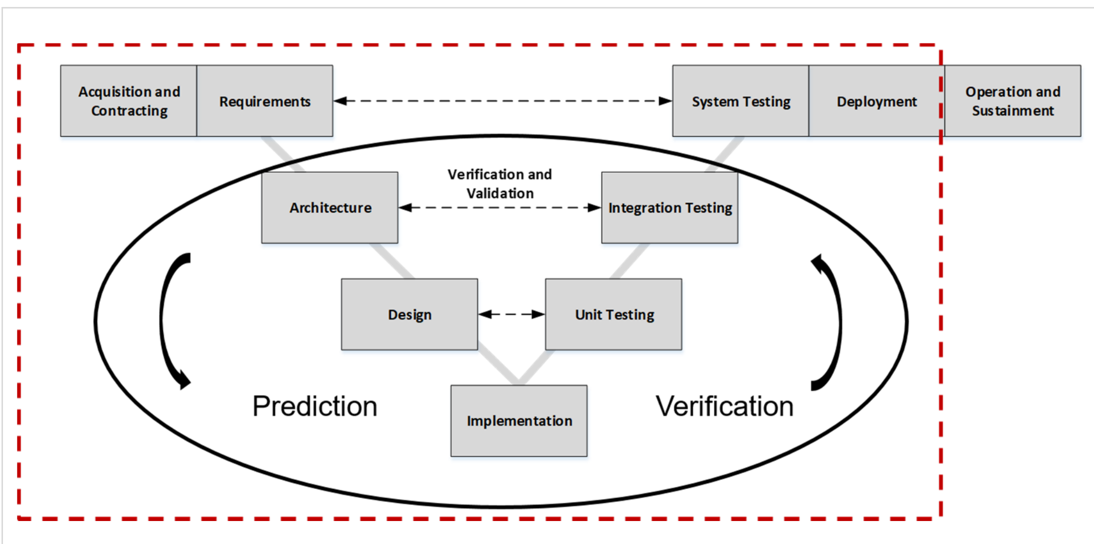

Figure 2: Lifecycle Measures

2 https://acc.dau.mil/CommunityBrowser.aspx?id=378067

1.2 Structure of This Report

This report was developed to provide acquirers, program managers, and contractors with an approach for using metrics to establish confidence that the systems they plan to field will have sufficient software assurance.

Section 2 provides insight into how measurement can be linked to practices and used as evidence of software assurance.

Section 3 provides insights into the range of available metrics that can be collected for software assurance practices and how the most useful ones, in a specific situation, might be selected.

Section 4 provides insights into the challenges of using lifecycle practices and suggests metrics to support software assurance.

Section 5 presents conclusions and proposed next steps.

2 Structuring Software Assurance Practices for Measurement

2.1 Defining the Software Assurance Target

Software assurance needs context to measure its practices usefully. Some software assurance targets[3] must be defined for the system to be fielded. It is then possible to identify ways that engineering and acquisition ensure—through policy, practices, verification, and validation—that the software assurance targets are addressed.

For example, if the system being delivered is a plane, a key mission concern is that the plane can continue to fly and perform its mission even if it's experiencing problems. Therefore, our stated software assurance goal for this mission might be "mission-critical and flight-critical applications executing on the plane or used to interact with the plane from ground stations will have low cybersecurity risk."

To establish activities that support meeting this software assurance goal, software assurance practices should be integrated into the lifecycle. The Software Assurance Framework (SAF), a baseline of good software assurance practices for system and software engineers assembled by the SEI, can be used to confirm the sufficiency of software assurance and identify gaps in current lifecycle practices [Alberts 2017]. A range of evidence can be collected from these practices across a lifecycle to establish confidence that software assurance is addressed.

Evaluation of this evidence should be integrated into the many monitoring and control steps already in a lifecycle, such as engineering design reviews, architecture evaluations, component acquisition reviews, code inspections, code analyses and testing, flight simulations, milestone reviews, and certification and accreditation. Through the analysis of the selected practices, evidence and metrics can be generated to quantify levels of assurance, which, in turn, can be used to evaluate the sufficiency of a system's software assurance practices. A well-defined evidence-collection process can be automated as part of a development pipeline to establish a consistent, repeatable process.

2.2 The SAF

The SAF [Alberts 2017] defines important software assurance practices for four categories: process management, project management, engineering, and support. (See Figure 3.) Each category comprises multiple areas of practice, and specific practices are identified in each area. To support acquirers, relevant acquisition and engineering artifacts—where evidence can be provided—are documented for each practice, and an evaluator looks for evidence that a practice is implemented by examining the artifacts related to that practice.

[3] In this report, use of the word *target* refers to a *goal* or *claim*.

Because most organizations use unique lifecycle models structured to support the specific systems and software products they deliver, using a framework of practices allows tailoring based on the specific needs of a program in any organization.

Many relevant practices focus on cybersecurity, which is defined in Merriam-Webster[4] as "measures taken to protect a computer or computer system (as on the Internet) against unauthorized access or attack." A system containing vulnerabilities that can be compromised to allow unauthorized access reduces the confidence of software assurance.

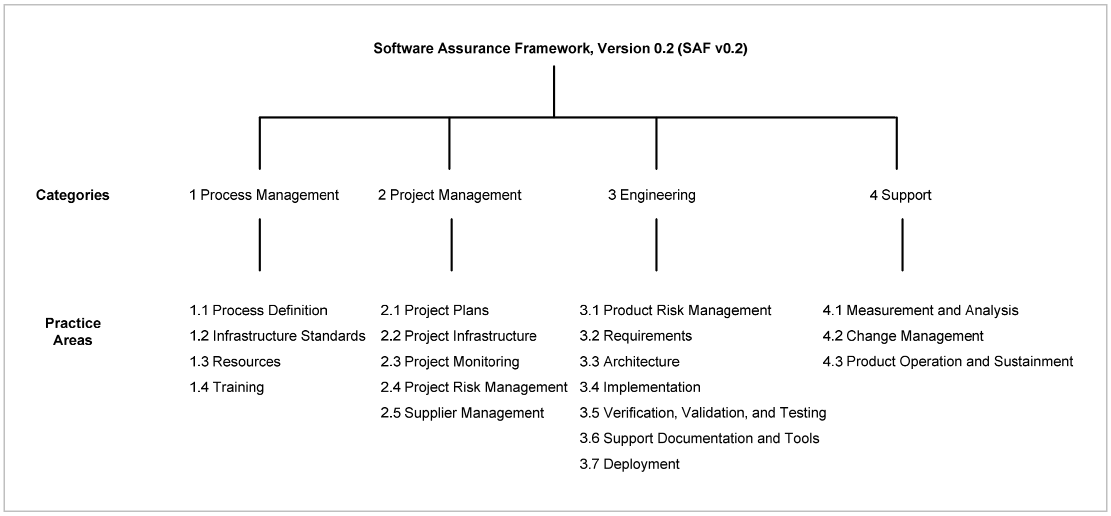

Figure 3: Software Assurance Framework

2.3 Justifying Sufficient Software Assurance Using Measurement

Just as there is no single practice that addresses software assurance, there is no one single measurement that demonstrates that a software assurance target has been achieved. The use of many metrics is required to determine that a range of practices is sufficiently addressed and the product performs as expected. These metrics must be connected to the software assurance target in a manner that supports increased confidence (or not) across the lifecycle.

One form of structuring metric information is an *assurance case*.[5] Metrics provide evidence in support of a software assurance target based on justification of the value of the evidence (aka argument). Such evidence does not imply any kind of guarantee or certification. It is simply a way to document rationale behind software assurance decisions. Assurance cases were originally used to show that systems satisfy their safety-critical properties. For that use, they are called *safety cases.* Effective measurements require planning to determine what to measure and analysis to determine what the measures reveal as evidence in support of a target.

4 https://www.merriam-webster.com/dictionary/cybersecurity

5 An assurance case is defined as a documented body of *evidence* that provides a convincing and valid *argument* that a specified set of critical *claims* about a system's properties are adequately justified for a given application in a given environment.

An assurance case simply documents the verification of a claim. For example, an assurance case for the performance example described in Section 1.1 consists of the computational model, simulations that verify that model, unit tests that verify the implementation of the model, and tests of the integrated system.

Several observations about how an assurance case can be used include the following:

- Creating a verification argument and identifying supporting evidence should be the expected output of normal development activities.
- An assurance case is developed incrementally. For this example, the outline of an assurance case was developed during design. It is likely refined during implementation to satisfy verification requirements.
- Independent reviewers can evaluate the assurance argument and sufficiency of proposed or supplied evidence throughout the development lifecycle.

Software assurance metrics are needed to evaluate both the practices in a software assurance practice area as well as the resulting assurance of the product. For example, in the SAF Engineering practice area, the engineers must (1) know what to do, (2) actually do it, and (3) provide evidence that what they did is sufficient.

However, there are many competing qualities (e.g., performance, safety, reliability, maintainability, usability) an engineer must consider in addition to software assurance, and the result must provide sufficient assurance to meet the target. Answers to the questions in Table 1 provide evidence that the engineering was performed effectively. Further evidence is needed to determine if the software assurance results based on the engineering decisions meet the target.

Table 1: Engineering Questions

Effectiveness	Was applicable engineering analysis incorporated in the development practices?
Trade-offs	When multiple practices are available, have realistic trade-offs been made between the effort associated with applying a technique and the improved result that is achieved? (The improved result refers to the efficiency and effectiveness of the techniques relative to the type of defect or weakness.)
Execution	How well was the engineering done?
Results applied	Was engineering analysis effectively incorporated into lifecycle development?

The Goal/Question/Metric (GQM) paradigm[6] can be used to establish a link between the software assurance target and the engineering practices that should support the target. The GQM approach was developed in the 1980s as a mechanism for structuring metrics and is a well-recognized and widely used metrics approach.

To focus the use of GQM on software assurance, consider an example. An engineering practice for software assurance identifies and protects the ways that a software component can be compro-

6 Read about the Goal Question Metric Approach on the University of Maryland website: https://www.cs.umd.edu/~basili/publications/technical/T78.pdf.

mised (aka attack paths). Such a practice must integrate into all phases of the acquisition and development lifecycles. Measures to provide assurance evidence can be collected from activities that implement this practice in several lifecycle steps, such as the following:

- Requirements: What are the requirements for software attack risks, and are they sufficient for the expected operational context?
- Architecture through design: What security controls and mitigations must be incorporated into the design of all software components to reduce the likelihood of successful attacks?
- Implementation: What steps must be taken to minimize the number of vulnerabilities inserted during coding?
- Test, validation, and verification: How will actions performed during test, validation, and verification address software attack risk mitigations?

For each of these engineering questions, explore relevant outputs and metrics that can be used to establish, collect, and verify appropriate evidence. Since each project is different in scope, schedule, and target assurance, actual implemented choices should be the metrics that provide the greatest utility.

2.4 An Implementation Process for Each Metric

Selecting a metric is only the first step in establishing useful measurement of software assurance. Metric data must also be collected, analyzed, and evaluated to identify potential concerns. Each concern triggers a response determination and an implementation of that response. Figure 4 describes the steps for establishing and using a metric.

Figure 4: Metrics Development Process

2.4.1 Collect Data

Collecting measurement data starts with answering the following standard questions of who, what, where, when, and how.

***Who** performs the practice(s) selected to measure?* If there is direct access to who performs the practices, it is possible to request the data. However, in many cases, the practices are performed by contracted resources, and a deliverable must be added to performance criteria to ensure practices are performed. This addition may mean contract modifications and increased costs.

***What** should be collected and by whom?* In some cases, the data is already available and is being used for a related secondary purpose. It's likely that no one is collecting the information because it hasn't been required, or what is being collected is imprecise or insufficiently correlated to what must be evaluated. Are mechanisms available to collect the needed data? Are there log entries that can be assembled or tools that can be applied to collect the data? If there is no way to collect the data needed, a surrogate may be able to provide a close approximation of what is needed.

***Where** might the data be collected and how many sources should be used?* How granular should the data be? Is information needed about every line of code, every software module, every component, or each product within the system? Or is information needed at an integration level? Is it necessary to collect detailed data and construct the combined view, or can the data be collected at a point where it will reflect the combinations? Is there a single point where the practice being measured is performed, or is it spread throughout many separate steps, separate lifecycle activities, and separate contractors? Are the practices being inserted into the lifecycle, and do the measurement activities need to be part of that transition? How many sources must participate to make the measurement useful? In many cases, the volume of data may be too high for manual analysis, and the collection process should be automated to be practical.

