NATO STANDARD

AEP-67

ENGINEERING FOR SYSTEM ASSURANCE IN NATO PROGRAMMES

Edition B Version 1
OCTOBER 2017

NORTH ATLANTIC TREATY ORGANIZATION

ALLIED ENGINEERING PUBLICATION

Published by the
NATO STANDARDIZATION OFFICE (NSO)

INTENTIONALLY BLANK

NORTH ATLANTIC TREATY ORGANIZATION (NATO)

NATO STANDARDIZATION OFFICE (NSO)

NATO LETTER OF PROMULGATION

23 October 2017

1. The enclosed Allied Engineering Publication AEP-67, Edition B, Version 1 ENGINEERING FOR SYSTEM ASSURANCE IN NATO PROGRAMMES, which has been approved by the nations in the Life Cycle Management Group (AC/327), is promulgated herewith. The recommendation of nations to use this publication is recorded in STANREC 4808.

2. AEP-67, Edition B, Version 1 is effective upon receipt and supersedes AEP-67 , Edition 1 which shall be destroyed in accordance with the local procedure for the destruction of documents.

3. No part of this publication may be reproduced, stored in a retrieval system, used commercially, adapted, or transmitted in any form or by any means, electronic, mechanical, photo-copying, recording or otherwise, without the prior permission of the publisher. With the exception of commercial sales, this does not apply to member or partner nations, or NATO commands and bodies.

4. This publication shall be handled in accordance with C-M(2002)60.

[illegible]

Edvardas MAŽEIKIS
Major General, LTUAF
Director, NATO Standardization Office

INTENTIONALLY BLANK

RESERVED FOR NATIONAL LETTER OF PROMULGATION

INTENTIONALLY BLANK

RECORD OF RESERVATIONS

CHAPTER	RECORD OF RESERVATION BY NATIONS
Note: The reservations listed on this page include only those that were recorded at time of promulgation and may not be complete. Refer to the NATO Standardization Document Database for the complete list of existing reservations.	

INTENTIONALLY BLANK

RECORD OF SPECIFIC RESERVATIONS

[nation]	[detail of reservation]
Note: The reservations listed on this page include only those that were recorded at time of promulgation and may not be complete. Refer to the NATO Standardization Document Database for the complete list of existing reservations.	

INTENTIONALLY BLANK

TABLE OF CONTENTS

Table of Contents

Figures

Tables

Executive Summary

NATO Programmes represent cooperative, joint, multi-national, and commonly funded efforts realized by the integration of the necessary capability components (Doctrine, Organisation, Training, Material, Leadership Development, Personnel, Facilities and Interoperability), and utilization of a systems life cycle approach. Utilizing this approach the programme managers take into account performance, cost, schedule, quality, operational environments, integrated logistics, and obsolescence. Risk management is a process that is at the core of all those processes and as part of risk management, system assurance must be thoroughly examined to achieve a successful programme. Programmes try to build affordable, secure, and trustworthy systems and despite significant strides toward this goal, there is ample evidence that adversaries retain their ability to compromise systems. Adversaries have a broad arsenal that includes:

- Insider attacks
- Social engineering attacks
- Physical attacks
- Attacks on the global supply chain
- Cyber attacks

Their attacks are often very hard to prevent or even detect. Consequently, there is growing awareness that systems must be designed, built, and operated with the expectation that system elements will have known and unknown vulnerabilities. Systems must continue to meet (possibly degraded) functionality, performance, and security goals despite these vulnerabilities. In addition, vulnerabilities should be addressed through a more comprehensive life cycle approach, delivering system assurance, reducing these vulnerabilities through a set of technologies and processes.

System assurance is *the justified confidence that the system functions as intended and is free of exploitable vulnerabilities, either intentionally or unintentionally designed or inserted as part of the system at any time during the life cycle*. This ideal of no exploitable vulnerabilities is usually unachievable in practice, so programmes must perform risk management to reduce the probability and impact of vulnerabilities to acceptable levels.

This confidence is achieved by system assurance activities, which include a planned, systematic set of multi-disciplinary activities to achieve the acceptable measures of system assurance and manage the risk of exploitable vulnerabilities. The assurance case is the enabling mechanism to show that the system will meet its prioritized requirements, and that it will operate as intended in the operational environment, minimizing the risk of being exploited through weaknesses and vulnerabilities. It is a means to identify all the assurance claims, and from those claims trace through to their supporting arguments, and from those arguments to

the supporting evidence. The assurance case is a critical mechanism for supporting the risk management process.

Today, the risk management process has often not considered assurance issues in an integrated way, resulting in project stakeholders unknowingly accepting assurance risks that can have severe financial, legal, and national security implications. Addressing assurance issues late in the process, or in a non-integrated way, may result in the system not being used as intended, or in excessive costs or delays in system certification or accreditation. Risk management must consider the broader consequences of a failure (e.g., to the mission, people's lives, property, business services, or reputation), not just the failure of the system to operate as intended. In many cases, layered defenses (e.g., defense-in-depth or engineering-in-depth) may be necessary to provide acceptable risk mitigation.