***When** should the metric be collected to be useful?* If a metric is used for prediction, then it must be part of early lifecycle activities. If it's used for lifecycle performance verification, then it should be part of later lifecycle activities. How frequently (e.g., daily, weekly, monthly, at the end of a cycle, or as part of planned reviews) is this information needed? There is no reason to expend resources to collect data more frequently than needed.

***How** should the information be assembled for analysis?* Data is useful only if it's analyzed, and data analysis is time and resource intensive. Mechanisms must be in place to isolate data needed to conduct assurance analysis from the many log files and other data repositories that potentially contain millions of records. Data that is classified and cannot be shared with decision makers is useless unless the analysis is framed so the decisions the data is intended to influence are addressed within the classification boundaries.

2.4.2 Analyze and Identify Issues and Gaps

Measurement data is collected so that it can be used to influence action. Measurements can show that work is proceeding as expected, and no action beyond continuing the current course is required. Measurements can show deviations from a desired range of performance, indicating the need for further evaluation, possible engineering changes, or different measures because the data does not correlate to expectations. Any of these outcomes requires knowledge of what constitutes expected data so that undesirable behavior can be identified. A worthy measurement plan predefines what the collected data means and how it should be used to influence actions so that the interpretation of the results and selected responses are appropriate.

Who reviews the data for potential response? How do they determine what is out of acceptable bounds and when action is required? Is there a single decision point? Or are performers at a granular level expected to (1) correct issues related to measures within a certain range and (2) notify decision makers at the next level when those bounds are exceeded? Each selected measure can have different responses to these questions based on how the organization chooses to implement its decision making.

2.4.3 Evaluate and Determine the Need for Response

There are several possible responses to measures that are considered out of bounds. Initially, the data should be confirmed to ensure its validity. Were the collection and submission processes followed so that the data has integrity? Are the metrics appropriate to indicate specific action, or are they potential warning indicators that should trigger further monitoring, data collection, and analysis?

If the data is believable, then what are the potential impacts indicated by an out-of-bounds condition? There could be mission success impacts, system/product performance impacts, operational capability impacts with future limitation implications, etc.

If the measures can be considered predictive, then what actions should be considered to prevent, mitigate, or monitor the possible impact? If the possible impact is unacceptable, what must change to align the predicted outcome with the desired result?

If the measures verify capability, are the conditions posed by the unexpected variance great enough to justify rework of some or all of the system? Or will responsibility, and possibly future change requests, be transferred to operations?

Any of the above responses requires criteria for evaluating the severity of impact and the immediacy of expected response. Mechanisms for communicating the need for response to current or future performers is also required.

2.4.4 Implement a Response and Determine Needed Monitoring

Once the desired response is determined, it’s necessary to communicate to those expected to respond so that they (1) know what they must do, (2) understand the expected response time, and (3) have the proper authorization to act. How are such situations tracked to determine resolution? Will additional measures be needed to confirm the expected outcome, or is future monitoring of the existing measures sufficient?

It’s beneficial to periodically monitor and tune this process to improve the metrics used and the actions that are determined and implemented based on those metrics. Also system and organizational changes can impact the metrics process.

3 Selecting Measurement Data for Software Assurance Practices

The SAF documents practices for process management, program management, engineering, and support. For any given software assurance target, there are GQM questions that can be linked to each practice area and individual practice to help identify potential evidence. In this section, this approach is used to develop an example that shows how practices in each area can be used to provide evidence in support of a software assurance target.

The SAF provides practices as a starting point for a program, based on the SEI's expertise in software assurance, cybersecurity engineering, and risk management. Each organization must tailor the practices to support its specific software assurance target—possibly modifying the questions for each relevant software assurance practice—and select a starting set of metrics for evidence that is worth the time and effort needed to collect it.

3.1 Example Software Assurance Target and Relevant SAF Practices

Consider the following software assurance target: *Supply software to the warfighter with acceptable software risk.* To meet this software assurance target, two sub-goals are needed (based on the definition of software assurance):

> Sub-Goal 1: Supply software to the warfighter that functions in the intended manner. (Since this is the primary focus of every program, and volumes of material are published about it, this sub-goal does not need to be further elaborated.)
>
> Sub-Goal 2: Supply software to the warfighter with a minimal number of exploitable vulnerabilities. (The remainder of this section provides a way to address this sub-goal.)

SAF-Based Questions

Using the SAF,[7] the following questions should be asked to address sub-goal 2: *Supply software to the warfighter with a minimal number of exploitable vulnerabilities*.

1. Process Management: Do process management activities help minimize the potential for exploitable software vulnerabilities?

   1.1. Process Definition: Does the program establish and maintain cybersecurity processes?

   1.2. Infrastructure Standards: Does the program establish and maintain security standards for its infrastructure?

[7] See Figure 3 for the SAF's structure of practice areas.

1.3. Resources: Does the program have access to the cybersecurity resources (e.g., personnel, data, assets) it needs?

1.4. Training: Does the program provide security training for its personnel?

2. Program Management: Do program management activities help minimize the potential for exploitable software vulnerabilities?

2.1. Program Plans: Has the program adequately planned for cybersecurity activities?

2.2. Program Infrastructure: Is the program's infrastructure adequately secure?

2.3. Program Monitoring: Does the program monitor the status of cybersecurity activities?

2.4. Program Risk Management: Does the program manage program-level cybersecurity risks?

2.5. Supplier Management: Does the program consider cybersecurity when selecting suppliers and managing their activities?

3. Engineering: Do engineering activities minimize the potential for exploitable software vulnerabilities?

3.1. Product Risk Management: Does the program manage cybersecurity risk in software components?

3.2. Requirements: Does the program manage software security requirements?

3.3. Architecture: Does the program appropriately address cybersecurity in its software architecture and design?

3.4. Implementation: Does the program minimize the number of vulnerabilities inserted into its software code?

3.5. Testing, Validation, and Verification: Does the program test, validate, and verify cybersecurity in its software components?

3.6. Support Tools and Documentation: Does the program develop tools and documentation to support the secure configuration and operation of its software components?

3.7. Deployment: Does the program consider cybersecurity during the deployment of software components?

4. Support: Do support activities help minimize the potential for exploitable software vulnerabilities?

4.1. Measurement and Analysis: Does the program adequately measure cybersecurity in acquisition and engineering activities?

4.2. Change Management: Does the program manage cybersecurity changes to its acquisition and engineering activities?

4.3. Product Operation and Sustainment: Is the organization with responsibility for operating and sustaining the software-reliant system managing vulnerabilities and cybersecurity risks?

There are many possible metrics that could provide indicators of how well each practice in each practice area is addressing its assigned responsibility for meeting the goal. The tables in Appendices B-E provide metric options to consider when addressing the questions for each practice area except *3.1 Product Risk Management*, which, for this example, was not useful since the system under development is the product.

There are many ways that the information provided in Appendices A-E can be used for practices, outputs, and metrics. An organization can start with

- existing practices to identify related metrics
- known outputs to identify useful software assurance metrics
- known attacks to identify useful practices and measures for future identification

Three examples are included in this section.

3.2 Example for Selecting Evidence for Software Assurance Practices

A reasonable starting point for software assurance measurement is with practices that the organization understands and is already addressing. Consider the following example, which draws practices and metrics from Appendix D.

The DoD requires a program protection plan, and evidence could be collected using metrics for engineering practices (see Figure 3, practice group 3) that show how a program is handling program protection.

In Engineering practice area *3.2 Requirements*, data can be collected to provide a basis for completing the program protection plan. Relevant software assurance data can come from requirements that include the following:

- the attack surface
- weaknesses resulting from the analysis of the attack surface, such as a threat model for the system

In Engineering practice area *3.3 Architecture*, data is collected to show that requirements can be addressed. This data might include the following:

- the results of an expert review by those with security expertise to determine the security effectiveness of the architecture
- attack paths identified and mapped to security controls
- security controls mapped to weaknesses identified in the threat modeling activities in practice 3.2

In Engineering practice area *3.4 Implementation*, data can be provided from activities, such as code scanning, to show how weaknesses are identified and removed. This data might include the following:

- results from static and dynamic tools and related code updates
- the percentage of software evaluated with tools and peer review

In Engineering practice area *3.5 Verification, Validation, and Testing*, data can be collected to determine that requirements have been confirmed and the following evidence would be useful:

- percentage of security requirements tested (total number of security requirements and MLOC)
- code exercised in testing (MLOC)
- code surface tested (% of code exercises)

Each selected metric must have a process that establishes how data is collected, analyzed, and evaluated based on information provided in Section 2.4 of this report.

3.3 Example for Finding Metrics Data in Available Documentation

For each SAF practice, a range of outputs (e.g., documents, presentations, dashboards) is typically created. In Appendices B through E, examples of these outputs are provided for each SAF practice. The form of an output may vary based on the lifecycle in use. An output may be provided at multiple points in a lifecycle with increased content specificity. Available outputs can be evaluated and tuned to include the desired measurement data.

In Engineering practice area *3.2 Requirements*, the SAF includes the following practice:

> *A security risk assessment is an engineering-based security risk analysis that includes the attack surface (those aspects of the system that are exposed to an external agent) and abuse/misuse cases (potential weaknesses associated with the attack surface that could lead to a compromise). This activity may also be referred to as threat modeling.*

A *security risk assessment* exhibits outputs with specificity that varies by lifecycle phase. Initial risk assessment results might include only that the planned use of a commercial database manager raises a specific vulnerability risk that should be addressed during detailed design. The risk assessment associated with that detailed design should recommend specific mitigations to the development team. Testing plans should cover high-priority weaknesses and proposed mitigations.

Examples of useful data related to measuring this practice and that support the software assurance target appear in the following list:

- recommended reductions in the attack surface to simplify development and reduce security risks
- prioritized list of software security risks
- prioritized list of design weaknesses
- prioritized list of controls/mitigations
- mapping of controls/mitigations to design weaknesses
- prioritized list of issues to be addressed in test, validation, and verification

The outputs of a security risk assessment depend on the experience of the participants as well as constraints imposed by costs and the schedule. An analysis of this data should include consideration for missing security weaknesses or poor mitigation analysis, which increases operational risks and future expenses.

Another practice in Engineering practice area *3.2 Requirements* is

> *Conduct reviews (e.g., peer reviews, inspections, and independent reviews) of software security requirements.*

Output from reviews includes issues raised in internal reviews, review status, and evaluation plans for software security requirements.

Analysis of the issues arising in various reviews should answer the questions shown in following list to determine data that would be useful in evaluating progress toward the software assurance goal.

- For software security requirements, what has not been reviewed? (Examples include the number, difficulty, and criticality of "to be determined" [TBD] and "to be added" [TBA] items.)
- Where are there essential inconsistencies in the analysis and/or mitigation recommendations? (Examples include the number/percentage, difficulty, and criticality of the differences.)
- Is there insufficient information for performing a proper security risk analysis? (Examples include emerging technologies and/or functionality where there is a limited history of security exploits and mitigation.)

3.4 Sustainment Example

The Heartbleed vulnerability is an example of a design flaw. Could software assurance practices and measures have identified this type of problem before it was fielded?

The `assert` function for the flawed software accepts two parameters: a string `S` and an integer `N` and returns a substring of `S` of length `N`. For example, `assert ("that",3)` returns `tha`. A vulnerability existed for calls where `N` is greater than then the length of `S`. For example, `assert("that",500)` returns a string starting with "`that`" followed by 496 bytes of memory data stored adjacent to the string `that`. Calls such as this one enable an attacker to view what should be inaccessible memory contents. The input data specification that the value of `N` was less than or equal to the length of the string was never verified.

The practices listed in Table 2 come from several SAF practices in the Engineering practice area that should provide enough evidence to justify the claim that the Heartbleed vulnerability was eliminated.

Table 2: Practices/Outputs for Evidence Supporting Sustainment Example

Practice	Output
Threat modeling	Software risk analysis identifies “input data risks with input verification” as requiring mitigation.
Design includes mitigation	Input data verification is a design requirement.
Software inspection	Software inspections confirm the verification of all input data.
Testing	Testing plans include invalid input data. Test results show mitigation is effective for supplied inputs.

4 Challenges for Addressing Lifecycle Software Assurance

As mentioned earlier in this report, the role of assurance metrics and data varies with the type of assurance target. Earlier examples demonstrated that the effective use of metrics for software assurance in engineering practices requires coordinating data across many practices in the Engineering practice area.

Functional requirements typically (1) describe what a system should do and (2) focus on required behavior that can be validated. Assurance requirements are more likely expressed in terms of what a system should not do and are much more difficult (if not impossible) to confirm. However, we should consider evaluations that show that a behavior is less likely to occur.

For example, we can verify that the authentication and authorization functions meet requirements and that authorization is confirmed when sensitive data is accessed. However, that evidence is insufficient to demonstrate assurance because only authorized users can access a data set. An attacker does not need to exploit a weakness in those functions. Instead, they can use a vulnerability in the functional software to change software performance and bypass authentication checks. In other words, vulnerabilities enable an attack to bypass system controls. To reduce the likelihood of this bypass occurring, practices that remove vulnerabilities are critically needed.

4.1 Acquisitions Can Initiate Software Assurance with Independent Verification and Validation

Challenge: Contractors are required to address a risk management framework based on existing policy; contractors need to consider software assurance as well. Can the two be combined?

Many government agencies use the *NIST Risk Management Framework (RMF)* [NIST 2014] to identify practices for cybersecurity that also address software assurance. These practices are included in a contract and evaluated as part of an independent verification and validation (IV&V) process to confirm the level of cybersecurity and software assurance risk addressed.

As an example, three areas of interest that could be combined were selected. (Additional examples are provided in Appendix A.)

1. The first area of interest is Software Flaw Remediation, which covers five RMF controls as follows:
   - SI-2 Flaw Remediation
     - SI-2(1) Flaw Remediation | Central Management
     - SI-2(2) Flaw Remediation | Automated Flaw Remediation Status
     - SI-2(3) Flaw Remediation | Time to Remediate Flaws/Benchmarks for Corrective Actions
     - SI-2(6) Flaw Remediation | Removal of Previous Versions of Software/Firmware

This area of interest is handled by SAF Engineering practice area *3.2 Implementation* as part of "Evaluation practices (e.g., code reviews and apply tools) are applied to identify and remove vulnerabilities in delivered code (including code libraries, open source, and other reused components)."

The same metrics could be selected to demonstrate meeting both RMF and software assurance expectations from the following list:

- % of vendor contracts requiring the use of evaluation practices and reporting vulnerability metrics
- code coverage (aka % of code evaluated [total and by each type of review])
- vulnerabilities per MLOC identified and removed
- unaddressed vulnerabilities per MLOC
- % code libraries evaluated
- % open source components evaluated
- % legacy components evaluated
- count of high-priority vulnerabilities identified and the count of those removed

2. The second area of interest is Malicious Code Protection, which covers the following four RMF controls:
   - SI-3 Malicious Code Protection
     - SI-3(1) Malicious Code Protection | Central Management
     - SI-3(2) Malicious Code Protection | Automatic Updates
     - SI-3(10) Malicious Code Protection | Malicious Code Analysis

   This area of interest is be handled by the SAF Engineering practice area *3.2 Implementation* as well. Specific metrics for these practice areas are provided in Appendix D.

3. The third area of interest is Software Supply Chain Protection, which covers the following seven RMF controls:
   - SA-12 Supply Chain Protection
     - SA-12(1) Supply Chain Protection | Acquisition Strategies/Tools/Methods
     - SA-12(5) Supply Chain Protection | Limitation of Harm
     - SA-12(8) Supply Chain Protection | Use of All-Source Intelligence
     - SA-12(9) Supply Chain Protection | Operations Security
     - SA-12(11) Supply Chain Protection | Penetration Testing/Analysis of Elements, Processes, and Actors
   - SA-22 Unsupported System Components

   This area of interest is addressed by practices in SAF Project Management practice area *2.5 Supplier Management*, which includes five practice activities and a range of metrics for each practice as shown in Appendix C.

An additional 15 cybersecurity areas that map to an additional 20 RMF controls (listed in Appendix A) can cross-reference to SAF practice areas and practices. These SAF practice areas and practices link to potential metrics that can be collected and analyzed at checkpoints throughout the acquisition lifecycle to confirm that they are addressed.

For the DoD, milestone reviews in an acquisition lifecycle can be used to review selected metrics and monitor how well the contractor is addressing the selected RMF controls and practices for software assurance. As described in Sections 2 and 3 of this report, the acquirer must determine which data to collect and how it will be evaluated to determine if the results are sufficient.

4.2 Monitoring the Development of a Custom Software Acquisition

Challenge: What evidence is needed to ensure that vulnerabilities are addressed by a contractor?

It is a common practice for a vendor to report the tools it uses to address vulnerabilities as part of its execution pipeline. This source of evidence should map to the expected practice that this evidence supports to determine how well each part of the practice is addressed. Also, all lifecycle activities must be considered since potential vulnerabilities can be introduced at any stage of the lifecycle. Therefore, the acquirer should not just accept what a vendor reports that it performs, but the acquirer should also map what is reported to the needed practices and identify gaps and opportunities for improvement.

Capers Jones analyzed over 13,000 projects for the effects of general practices (e.g., inspections, testing, and analysis) on improving software quality [Jones 2012]. His analysis shows that using a combination of techniques is best. Many of the limitations associated with tools such as static analysis, which have high rates of false positives and false negatives [Wedyan 2009], can be mitigated by other development practices.

Jones' analysis of projects showed that a combination of inspections, static analysis, and testing was greater than 97% efficient in identifying defects. However, these analyses address only the identify part of SAF Engineering practice area *3.2 Implementation* as part of "Evaluation practices (e.g., code reviews and apply tools) are applied to identify and remove vulnerabilities in delivered code (including code libraries, open source, and other reused components)," and additional actions must be performed to remove them.

The *Security Development Lifecycle (SDL)* encouraged other developers to include security analysis earlier in the development lifecycle [Howard 2006]. Vulnerabilities created during design should be identified and removed during risk assessments or in design and implementation. Assurance now depends, in part, on how well a developer anticipates how a system can be compromised and how well the developer chooses and implements effective mitigations. Practices that anticipate software weaknesses are included in SAF area 3.2, as shown in Table 3.

Table 3: Requirements (SAF Engineering Practice Area 3.2)

Activities/Practices	Outputs
Conduct a security risk analysis, including threat modeling and abuse/misuse cases.	Prioritized list of software security risks Prioritized list of design weaknesses Prioritized list of controls/mitigations Mapping of controls/mitigations to design weaknesses

Threat modeling analyzes how a software design can be compromised. Such analysis typically considers how an attack can compromise the information, flows, data stores, and software that processes the data and can draw on the extensive documentation of security exploits as represented by the Common Weakness Enumeration (CWE),[8] the Common Vulnerabilities and Exposure Enumeration (CVE),[9] and the Common Attack Pattern Enumeration and Classification (CAPEC).[10] The output can describe the likelihood of various classes of threats, such as a denial of service or disclosure of information.

Verification should guide the choice of mitigations. Can claims about a mitigation be verified? In other words, what is the level of confidence an acquirer should have with the choice of mitigations? Creating an argument that a developer reduced or eliminated vulnerabilities (i.e., a developer's assurance case) should start with risk analysis. The strength of the assurance argument and its eventual verification depends, in part, on the evidence provided to support the mitigation of software risks. An acquirer should consider the evidence that supports the following:

1. validity of the risk analysis
2. cost effectiveness of the mitigations with respect to their effects on mission outcomes
3. effective implementation of the chosen mitigations

The output of a risk assessment includes predictions of how a system can be compromised with the risk priorities weighted by likelihood and consequences. Metrics now evaluate the engineering analysis in items 1 and 2, while the incorporation of that engineering analysis is determined in later lifecycle activities (item 3).

Instead of trying to confirm that the evidence provided for a practice is sufficient, instead ask why the evidence may be insufficient or defective [Goodenough 2010]. For example, unanticipated risks raised during a program technical review or by an independent product risk assessment reduce the confidence in a developer's risk analysis. Examples of other doubts that could arise include the following:

- The test plans did not include all hazards identified during design.
- The web application developers had limited security experience.
- The acquirer did not provide sufficient data to validate the modeling and simulations.
- Integration testing did not adequately test recovery after component failures.

A developer should be able to provide evidence that confirms items 2 and 3 were addressed. For example, assume a data flow includes an SQL database as a data store. A risk assessment does the following:

- estimates the risk of an SQL-injection attack as described in CWE-135
- describes how a successful exploit could lead to a malicious modification of data or the exposure of information to individuals who are not supposed to have access to it

---

8 http://cwe.mitre.org/community/swa/index.html

9 https://cve.mitre.org/cve/

10 https://capec.mitre.org/

- recommends mitigations to reduce the risk of an SQL-injection vulnerability

It is difficult to verify that a routine, even written by an experienced coder, prevents an SQL injection. A CWE-recommended mitigation is to use a vetted library or framework. Such a recommendation is an engineering decision expressed as a coding rule to be enforced during implementation. *The Consortium for IT: Software Quality (CISQ)* states that the validation of the use of such a library can be automated by scanning the source code and does not require the coder to have extensive security expertise [CISQ 2012]. A developer following the CISQ approach can provide an acquirer with an assurance justification (as shown in Figure 5).

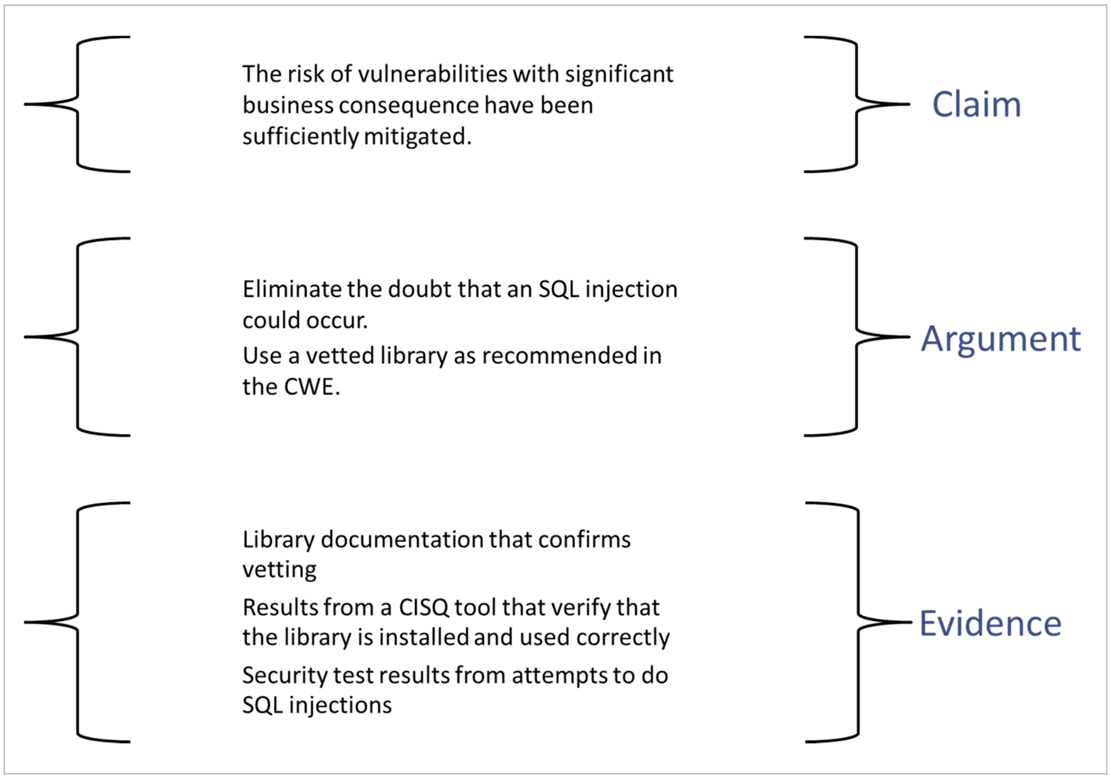

Figure 5: SQL-Injection Assurance Case

The CISQ approach, like static analysis, is based on the analysis of developed source code. However, the objective of the approach is to eliminate vulnerabilities during coding rather than identifying defects after they are injected.

Confidence in reducing defects, as demonstrated by Capers Jones, depends on evidence that the security risks and recommended mitigations were (1) considered during design and design reviews, and during inspections; and (2) incorporated in test plans (like what was done for the SQL-injection example).