This guidebook provides process and technology guidance to increase the level of system assurance. This guidebook is intended primarily to aid programme managers (PMs) and systems engineers (SEs) who are seeking guidance on how to incorporate assurance measures into their system life cycles. Assurance for security must be integrated into the systems engineering activities to be cost-effective, timely, and consistent. In systems engineering, the activities for developing and maintaining the assurance case enable rational decision making, so that only the actions necessary to provide adequate justification (arguments and evidence) are performed. This guidebook is a synthesis of knowledge gained from existing practices, recommendations, policies, and mandates. System assurance activities are executed throughout the system life cycle.

This guidebook is organized based on ISO/IEC 15288:2008, Systems and software engineering – System life cycle processes. While there are other life cycle frameworks, this ISO/IEC standard combines a suitably encompassing nature while also providing sufficient specifics to drive system assurance. Those who use other life cycle frameworks should find it easy to map their frameworks to the ISO framework. Unless otherwise noted, cross-references refer to sections of this guidebook.

1 Introduction

1.1 Background

Despite significant strides in developing secure systems, adversaries continue to compromise systems. Adversaries have a broad arsenal of attacks that they can use during the development and operation of systems, including:

- Insider attacks
- Social engineering attacks
- Physical attacks
- Attacks on the global supply chain
- Cyber attacks

Developing effectively secure systems requires a combination of approaches to ensure mission success, including minimizing vulnerabilities and managing the remaining vulnerabilities. These approaches should be driven by threat analysis and oriented to mission success—in other words, they should address the prioritized threats that will most impact mission success.

1.2 Definition of System Assurance

For the purposes of this guidebook, system assurance (SA) is *the justified confidence that the system functions as intended and is free of exploitable vulnerabilities, either intentionally or unintentionally designed or inserted as part of the system at any time during the life cycle*. This ideal of no exploitable vulnerabilities is usually unachievable in practice, so programmes must perform risk management to reduce the probability and impact of vulnerabilities to acceptable levels.

This confidence is achieved by system assurance activities, which include a planned, systematic set of multidisciplinary activities to achieve the acceptable measures of system assurance and manage the risk of exploitable vulnerabilities.

1.3 Purpose

The purpose of this guidebook is to provide guidance in how to build assurance into a system throughout its life cycle. This guidebook identifies and discusses systems engineering activities, processes, tools, and considerations to address system assurance. To be efficient, assurance issues must be addressed as early as possible; system assurance is often much more expensive to add to systems later in the life cycle. Assurance efforts should be commensurate with mission needs and threats (both identified and predictable).

1.4 Scope

This guidebook identifies processes, methods, techniques, activities, and tools for system assurance, using a systems engineering approach, but is not restricting to use of other equivalent and compatible processes, methods, techniques, activities and tools of Member Nations, which are not mentioned and listed here. As defined in ISO 15288, a system is a combination of interacting elements organized to achieve one or more stated purposes (missions). A system may be a system of systems or a family of systems (SoS/FoS).

This guidebook focuses on the electronic hardware (HW), firmware, and software (SW) elements that make up the system, including information storage, electronic sensor elements, information processors, and computer and communication elements. It focuses on the identification of implementable techniques, activities, and tools that can be applied to system assurance at the system and system element level.

This guidebook discusses system assurance by specifically addressing the assurance of security properties throughout the system life cycle. These properties include confidentiality, integrity, availability, authentication, accountability (including non-repudiation), and auditability. It does not address assurance for quality, safety, or dependability. However, an intelligent adversary may be able to subvert a system's functionality, quality, safety, or dependability if there is inadequate assurance of security properties.

This guidebook focuses on assurance of the entire system, not merely of specific system elements. Systems are normally composed from many elements—some commercial and some custom—with many different levels of assurance. Some elements may be "high assurance," meaning that compelling evidence is provided that the element delivers its services in a manner that satisfies certain critical properties (including compelling evidence that there are no software defects that would interfere with those properties). Developing software for high-assurance elements often relies on formal methods, which are rigorous, mathematically based techniques and tools for specifying, designing, and verifying hardware and software systems, as well as extensive testing. Some elements may be "medium assurance," meaning that the element has been designed to meet its critical properties, and that significant effort has been expended to detect and address potential failures to meet critical properties (but not to the level of a high-assurance element). The assurance of an entire system depends on some (or all) of the system elements, but assuring specific elements is insufficient—the system must be considered as a whole. System developers may leverage specific high-assurance elements (so others need less assurance), design the system (e.g., by limiting privileges) so that weaknesses in one element will not harm system assurance, or use compensating processes. A *systems view* is vital for achieving system assurance.

1.5 Document Overview

Following this Introduction, Section 2 presents key concepts of system assurance that will appear throughout the guide. The section provides a foundation of the components that make up an assurance case and its value to the system development life cycle.

Section 3 provides general SA guidance that applies to all systems. Whenever possible, this section follows the structure of the ISO/IEC 15288:2008, Systems and Software Engineering– System Life Cycle Processes. Section 3.1, Agreement Processes, focuses on acquisition and supply. Section 3.2 discusses Organisational Project-Enabling Processes. Section 3.3, Project Processes, includes configuration management and risk management processes. Section 3.4, Technical Processes, discusses the recommended system assurance engineering activities from stakeholder requirements through disposal. Within each process, there is a dedicated subsection called “building the assurance case,” which describes key activities that are required for building a system case.

Annex A, Correspondence with Existing Standards, discusses related standards. The guide also includes a glossary, references, and acronym list. Unless otherwise noted, cross-references refer to sections of this guide.

2 Key System Assurance Concepts

2.1 Assurance