4.3 Monitoring Integration of Third-Party Software

Challenge: Why is supply chain risk management such a growing source of acquisition concern?

An increasing proportion of software development involves integrating commercial software. An acquirer has limited visibility into the engineering of that software and may rely on test labs and other alternative practices. Such software includes database management systems and infrastructure services, such as identity management for authorization and authentication. The appropriate security measures depend on the context, which only the acquirer knows.

Supply chain risk management refers to the collection of practices that manage the risks associated with the external manufacture or development of hardware and software components. There are two sources of supply chain risks:

1. The supply chain is compromised, and counterfeit and tampered products are inserted.
2. Poor development and manufacturing practices introduce vulnerabilities.

For example, there was a vulnerability in a widely used implementation of the secure socket layer protocol that was used for securing web communications. The vulnerability potentially exposed memory data (e.g., passwords, user identification information, and other confidential information) to unauthorized users. At the time of the announcement in 2014, there did not appear to be any tools available that would have discovered the vulnerability [Kupsch 2014]. The vulnerability occurred because the validity of the input to a software function was not verified. In all likelihood, the defect could have been found during a code inspection, but this activity was not part of the development process for this software.

For commercial development, most of the practices that address defects are early in the lifecycle. The acquirer does not see the product until integration and will only be able to monitor the early lifecycle activities through provisions in the contract. This separation is shown in Figure 6. Monitoring vendor development practices depends entirely on information provided by the vendor. When the acquirer simply receives the final product at integration, it does not have direct visibility into the vendor's development practices.

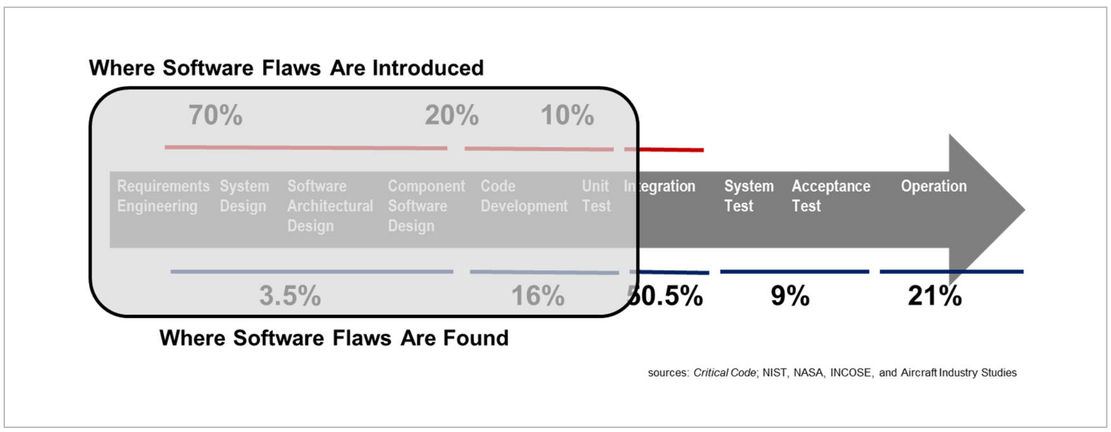

Figure 6: Supply Chain Monitoring

An acquirer must not only monitor a supplier's development practices, but they must also understand how that supplier monitors its suppliers. For example, how does the prime contractor reduce supply chain risks associated with subcontractors and commercial suppliers? Supply chains can be many layers deep, linking organizations with a wide range of defect management approaches.

Product Development

Characteristics of commercial product development that can be available to an acquirer might include the following:

- vulnerability history for the product as reported to the NIST National Vulnerability Database
- standards that a product developer applies, such as The Open Group's *Open Trusted Technology Provider Standard*[11] (O-TTPS) (ISO 20243), which uses evidence of a supplier's capabilities and product security as shown in Table 4

Table 4: Evidence of Supplier Capabilities and Product Security

Evidence of Quality Product Development	Supplier practices conform to best practice requirements and recommendations primarily associated with the activities relating to the product's development.
Evidence of Secure Development	Providers employ a secure engineering method when designing and developing their products. Software providers and suppliers often employ methods or processes with the objective of identifying, detecting, fixing, and mitigating defects and vulnerabilities that could be exploited as well as verifying the security and resiliency of the finished products.
Evidence of Supply Chain Security	Suppliers manage their supply chains through the application of defined, monitored, and validated supply chain processes.

Integrated System Development

A commercial product developer can take advantage of a relatively stable set of suppliers and knowledge of the security risks associated with earlier versions; however, a system integrator requires general knowledge that is applicable across multiple components and suppliers. Characteristics of integrated development include the following:

- integration of independently developed components with limited visibility into the actual code
- inconsistencies in security assumptions among components
- component behavior that is dynamic over time (i.e., each component supported and updated separately)
- components that provide extensibility and customization
- ongoing product upgrades
- multiple components that compound threat analysis and mitigations
- supply chain risk management that includes integration and product risks

While threat modeling for a product can be incrementally upgraded as functionality and threats evolve over time, a distinct threat model must be constructed for each system by the acquirer. For a product to be integrated into a commercial product, the supply chain must be managed by the integrator. For the acquirer of the integrated product, visibility into how the integrator manages its suppliers may be difficult.

11 http://www.opengroup.org/certifications/o-ttps

Commercial software typically can customize and extend capabilities so that an organization can tailor that software to its requirements and operational environment. The implementation of a mitigation might take advantage of such capabilities, but it is more likely that an attack exploits these features. Threat modeling should be applied to identify any new risks and the effect of the changes on recommended mitigations.

4.4 System-of-Systems Assurance

Challenge: Systems are typically integrated with other systems to address a mission. Can software assurance be applied to a system of systems?

The assurance discussed for custom development and for supply chain assurance were associated with eliminating identified defects and vulnerabilities. Threat modeling attempts to reduce the risk of vulnerabilities associated with unexpected conditions. Assurance should also be considered for an organization's work processes, which are based on systems working together to address a mission or business process.

A good example is the August 2003 power grid failure. Approximately 50 million electricity consumers in Canada and the northeastern U.S. were subject to a cascading blackout. The events preceding the blackout included a mistake by tree trimmers in Ohio that took three high-voltage lines out of service and a software failure (a race condition[12]) that disabled the computing service that notified the power grid operators of changes in power grid conditions. With the alarm function disabled, the power grid operators did not notice a sequence of power grid failures that eventually lead to the blackout [NERC 2004].

The alert server was a commercial product. The integration of that component into the power company's system included a rollover to a second server if there was a hardware failure in the primary server. However, the software error that disabled the primary server also disabled the secondary server. This event was the first time that this software fault had been reported for the commercial product.

A key observation by the technical reviewers was that the blackout would not have occurred if the operators knew the alarm service failed. Typically, a response involves finding alternative sources of electricity, and this response typically can be implemented in 30 minutes. Instead of analyzing the details of the alarm server failure, the reviewers asked why the following software assurance claim had not been met [NERC 2004]:

> **Claim:** Power grid operators had sufficient situational awareness to manage the power grid to meet its reliability requirements.

The reviewers proposed the following assurance case. The claim is met if one out of five of the subclaims are satisfied.

[12] The software failure was caused by a race condition. An error in the implementation of the software controls that managed access to the data by multiple processes caused the alarm system to stall while processing an event. With the software unable to complete the alarm event and move to the next one, the alarm processor buffer filled and eventually overflowed.

Sub-Claim	Status
A server provides alarms for condition changes.	Alarm server recovery was designed for a hardware failure. The alarm service did fail over to the secondary server, but the software failure that disabled the primary server also disabled the backup.
Server recovery can be completed within ten minutes.	The commercial system required 30 minutes for a restart.
Operators are notified of the loss of the alarm server.	Automatic notification of server failure was not implemented.
Operators periodically check the output from contingency analysis and state estimators.	This practice was not done since those tools had repeated failures in the preceding week.
An independent real-time monitor of the regional power grid provides alerts.	The independent monitoring organization had concurrent failures.

This operational assurance case should guide the acquisition and integration of commercial power grid software.

5 Conclusions

The Object Management Group established that software measurement relies on discrete indicators to support real-world decision making. It also established that a software assurance indicator is a metric or combination of metrics that provides useful information about the development process, the how the project was conducted, or the characteristics of the product itself.[13]

A key aspect of software assurance in practice is performing activities associated with sound software results. These activities help determine whether the software functions as intended and is free of vulnerabilities. Experience shows that just performing what has traditionally been done for hardware is not sufficient for software. The SAF was used as a set of software practices for exploring possible measurement options. A set of candidate metrics was identified that can connect to some aspect of the execution of each practice in the SAF.

There are many lifecycles used to address software acquisition and development. Each SAF practice can be performed at varying points in a specific lifecycle. The level of specificity available at each point in the lifecycle can be different. Measures taken at some points in the lifecycle are predictive, since they are connected with what is planned. Measures taken after plans are executed can be used to verify that what was planned is what was actually performed.

Identifying a measurement for a practice by itself does not really tell us anything about software assurance. To associate measures with software assurance, it is necessary to determine what a measure tells us in relation to a target, but there is limited field experience in making this association. The examples in this report were provided to demonstrate ways to navigate the various aspects of assurance goal, practice, and measurement in a logical structure. This report also covered use of GQM and aspects of an assurance case to structure examples and show how measurement can demonstrate some aspects of a practice.

The selection of a metric is only the first step in establishing a useful measurement of software assurance. Metric data must be collected, analyzed, and evaluated to identify potential concerns.

Measurement is not unique to software assurance. Performing sound software engineering also includes considering measures for monitoring and controlling results. The examples in this report explore aspects of integrating software assurance measurement into what is already being done for other qualities instead of defining an entirely separate approach.

This report explores what is different about software assurance that must be added to what software engineers are already doing. Based on this exploration, it is asserted that improved software assurance depends on improved engineering. The DoD RAM guide makes that statement for reliability, and the examples in this report confirm the criticality of good engineering for software assurance. Engineering requires that evidence is collected across the lifecycle since the product and what can be measured changes.

13 www.dtic.mil/dtic/tr/fulltext/u2/a592417.pdf

Motivating vendors to address software assurance requires establishing criteria for evaluating the products they produce as well as the processes used to produce them at strategic points in the lifecycle. These evaluations must depend on expert opinion since the range of available data is insufficient for researchers to structure useful patterns of “goodness.” However, the selection and consistent collection of metrics at various points in the lifecycle provide indicators over time that an acquirer can use to monitor and incentivize software assurance improvement.

Appendix A: RMF Controls

	Critical Software Cybersecurity Requirements	RMF Controls Addressed	SAF Practice Areas
1	Secure System/Software Development Lifecycle	SA-3 System Development Life Cycle SA-4(3) Acquisition Process | Development Methods/Techniques/Practices	1.1 Process Definition 2.1 Project Plans
2	Software Development Process, Standards, and Tools	SA-15 Development Process, Standards, and Tools	1.1 Process Definition 1.2 Infrastructure Standards 1.3 Resources 1.4 Implementation 1.5 Verification, Validation and Testing 1.6 Support Documentation and Tools 4.3 Product Operation and Sustainment
3	Software Security Requirements	SA-4 Acquisition Process SA-4(1) Acquisition Process | Functional Properties of Security Controls	1.1 Process Definition 2.5 Supplier Management 3.2 Requirements
4	Software Security Architecture and Design	SA-17 Developer Security Architecture and Design SA-4(2) Acquisition Process | Design/Implementation Information for Security Controls	1.3 Resources 3.3 Architecture
5	Software Configuration Management	SA-10 Developer Configuration SA-10 (1) Developer Configuration Management | Software/Firmware Integrity Verification	1.2 Infrastructure 2.2 Project Infrastructure 3.1 Project Risk Management 3.6 Support Documentation and Tools 3.7 Deployment 4.2 Change Management 4.3 Product Operation and Sustainment
6	Developer Security Testing and Evaluation	SA-11 Developer Security Testing and Evaluation	3.4 Implementation 3.5 Verification, Validation, and Testing 4.3 Product Operation and Sustainment

	Critical Software Cybersecurity Requirements	RMF Controls Addressed	SAF Practice Areas
7	Static Code Analysis	SA-11 (1) Developer Security Testing and Evaluation | Static Code Analysis	3.4 Implementation 3.5 Verification, Validation, and Testing 4.3 Product Operation and Sustainment
8	Dynamic Code Analysis	SA-11 (8) Developer Security Testing and Evaluation | Dynamic Code Analysis	3.4 Implementation 3.5 Verification, Validation, and Testing 4.3 Product Operation and Sustainment
9	Manual Code Reviews	SA-11 (4) Developer Security Testing and Evaluation | Manual Code Reviews	3.4 Implementation 3.5 Verification, Validation, and Testing 4.3 Product Operation and Sustainment
10	Attack Surface Reviews	SA-11 (6) Developer Security Testing and Evaluation | Attack Surface Reviews	3.1 Product Risk Management 3.3 Architecture 3.5 Verification, Validation, and Testing 4.3 Product Operation and Sustainment
11	Software Threat Analysis	SA-11(2) Developer Security Testing and Evaluation | Threat and Vulnerability Analysis	3.1 Product Risk Management 4.3 Product Operation and Sustainment
12	Penetration Testing/Analysis	SA-11(5) Developer Security Testing and Evaluation | Penetration Testing/Analysis	3.5 Verification, Validation, and Testing 4.3 Product Operation and sustainment
13	Verifying Scope of Testing and Evaluation	SA-11(7) Developer Security Testing and Evaluation | Verify Scope of Testing/Evaluation	3.5 Verification, Validation, and Testing
14	Independent Verification of Assessment Plans/Evidence	SA-11(3) Developer Security Testing and Evaluation | Independent Verification of Assessment Plans/Evidence	3.5 Verification, Validation, and Testing

	Critical Software Cybersecurity Requirements	RMF Controls Addressed	SAF Practice Areas
15	Software Flaw Remediation	SI-2 Flaw Remediation SI-2(1) Flaw Remediation | Central Management SI-2(2) Flaw Remediation | Automated Flaw Remediation Status SI-2(3) Flaw Remediation | Time to Remediate Flaws/Benchmarks for Corrective Actions SI-2(6) Flaw Remediation | Removal of Previous Versions of Software/Firmware	2.4 Project Risk Management 3.4 Implementation 4.3 Product Operation and Sustainment
16	Malicious Code Protection	SI-3 Malicious Code Protection SI-3(1) Malicious Code Protection | Central Management SI-3(2) Malicious Code Protection | Automatic Updates SI-3(10) Malicious Code Protection | Malicious Code Analysis	2.4 Project Risk Management 3.4 Implementation 4.3 Product Operation and Sustainment
17	Software and Firmware Integrity	SI-7 Software, Firmware, and Information Integrity SI-7(1) Software, Firmware, and Information Integrity | Integrity Checks	1.2 Infrastructure Standards 2.5 Supplier Management 4.3 Product Operation and Sustainment
18	Software Supply Chain Protection	SA-12 Supply Chain Protection SA-12(1) Supply Chain Protection | Acquisition Strategies/Tools/Methods SA-12(5) Supply Chain Protection | Limitation of Harm SA-12(8) Supply Chain Protection | Use of All-Source Intelligence SA-12(9) Supply Chain Protection | Operations Security SA-12(11) Supply Chain Protection | Penetration Testing/Analysis of Elements, Processes, and Actors SA-22 Unsupported System Components	2.4 Project Risk Management 2.5 Supplier Management

Appendix B: SAF Process Management

SAF Practice Area 1.1 Process Definition: Does the program establish and maintain cybersecurity processes?

Activities/Practices	Outputs	Candidate Metrics
Establish and maintain a standard set of cybersecurity policies, laws, and regulations with which projects must comply.	Organizational Cybersecurity Policies	% of program managers trained in cybersecurity policy % of senior managers trained in cybersecurity policy # of updates to the organization's cybersecurity policy in the last year
Establish and maintain standard cybersecurity processes (including lifecycle models) that align with policies, laws, and regulations.	Organizational Cybersecurity Processes Organizational Cybersecurity Lifecycles	% cybersecurity policy requirements directly addressed in the organization's Cybersecurity Processes # and % of organization's applicable processes updated and integrated with the organization's Cybersecurity Processes % of organization's staff trained in the organization's updated processes that include cybersecurity # of Organizational Cybersecurity Lifecycles % current programs using Organizational Cybersecurity Lifecycles % current applicable staff trained in one or more Organizational Cybersecurity Lifecycles
Establish and maintain tailoring criteria and guidelines for the organization's cybersecurity processes (including lifecycle models).	Organizational Cybersecurity Tailoring Criteria and Guidelines	# and % of Organizational Cybersecurity Lifecycles with applicability and tailoring guidance # and % of applicable staff trained in Organizational Cybersecurity Tailoring Criteria and Guidelines

SAF Practice Area 1.2 Infrastructure Standards: Does the program establish and maintain security standards for its infrastructure?

Activities/Practices	Outputs	Candidate Metrics
Establish and maintain cybersecurity standards for information technology systems and networks.	Organizational Cybersecurity Standards	% Organizational Cybersecurity Standards planned vs actual % Organizational Cybersecurity Standards updated within the last year % Applicable personnel trained on Organizational Cybersecurity Standards planned vs actual
Establish and maintain physical security standards for physical work spaces and facilities...	Organizational Physical Security Standards	% Organizational Physical Security Standards planned vs actual % Organizational Physical Security Standards updated within the last year % Applicable personnel trained on Organizational Physical Security Standards planned vs actual

SAF Practice Area 1.3 Resources: Does the program have the cybersecurity resources (e.g., personnel, data, assets) it needs?

Activities/Practices	Outputs	Candidate Metrics
Establish and maintain standard cybersecurity process assets (e.g., procedures, tools) that align with processes and maintain them in a repository.	Organizational Cybersecurity Process Assets Security Resource Repository	% processes with supporting procedures % processes with supporting tools % staff trained in applicable processes and tools % processes changed/updated in last 12 months % tools changed/updated in last 12 months
Collect and maintain security-related intelligence data (e.g., attack data, vulnerabilities, design weaknesses, abuse/misuse cases, threats).	Security-Related Intelligence Data	Amount of applicable attack data # staff (planned vs actual) responsible for collecting, organizing, and maintaining security related intelligence data % applicable staff trained in collecting, organizing, and maintaining security related intelligence data Amount of resources (budget, tools, equipment) (planned vs actual) responsible for collecting, organizing, and maintaining security related intelligence data

Activities/Practices	Outputs	Candidate Metrics
Develop and document security features, frameworks, and patterns.	Approved Security Features, Frameworks, and Patterns	# (planned vs actual) approved Security Features, Frameworks, and Patterns Amount of time (planned vs actual) to develop and approve Security Features, Frameworks, and Patterns # disapproved Security Features, Frameworks, and Patterns # pending Security Features, Frameworks, and Patterns
Establish and maintain guidance for classifying data.	Data Management System	# data types # classification categories % data typed and classified # (planned vs actual) personnel responsible for maintaining Data Management System % applicable staff trained in Data Management System % applicable staff trained in classifying data
Provide specialized security experts to assist project personnel.	Security Roles and Responsibilities	# (planned vs actual) specialized security experts assigned to assist project personnel Budget (planned vs actual) for specialized security expert support # staff trained in use of specialized security experts % Security Roles and Responsibilities with specialized security experts assistance (% full, % partial, % no support)

SAF Practice Area 1.4 Training: Does the program provide security training for its personnel?

Activities/Practices	Outputs	Candidate Metrics
Provide security awareness training for program personnel (including vendors, contractors, and out-sourced workers).	Project Training Plan Training Products Vendor Contracts and Service Level Agreements	% project personnel trained % support personnel trained % contractor personnel trained % contracts and service level agreements with security aware-ness training requirement
Provide role-based security training for technical staff (including ven-dors, contractors, and outsourced workers).	Project Training Plan Training Products Vendor Contracts and Service Level Agreements	% project personnel trained % support personnel trained % contractor personnel trained % contracts and service level agreements with security role-based training requirement
Track completion of security train-ing activities.	Program Status Reports	% project personnel scheduled % project personnel scheduled to date vs completed % support personnel scheduled % support personnel scheduled to date vs completed % contractor personnel scheduled % contractor personnel scheduled to date vs completed % contracts and service level agreements with security training requirements

Appendix C: SAF Project Management

SAF Practice Area 2.1 Program Plans: Has the program adequately planned for cybersecurity activities?

Activities/Practices	Outputs	Candidate Metrics
Attend training for developing cybersecurity plans (for program managers and senior managers).	Training completed	% of program managers trained in cybersecurity planning % of senior managers trained in cybersecurity planning
Define and document cybersecurity objectives in the Program Plan.	Published Program Plan Published System Engineering Plan (SEP) Cybersecurity objectives defined and documented in the Program Plan or SEP	% cybersecurity objectives defined and documented in the Program Plan or SEP vs. the applicable number required in the organization's policies
Integrate cybersecurity tasks into the project plan.	Program Plan Documented cybersecurity tasks. Bi-directional traceability of Cybersecurity tasks to cybersecurity objectives. Cybersecurity tasks integrated with other program tasks into Program Plan	Traceability • Number of cybersecurity tasks without corresponding cybersecurity objectives • Number of cybersecurity objectives without corresponding cybersecurity tasks • % cybersecurity tasks integrated into the Program Plan
Define and assign cybersecurity roles and responsibilities.	Defined and documented cybersecurity roles and responsibilities. Cybersecurity roles and responsibilities assigned in Program Plan Completed Roles and Responsibilities Matrix	Completeness • Number of to be determined (TBD) and to be added (TBA) roles and responsibilities for cybersecurity in Program Plan Traceability • Number of cybersecurity tasks not mapped to cybersecurity roles and responsibilities • Number of cybersecurity roles and responsibilities without cybersecurity tasks • % cybersecurity roles and responsibilities assigned in the Program Plan

Activities/Practices	Outputs	Candidate Metrics
Provide adequate resources to implement planned cybersecurity tasks.	Required resources needed to complete program cybersecurity roles and responsibilities are identified and provided Funding for required resources is identified and provided Training is identified, scheduled, and provided Training procured or developed in-house Facilities identified, planned, and provided Equipment and tools identified and provided	For each category (personnel, training, facilities, equipment, and tools): % funding required vs approved % personnel positions filled % personnel positions open % training available % training procured vs developed % training scheduled % training complete % training complete by role # facilities not yet available and type % and type equipment not yet available % and type tools not yet available
Select and implement a secure software development lifecycle (SSDL).	Program Processes selected, developed, documented, trained, and maintained Process tailoring guidelines developed and applied Process waiver guidelines developed and applied	# and % program processes TBD # and % program processes added, changed, and deleted # and % program processes mapped to roles and responsibilities # and % program processes trained % processes with existing tailoring guidelines % processes tailored % processes with existing waiver guidelines % processes with requested waivers % requested waivers approved
Define and implement a project compliance initiative for cybersecurity.	Program Compliance Documents developed and maintained Roles and Responsibilities assigned Program compliance planned, scheduled, and initiated	% project compliance planning and scheduling completed % of project compliance planning and scheduling tasks behind schedule # of project compliance planning and scheduling tasks TBD

SAF Practice Area 2.2 Program Infrastructure: Is the program's infrastructure adequately secure?

Activities/Practices	Outputs	Candidate Metrics
Attend training for developing cybersecurity plans (for program managers and senior managers).	Project Cybersecurity Documentation Training provided	% of program managers trained in cybersecurity planning % of senior managers trained in cybersecurity planning
Establish and maintain the physical security of the project's physical work spaces and facilities.	Project Physical Security Documentation completed	% physical security objectives implemented for the project vs physical security objectives defined and documented in the Program Plan or SEP Number of physical security incidents per month Number and frequency of changes made to the Physical Security Documentation

SAF Practice Area 2.3 Program Monitoring Does the program monitor the status of cybersecurity activities?

Activities/Practices	Outputs	Candidate Metrics
Monitor the progress of the project's cybersecurity tasks.	Program Status Reports (monitor and control status against the Program Plan)	% scheduled tasks completed % tasks completed on schedule % tasks completed within budget # and percent tasks 10% over budget # and percent tasks 20% over budget Note: An EVM system could provide the above if implemented

Activities/Practices	Outputs	Candidate Metrics
Monitor project compliance with cybersecurity policies, laws, and regulations.	Program Compliance Documents Program Plan Program Master Schedule Roles and Responsibilities identified and assigned Program Compliance Audit Results	# project compliance audits completed # findings per audit by category % findings by per audit category # findings closed by category % findings closed by category Average time to close a finding by category # findings last audit # findings by category % findings by category # findings closed by category % findings closed by category Average time to close a finding by category
Conduct independent cybersecurity reviews of project tasks	Independent Review Results	# independent cybersecurity reviews completed % program tasks reviewed # findings per review by category % findings by per review category # findings closed by category % findings closed by category Average time to close a finding by category # findings last review % program tasks reviewed # findings by category % findings by category # findings closed by category % findings closed by category Average time to close a finding by category

SAF Practice Area 2.4 Program Risk Management: Does the program manage program-level cybersecurity risks?

Activities/Practices	Outputs	Candidate Metrics
Ensure that project strategies and plans address project-level cybersecurity risks (e.g., program risks related to cybersecurity resources and funding).	Program Plan Technology Development Strategy (TDS) Analysis of Alternatives (AoA)	% program managers receiving cybersecurity risk training % programs with cybersecurity related risk management plans
Identify and manage project-level cybersecurity risks (e.g., program risks related to cybersecurity resources and funding).	Risk Management Plan Risk Repository	% programs with cybersecurity related risks # cybersecurity related risks tracked per month

SAF Practice Area 2.5 Supplier Management: Does the program consider cybersecurity when selecting suppliers and managing their activities?

Activities/Practices	Outputs	Candidate Metrics
Integrate cybersecurity considerations (e.g., risks, compliance requirements) into the proposal process.	Acquisition Strategy Request for Proposal (RFP) Statement of Work (SOW) Software Development Plan (SDP) Integrated Master Plan (IMP)	# and % of Key acquisition documents that include supplier cybersecurity considerations/requirements
Define cybersecurity requirements for suppliers	Acquisition Strategy Request for Proposal (RFP) Statement of Work (SOW) Service Level Agreement (SLA)	# of cybersecurity requirements for suppliers in RFP # of cybersecurity requirements for suppliers in SOW/CDRL # of cybersecurity requirements for suppliers in SLA
Select suppliers based on their ability to meet specified cybersecurity requirements.	Source Selection Criteria	# of supplier cybersecurity related criteria in Source Selection Criteria Relative raking/importance of supplier cybersecurity related criteria in Source Selection Criteria
Provide oversight of cybersecurity activities that are performed by suppliers.	Program Management Documentation	% of Program Management Documentation (PMP, SOW, CDRL, IMP, SLA) containing monitoring/oversight requirements of supplier cybersecurity activities (including supplier monitoring/oversite activities of subs) # of supplier cybersecurity related monitoring/oversite requirements in Program Management Documentation (PMP, SOW, CDRL)

Activities/Practices	Outputs	Candidate Metrics
Conduct independent cybersecurity reviews of tasks being performed by suppliers.	Independent Review Results	# of independent reviews conducted per month # and % of independent reviews with significant findings per month Average time required to address/mitigate significant findings
Evaluate supplier deliverables against cybersecurity acceptance criteria.	Supplier Deliverables	# of cybersecurity related supplier deliverables # of recurring cybersecurity related supplier deliverables per month % of cybersecurity related supplier deliverables evaluated against acceptance criteria % of recurring cybersecurity related supplier deliverables per month evaluated against acceptance criteria # and % of cybersecurity related supplier deliverables rejected or with significant findings # and % of recurring cybersecurity related supplier deliverables per month rejected or with significant findings

Appendix D: SAF Engineering

SAF Area 3.2 Requirements: Does the program manage software security requirements?

Activities/Practices	Outputs	Candidate Metrics
Attend training for developing security requirements for software (for selected software engineers).	Training completed	% of software engineers trained in security requirements development
Conduct security risk analysis (includes threat modeling and abuse/misuse cases).	Prioritized list of software security risks Prioritized list of design weaknesses Prioritized list of controls/mitigations Mapping of controls/mitigations to design weaknesses	Number of software security risks controlled/mitigated (e.g., high and medium risks) Number of software security risks accepted/transferred Number of software security controls/mitigations selected for requirements development
Define and document software security requirements.	Documented software security requirements Traceability of software security requirements to controls/mitigations	Traceability ▪ Number of selected controls/mitigations without corresponding security requirements ▪ Number of security requirements traced to high and medium risks
Conduct reviews (e.g., peer reviews, inspections, and independent reviews) of software security requirements.	Defects identified in internal reviews	Completeness ▪ Number of to be determined (TBD) and to be added (TBA) items for software security requirements Correctness ▪ Number of software security requirements not validated ▪ % of software security requirements that have not been validated Understandability ▪ Number of software security requirements not understood by reviewers
Manage changes to software security requirements.	Change requests for software security requirements	Volatility ▪ Number of change requests for software security requirements ▪ % of software security requirements changed

SAF Practice Area 3.3 Architecture Does the program appropriately address cybersecurity in its software architecture and design?

Activities/Practices	Outputs	Candidate Metrics
Attend training for secure/resilient software architectures (for selected software engineers).	Training completed	% of software engineers trained in secure/resilient software architectures
Incorporate security requirements into software architecture.	Security features in architecture (e.g., authentication, access control, encryption, and auditing) Traceability of software security requirements to security features	Number of applicable security requirements not implemented in software architecture Number of security features without corresponding security requirements % of security requirements addressed by the architecture
Conduct security risk analysis of architecture.	Prioritized list of software architecture security risks Prioritized list of architecture design weaknesses Mapping of architecture security features to design weaknesses List of architecture design weaknesses without security controls/mitigations	Number of software security risks controlled/mitigated (e.g., high and medium risks) Number of software security risks accepted/transferred Number of architecture design weaknesses without security controls/mitigations
Address design weaknesses identified during architectural security risk analysis.	Security controls/mitigations implemented in software architecture	Number of architecture design weaknesses without security controls/mitigations
Conduct security reviews of software architecture (e.g., peer reviews, inspections, and independent reviews).	Security defects in software architecture identified in internal reviews	Number of security defects in software architecture
Manage security changes to software architecture.	Security change requests for software architecture	Number of security change requests for software architecture

SAF Practice Area 3.4 Implementation: Does the program minimize the number of vulnerabilities inserted into the code?

Activities/Practices	Outputs	Candidate Metrics
Secure coding standards are applied	Policy that requires the use of secure coding standards Contract language to ensure vendor(s) practices require use of secure coding standards	% of vendor contracts including requirements for the use of secure coding standards % of system developed using secure coding standards % of code verified for secure coding standard conformance

Activities/Practices	Outputs	Candidate Metrics
Code developers are trained in the use of secure coding standards	Competency standards for code developers require training in secure coding standards Hiring qualifications require training in secure coding standards Contract language requires use of developers trained in secure coding standards	% of software developers trained in secure coding standards % of code supported by developers trained in secure coding standards
Evaluation practices (e.g. code reviews and apply tools) are applied to identify and remove vulnerabilities in delivered code (including code libraries, open source, and other reused components)	Policy that requires the use of evaluation practices to identify and remove vulnerabilities and reporting of metrics Output of evaluations Corrections documented Contract language requires use of evaluation practices to identify and remove vulnerabilities and metrics reporting	% of vendor contracts requiring use of evaluation practices and reporting of vulnerability metrics Code coverage: % of code evaluated (total and by each type of review) Vulnerabilities per MLOC identified and removed Unaddressed vulnerabilities per MLOC % code libraries evaluated % open source evaluated % legacy components evaluated Count of high priority vulnerabilities identified and count of those removed

SAF Practice Area 3.5 Testing, Validation, and Verification: Does the program test, validate, and verify cybersecurity in its software components?

Activities/Practices	Outputs	Candidate Metrics
Develop cybersecurity test cases based on software requirements and risks and issues from prior agency/program/element experience	Cybersecurity related test cases based on software requirements, risks, and prior lessons learned Policy level and legal requirements included in test cases Requirements Traceability and Verification Matrix (RTVM)	Also build off of requirements metrics in addition to those provided below. Cybersecurity SW spec requirements in test spec • RTVM (% in test spec - RTVM) Policy level requirements (% addressed) Legal requirements (% addressed) Cybersecurity requirements tested successfully? • (% passed without issues) (% passed with issues) (% failed) (% tests to be rerun) (% problems open by category) (# problems open per category) (avg. time open per category) Number of test cases Average Number of test cases per program/function (normalized by size or function or function point or other) % requirements covered % requirements passed Defect Rates • Total number of defects • Categories of defects • Criticality (Low, Med, High) • Number by Criticality • Number by Criticality over time • Number remaining open by Criticality over time • Average time to correct a defect by Criticality over time • Total Time to fix defects by category over time
Perform a Software requirements based test coverage analysis	Software requirements based test coverage analysis results	% SW requirements covered in test
Perform a Code Coverage Data Flow analysis	Code Coverage Data Flow analysis results	# of code decision paths not exercised % of code decision paths not exercised
Perform a Software structural test coverage analysis	Software structural test coverage analysis results	% of code not exercised % of code not accessible # of functions not exercised

Activities/Practices	Outputs	Candidate Metrics
Perform a Functional Test Coverage analysis • Stress Test • Test Cases	Functional Test Coverage analysis results	# functions tested % functions tested # functions stress tested % functions stress tested # test cases per function Avg. # test cases per function
Perform Regression Testing on all code impacted by SW changes • How Much • Test Cases	Regression Testing results	# changes # regression tests # test cases per regression test % SW tested Size SW tested Time to perform each regression test Avg. time to perform regression tests # defects inserted by category based on SW changes Defect density based on SW changes
Perform Peer Reviews of select test products throughout the SW life cycle	Peer Review results	# products peer reviewed % products peer reviewed # defects removed by category Avg. number of defects removed by category
Perform Independent Reviews of select test products throughout the SW life cycle	Independent Review results	# products independently reviewed % products independently reviewed # defects removed by category Avg. number of defects removed by category
Perform a SW requirements analysis to determine which are to be verified using Modeling and Simulation (M&S)	Modeling and Simulation verification analysis results Modeling & Simulation Test Cases for • Flight Test • Ground Test	# SW requirements to be verified using M&S % SW requirements to be verified using M&S % safety SW requirements to be verified using M&S % mission critical SW requirements to be verified using M&S

Activities/Practices	Outputs	Candidate Metrics
Perform a detailed resources analysis to determine system level capacity	Resources usage analysis results	% CPU Usage during peak performance and stress % memory Usage during peak performance and stress Avg. Time to Restore System to baseline state Total Time to Restore System to baseline state
Verify coding standards have been followed	Coding standards verification results	Number of code standard violations Number of code standard violations per module SW Complexity per module Avg. SW Complexity % modules in each complexity category Number of Memory Leaks Number of Memory Usage Issues Avg. Number of Memory Leaks per SCI Avg. Number of Memory Usage Issues per SCI
Conduct cybersecurity test readiness reviews as part of a Test Readiness Review (TRR)	Cybersecurity test readiness review results An indication of extent of Bi-directional traceability provided between requirements under test and test cases and test procedures in which requirements will be verified (see also RTVM first row) An indication of extent of Bi-directional traceability provided between SW requirements specs and SW requirements under test (see also RTVM first row)	Number of issues per Readiness Review Avg. Number of issues per Readiness Review Number of issues per Readiness Review per hour (or normalized by some size measure) % SW requirements with Bi-directional traceability provided between requirements under test and test cases and test procedures in which requirements will be verified % SW requirements with Bi-directional traceability provided between SW requirements specs and SW requirements under test % SW requirements with full test coverage % SW requirements with partial test coverage % SW requirements with no test coverage

Activities/Practices	Outputs	Candidate Metrics
Perform functional and risk-based cybersecurity testing for selected software components at various levels of integration	Functional and risk-based cybersecurity testing of the integrated system Functional and risk-based cybersecurity testing results at lower levels of integration • Test results at various levels of integration An indication of which levels of integration were not tested and why	%Safety components tested and % passed and % open over time %Mission critical components tested and % passed and % open over time # of levels of integration where tests were performed % levels of integration where tests were performed Number of functional tests and % passed and % open Number of risk-based tests and % passed and % open over time % levels of integration tested
Perform operational security testing for the integrated system	Operational security testing results	Number of operational security test cases completed and % passed and % open # of total issues by category # of total issues by criticality # of open issues by category # of open issues by criticality % of total issues open by category % of total issues open by criticality
Red Team Assessments have been completed and results addressed	Completed Red Team Assessment results Report on issues and how they were or will be addressed	# of total red team findings by category # of total red team findings by criticality # of open red team findings by category # of open red team findings by criticality % of total red team findings open by category % of total red team findings open by criticality

Activities/Practices	Outputs	Candidate Metrics
SW scanned for Vulnerabilities using Scanning Tools • Scanning Tools’ Capabilities are known Dynamic/Static Analysis • Developer • Independent	Identified vulnerabilities from performed scans Coverage analysis available Dynamic/Static Analysis results available	Confidence levels in results categorized (very high, high, medium, low, very low) % results in each confidence category Coverage analysis metrics TBD % operational code scanned for vulnerabilities % of known Vulnerabilities Covered (Scanning Tool Capabilities) # vulnerabilities by category # vulnerabilities by criticality % vulnerabilities addressed by category % vulnerabilities addressed by criticality % scanned Dynamic Analysis % scanned Static Analysis
Perform independent cybersecurity validation of selected components	Independent validation results	% components validated Number of validation test cases completed and % passed and % open # of total issues by category # of total issues by criticality # of open issues by category # of open issues by criticality % of total issues open by category % of total issues open by criticality Avg. Time issues open by category Avg. Time issues open by criticality Number of defects per LOC per hour (or some other normalization)

Activities/Practices	Outputs	Candidate Metrics
Perform independent cybersecurity verification of selected components?	Independent verification results	% components verified Number of verification test cases completed and % passed and % open # of total issues by category # of total issues by criticality # of open issues by category # of open issues by criticality % of total issues open by category % of total issues open by criticality Avg. Time issues open by category Avg. Time issues open by criticality Number of defects per LOC per hour (or some other normalization)
Review/inspect Test procedures for compliance with test plans and descriptions, adequacy to accomplish test requirements, and satisfying subsystem specification requirements for verifications	Review/inspection results	# issues identified # retests % tests redone Total retest time

Appendix E: SAF Support

SAF Practice Area 4.1 Measurement and Analysis: Does the program adequately measure cybersecurity in acquisition and engineering activities?

Activities/Practices	Outputs	Candidate Metrics
Define and improve cybersecurity measures.	Published Program Plan Program Status Reports	# cybersecurity measures defined # and % cybersecurity measures implemented % cybersecurity measures improved over time
Collect and analyze cybersecurity measures..	Published Program Plan Program Status Reports	# and % cybersecurity measures collected and analyzed

SAF Practice Area 4.2 Change Management: Does the program manage cybersecurity changes to its acquisition and engineering activities?

Activities/Practices	Outputs	Candidate Metrics
Incorporate cybersecurity changes into the strategy and plan documents and artifacts.	Change Requests Configuration/Change Management System Updated cybersecurity related plans	# change requests related to cybersecurity # and % changes incorporated in existing cybersecurity related plans and other artifacts
Incorporate cybersecurity changes into the engineering documents and artifacts.	Change Requests Configuration/Change Management System Updated engineering documents and artifacts	# change requests related to cybersecurity # and % changes incorporated in existing cybersecurity related engineering documents and other artifacts

SAF Practice Area 4.3 Product Operation and Sustainment: Does the organization responsible for operating and sustaining the software-reliant system manage vulnerabilities and cybersecurity risks?

Activities/Practices	Outputs	Candidate Metrics
Perform detailed cybersecurity risk analyses of operational systems	Operational Risk Management Plan Operational Risk Repository Established definition of Cat 1, 2, 3 Risks	# Cat 1 risks per month # new Cat 1 risks per month # Cat 2 risks per month # new Cat 2 risks per month

Activities/Practices	Outputs	Candidate Metrics
Assess cybersecurity during maintenance testing.	Maintenance Testing Results Established definition of Cat 1, 2, 3 defects	# of new defects per month # defects per line of code Avg. time to close a defect # of new Cat 1 defects per month # Cat 1 defects per line of code Avg. time to close a Cat 1 defect # of new Cat 2 defects per month # Cat 2 defects per line of code Avg. time to close a Cat 2 defect
Conduct periodic penetration testing of all software to identify cybersecurity vulnerabilities.	Penetration Testing Results Relationship to 4.3.1 above	# of vulnerabilities per month # of vulnerabilities remediated per month # of new vulnerabilities per month
Conduct deep-dive penetration testing of critical software to identify cybersecurity vulnerabilities.	Penetration Testing Results Relationship to 4.3.1 above	# of vulnerabilities per month # of vulnerabilities remediated per month # of new vulnerabilities per month
Run vulnerability scanning tools on operational systems.	Vulnerability Management Reports Relationship to 4.3.1 above	# of vulnerabilities per month # of vulnerabilities remediated per month # of new vulnerabilities per month
Remediate identified cybersecurity vulnerabilities and risks.	Defect Management System Relationship to 4.3.1 above	# of vulnerabilities per month # of vulnerabilities remediated per month # of new vulnerabilities per month
Monitor the behavior of operational software/systems to identify signs of attack.	Software Monitoring Results Relationship to 4.3.1 above	# attacks per month (may need to be more frequent) # false positive attacks per month (may need to be more frequent) # successful attacks per month (may need to be more frequent) # times per month necessary to restore system to operational state Avg. time to restore system to operational state # attacks per month discovered at a future time (may need to be more frequent)

Activities/Practices	Outputs	Candidate Metrics
Respond to cybersecurity incidents as appropriate	Incident Response Ticketing System	# attacks per month (may need to be more frequent) # incident response tickets per month (may need to be more frequent) # incident response tickets closed per month (may need to be more frequent) Avg. time to close incident response tickets per month (may need to be more frequent) # false positive attacks per month (may need to be more frequent) # successful attacks per month (may need to be more frequent) # times per month necessary to restore system to operational state Avg. time to restore system to operational state # attacks per month discovered at a future time (may need to be more frequent)
Ensure the ability to roll back to a previous version of the system when needed and maintain the expected level of cybersecurity.	Configuration/Change Management System	# times per month necessary to restore system to operational state Avg. time to restore system to operational state # changes per month to Configuration/Change Management System Avg. time to complete change to Configuration/Change Management System (by month)
Communicate suggested product changes or improvements related to cybersecurity to the engineering team.	Field Change Requests Configuration/Change Management System	# changes per month suggested per product # changes per month accepted per product Avg. time to complete change by product per month Avg. time to complete change to Configuration/Change Management System (by month) # changes per month to Configuration/Change Management System Avg. time to complete change to Configuration/Change Management System per month

References/Bibliography

URLs are valid as of the publication date of this document.

[Agrawal 2013]
Agrawal, A. & Khan, R. A. Software security metric development framework (an early stage approach). *American Journal of Software Engineering and Applications*. Volume 2. Issue 6. December 2013. Pages 150–155. http://www.sciencepublishinggroup.com/journal/paperinfo?journalid=137&doi=10.11648/j.ajsea.20130206.14

[Ahmad 2007]
Ahmad, N. & Laplante, P. A. Employing expert opinion and software metrics for reasoning about software. Pages 119–124. In *Proceedings of the Third IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC 2007).* September 2007. http://ieeexplore.ieee.org/document/4351396/

[Alberts 2017]
Alberts, Christopher J. & Woody, Carol C. *Prototype Software Assurance Framework (SAF): Introduction and Overview*. CMU/SEI-2017-TN-001. 2017. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=496134

[Allen 2007]
Allen, Edward B.; Gottipati, Sampath; & Govindarajan, Rajiv. Measuring size, complexity, and coupling of hypergraph abstractions of software: An information-theory approach. *Software Quality Journal.* Volume 15. Issue 2. June 2007. Pages 179–212. http://dl.acm.org/citation.cfm?id=1232684.1232687

[Almutairi 2013]
Almutairi, Abdulrahman; Shawly, Tawfeeq A.; Basalamah, Saleh M.; & Ghafoor, Arif. Policy-Driven High Assurance Cyber Infrastructure-Based Systems. Pages 146–153. In *Proceedings of the 2014 IEEE 15th International Symposium on High-Assurance Systems Engineering (HASE 2014).* January 2014. http://ieeexplore.ieee.org/document/6754599/

[An 2012]
An, W.; Jiang, C.; Lin, J.; Zhang, X.; & Yuan, W. GB/T 20274-based information system security technical assurance evaluation and computer realization. *Huadong Ligong Daxue Xuebao/Journal of East China University of Science and Technology*. Volume 38. Issue 5. October 2012. Pages 645-651.

[Azuwa 2012]
Azuwa, M. P.; Ahmad, R.; Sahib, S.; & Shamsuddin, S. A propose technical security metrics model for SCADA systems. Pages 70–75. In *Proceedings of the 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec 2012).* June 2012. doi:10.1109/CyberSec.2012.6246089. http://ieeexplore.ieee.org/document/6246089/

[Barabanov 2011]
Barabanov, Rostyslav; Kowalski, Stewart [ed.]; & Yngström, Louise [ed.]. *Information Security Metrics: State of the Art.* DSV Report Series No 11-007. March 2011 https://www.diva-portal.org/smash/get/diva2:469570/FULLTEXT01.pdf

[Black 2007a]
Black, Paul E.. Software Assurance With SAMATE Reference Dataset, Tool Standards, and Studies. Pages 6C11–6C16. In *Proceedings of the 26th AIAA/IEEE Digital Avionics Systems Conference (DASC 2007).* October 2007. http://ieeexplore.ieee.org/document/4391957/

[Black 2007b]
Black, Paul E. SAMATE and Evaluating Static Analysis Tools. *Ada User Journal*. Volume 28. Issue 3. September 2007. Pages 184–188. https://hissa.nist.gov/~black/Papers/staticAnalyExper%20Ada%20Geneva%20Jun%20007.pdf

[Black 2011]
Black, Paul E. Counting Bugs is Harder than You Think. Pages 1–9. In *Proceedings of the 11th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2011).* September 2011. http://ieeexplore.ieee.org/document/6065191/

[Black 2012]
Black, Paul E. Static Analyzers: Seat Belts for Your Code. *IEEE Security & Privacy.* Volume 10. Issue 3. May–June 2012. Pages 48–52. http://ieeexplore.ieee.org/document/6127855/

[Bowen 2011]
Bowen, Brian M.; Devarajan, Ramswamy; & Stolfo, Salvatore. Measuring the human factor of cyber security. Pages 230–235. In *Proceedings of the 2011 IEEE International Conference on Technologies for Homeland Security (HST 2011).* November 2011. http://ieeexplore.ieee.org/document/6107876/

[Breier 2014]
Breier, Jakub. Security Evaluation Model Based on the Score of Security Mechanisms. *Information Sciences and Technologies Bulletin of the ACM Slovakia.* Volume 6. Number 1. 2014. Pages 19–27. http://acmbulletin.fiit.stuba.sk/vol6num1/breier2014.pdf

[Brotby 2013]
Brotby, W. Krag & Hinson, Gary. *PRAGMATIC Security Metrics: Applying Metametrics to Information Security*. Auerbach Publications. January 2013. ISBN-13: 1439881529.

[CISQ 2012]
Consortium for IT Software Quality. *CISQ Specifications for Automated Quality Characteristic Measures*. CISQ–TR–2012–01. Consortium for IT Software Quality. 2012. http://it-cisq.org/wp-content/uploads/2012/09/CISQ-Specification-for-Automated-Quality-Characteristic-Measures.pdf

[CNSS 2010]
Committee on National Security Systems. National Information Assurance (IA) Glossary CNSS Instruction. CNSS Instruction No. 4009. Fort George G. Meade, MD: s.n. 2010.

[DAU 2017]
Defense Acquisition Guidebook. *Defense Acquisition University (DAU) Website.* March 2017 [accessed]. https://www.dau.mil/tools/dag

[Da-wei 2007]
Da-wei, E. The Software Complexity Model and Metrics for Object-Oriented. Pages 464–469. In *Proceedings of the 2007 IEEE International Workshop on Anti-Counterfeiting, Security, and Identification (ASID 2007)*. April 2007. http://ieeexplore.ieee.org/document/4244872/

[Delaitre 2013]
Delaitre, Aurelien; Okun, Vadim; & Fong, Elizabeth. Of Massive Static Analysis Data. Pages 163–167. In *Proceedings of the 7th International Conference on Software Security and Reliability Companion (SERE-C 2013).* June 2013. http://ieeexplore.ieee.org/document/6616339/

[Ellison 2014]
Ellison, Robert J *Assuring Software Reliability.* CMU/SEI-2014-SR-008. Software Engineering Institute. Carnegie Mellon University. 2014. https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=301625

[Feiler 2012]
Feiler, Peter; Goodenough, John; Gurfinkel, Arie; Weinstock, Charles; & Wrage, Lutz. *Reliability Improvement and Validation Framework*. CMU/SEI-2012-SR-013. Software Engineering Institute. Carnegie Mellon University. 2012. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=34069

[Garousi 2013]
Garousi, Vahid & Zhi, Junji. A survey of software testing practices in Canada. *Journal of Systems and Software*. Volume 86. Issue 5. May 2013. Pages 1354–1376. http://dx.doi.org/10.1016/j.jss.2012.12.051

[Gaudan 2008]
Gaudan, Stéphanie; Motet, Gilles; & Auriol, Guillaume. Metrics for Object-Oriented Software Reliability Assessment – Application to a Flight Manager. Pages 13–24. In *Proceedings of the Seventh European Conference on Dependable Computing (EDCC 2008).* May 2008. http://ieeexplore.ieee.org/document/4555986/

[Geer 2003]
Geer, Daniel; Hoo, Kevin Soo; & Jaquith, Andrew. Information security: Why the future belongs to the quants. *IEEE Security and Privacy*. Volume 99. Issue 4. July–August 2003. Pages 24–32. http://ieeexplore.ieee.org/document/1219053/

[Geer 2011]
Geer, Daniel. *Index of Cyber Security website*. 2011. http://cybersecurityindex.org/

[Goodenough 2010]
Goodenough, J. B. *Evaluating Software's Impact on System and System of Systems Reliability*. Software Engineering Institute. Carnegie Mellon University. March 2010. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=28933

[Haddad 2011]
Haddad, S.; Dubus, S.; Hecker, A.; Kanstrén, T.; Marquet, B.; & Savola, R. Operational security assurance evaluation in open infrastructures. In *Proceedings of the 6th International Conference on Risk and Security of Internet and Systems (CRiSIS 2011).* September 2011. http://ieeexplore.ieee.org/document/6061831/

[Hallberg 2009]
Hallberg, J. & Lundholm, K. Information security metrics based on organizational models. Linköping: FOI, Swedish Defence Research Agency. 2009.

[Harrington 2013]
Harrington, David; Montville, Adam; & Waltermire, David. Using Security Posture Assessment to Grant Access to Enterprise Network Resources. *Analysis.* 2013.

[Heinzle 2013]
Heinzle, Bernhard & Furnell, Steven. Assessing the Feasibility of Security Metrics. Pages 149–160. In *Proceedings of the 10th International Conference on Trust, Privacy, and Security in Digital Business (TrustBus 2013).* Volume 805. August 2013. http://dl.acm.org/citation.cfm?id=2954132.2954150

[Hendradjaya 2007]
Hendradjaya, B. & Narasimhan, V. L. Some theoretical considerations for a suite of metrics for the integration of software components. *Information Sciences.* Volume 177. Issue 3. February 2007. Pages 844–864.

[Herrmann 2007]
Herrmann, Debra S. *Complete Guide to Security and Privacy Metrics. Measuring Regulatory Compliance, Operational Resilience, and ROI.* Auerbach Publications. 2007. ISBN-13: 978-0849354021.

[Hindle 2008]
Hindle, A.; Godfrey, M.W.; & Holt, R.C. Pages 133–42. Reading beside the lines: indentation as a proxy for complexity metrics. In *Proceedings of the IEEE 16th International Conference on Program Comprehension (ICPC 2008).* June 2008. http://ieeexplore.ieee.org/document/4556125/

[Howard 2006]
Howard, Michael & Lipner, Steve. *The Security Development Lifecycle*. Microsoft Press. 2006.

[Izurieta 2013]
Izurieta, C.; Griffith, I; Reimanis, D.; & Luhr, R. On the Uncertainty of Technical Debt Measurements. Pages 1–4. In Proceedings of the *2013 International Conference on Information Science and Applications (ICISA 2013).* June 2013. http://ieeexplore.ieee.org/document/6579461/

[Jaquith 2007]
Jaquith, Andrew. *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Addison-Wesley. 2007. ISBN-13: 978-0321349989.

[Jones 2011]
Jones, Caper and Bonsignour, Oliver. *The Economics of Software Quality. s.l.* Addison-Wesley Professional. 2011.

[Jones 2012]
Jones, Capers. *Software Quality in 2012: A Survey of the State of the Art.* Nancook Analytics LLC. May 2012. http://sqgne.org/presentations/2012-13/Jones-Sep-2012.pdf

[Jones 2014]
Jones, Capers. *Evaluating Software Metrics and Software Measurement Practices*. Version 4. Namcook Analytics LLC. March 2014.

[Jonsson 2013]
Jonsson, Erland; & Pirzadeh, Laleh. Identifying Suitable Attributes for Security and Dependability Metrication. Pages 1–7. In *Proceedings of the Seventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2013)*. August 2013. http://publications.lib.chalmers.se/records/fulltext/182820/local_182820.pdf

[Kupsch 2014]
Kupsch, James A. & Miller, Barton P. *Why Do Software Assurance Tools Have Problems Finding Bugs Like Heartbleed?* WP003. Software Assurance Marketplace. April 2014. https://www.swampinabox.org/doc/SWAMP-WP003-Heartbleed.pdf

[Martin 2007]
Martin, C. & Refai, M. A Policy-Based Metrics Framework for Information Security Performance Measurement. Pages 94–101. In *Proceedings of the 2nd IEEE International Workshop on Business-Driven IT Management*. BDIM 2007. May 2007. http://ieeexplore.ieee.org/document/4261105/

[McCabe 1976]
McCabe, Thomas J. A complexity measure. *IEEE Transactions on Software Engineering.* Volume SE-2. Issue 4. Pages 308–320. December 1976. http://ieeexplore.ieee.org/document/1702388/

[Misra 2008]
Misra, Sanjay & Akman, Ibrahim, A unique complexity metric. In *Proceedings of the 8th International Conference on Computational Science and Its Applications (ICCSA 2008)*. June – July 2008. https://www.researchgate.net/publication/221433680_A_Unique_Complexity_Metric

[NERC 2004]
North American Electric Reliability Corporation. *Technical Analysis of the August 14, 2003 Blackout.* July 2004. http://www.nerc.com/docs/docs/blackout/NERC_Final_Blackout_Report_07_13_04.pdf

[NIST 2014]
Joint Task Force Transformation Initiative. *Guide for Applying the Risk Management Framework to Federal Information Systems: A Security Life Cycle Approach*. SP 800-37 Rev. 1. February 2010 (Updated June 5, 2014). https://csrc.nist.gov/publications/detail/sp/800-37/rev-1/final

[Nord 2008]
Nord, R. L.; Ozkaya, I; Kruchten, P.; & Gonzalez-Rojas, M. In Search of a Metric for Managing Architectural Technical Debt. Pages 91–100. In *Proceedings of the 2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*. August 2012. http://ieeexplore.ieee.org/document/6337765/

[Ohlhausen 2014]
Ohlhausen, P.; Poore, M.; McGarvey, D.; & Anderson, L. *Persuading Senior Management with Effective, Evaluated Security Metrics.* ASIS Foundation. 2014. https://capindex.com/wp-content/uploads/ASIS_Report_Complete1.pdf

[Olague 2008]
Olague, H. M.; Etzkorn, L. H.; Messimer, S. L.; & Delugach, H. S. An empirical validation of object-oriented class complexity metrics and their ability to predict error-prone classes in highly iterative, or agile, software: a case study. *Journal of Software Maintenance and Evolution: Research and Practice*. Volume 20. Issue 3. May 2008. Pages 171–197. http://dl.acm.org/citation.cfm?id=1379052.1379053

[Ouedraogo 2012a]
Ouedraogo, M. Towards security assurance metrics for service systems security. *Lecture Notes in Business Information Processing*. Volume 103. 2012. Pages. 361–370. ISSN: 1865-1348. http://link.springer.com/chapter/10.1007%2F978-3-642-28227-0_28

[Ouedraogo 2012b]
Ouedraogo, M.; Khadraoui, D.; Mouratidis, H.; & Dubois, E. Appraisal and reporting of security assurance at operational systems level. *Journal of Systems and Software*. Volume 85. Issue 1. January 2012. Pages 193–208. http://dl.acm.org/citation.cfm?id=2064367

[Ouedraogo 2013]
Ouedraogo, M.; Savola, R.M.; Mouratidis, H.; Preston, D.; Khadraoui, D.; & Dubois, E. Taxonomy of quality metrics for assessing assurance of security correctness. *Software Quality Journal*. Volume 21. Issue 1. March 2013. Pages 67–97. http://dl.acm.org/citation.cfm?id=2431353.2431368

[Paul 2002]
Paul, R. A.; Kunii, T.; Shinagawa, Y.; & Khan, M. F. Software metrics knowledge and databases for project management. *IEEE Transactions on Knowledge and Data Engineering.* Volume 11. Number 1. August 2002. Pages 255–264. http://ieeexplore.ieee.org/document/755633/

[Pieters 2012]
Pieters, Wolter; van der Ven, Sanne H. G.; & Probst, Christian W. A move in the security measurement stalemate: elo-style ratings to quantify vulnerability. Pages 1–14. In *Proceedings of the 2012 Workshop on New Security Paradigms (NSPW 2012).* September 2012. http://dl.acm.org/citation.cfm?doid=2413296.2413298

[Savola 2010]
Savola, R.M.; Kanstrén, T.; & Evesti, A. First International Workshop on Measurability of Security in Software Architectures—MeSSa 2010. Pages 151–154. In *Proceedings of the Fourth European Conference on Software Architecture (ECSA 2010): Companion Volume.* August 2010. http://dl.acm.org/citation.cfm?doid=1842752.1842785

[Savola 2011]
Savola, R. M. & Heinonen, P. A visualization and modeling tool for security metrics and measurements management. Pages 1–8. In *Proceedings of Information Security South Africa (ISSA 2011)*. August 2011. http://ieeexplore.ieee.org/document/6027518/

[Sedigh-Ali 2001]
Sedigh-Ali, S.; Ghafoor, A.; & Paul, R. A. Software engineering metrics for COTS-based systems. *IEEE Computer*. Volume 34. Number 5. May 2001. Pages 44–50. http://ieeexplore.ieee.org/document/920611/

[Ting 2010]
Ting, W. W. & Comings, D. R. Information Assurance Metric for Assessing NIST's Monitoring Step in the Risk Management Framework. *Information Security Journal: A Global Perspective.* Volume 19. Issue 5. January 2010. Pages 253–262. http://dl.acm.org/citation.cfm?id=1882988

[Vaughn 2003]
Vaughn, R. B., Jr.; Henning, R.; & Siraj, A. Information assurance measures and metrics – state of practice and proposed taxonomy. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences.* January 2003. http://ieeexplore.ieee.org/document/1174904/

[Wedyan 2009]
Wedyan, F.; Alrmuny, D; Bieman, J.M. *The Effectiveness of Automated Static Analysis Tools for Fault Detection and Refactoring Prediction*. International Conference on Software Testing Verification and Validation. 2009. https://ieeexplore.ieee.org/document/4815346

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	December 2018	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Exploring the Use of Metrics for Software Assurance	FA8702-15-D-0002

6. **AUTHOR(S)**

Carol Woody, Robert Ellison, & Charlie Ryan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2018-TN-004

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100	n/a

11. **SUPPLEMENTARY NOTES**

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. abstract (maximum 200 words)

The Software Assurance Framework (SAF) is a collection of cybersecurity practices that programs can apply across the acquisition lifecycle and supply chain. The SAF can be used to assess an acquisition program's current cybersecurity practices and chart a course for improvement, ultimately reducing the cybersecurity risk of deployed software-reliant systems.

This report proposes measurements for each SAF practice that a program can select to monitor and manage the progress it's making toward software assurance. Metrics are needed to determine how effectively a practice is performed and how well software assurance is addressed. This report presents an approach for determining which SAF practices should be measured and how. It provides acquirers, program managers, and contractors with an approach for using metrics to establish confidence that the systems they plan to field will have sufficient software assurance.

14. SUBJECT TERMS	15. NUMBER OF PAGES
software assurance framework, SAF, cybersecurity, supply chain, acquisition, software-reliant systems	69

16. **PRICE CODE**

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL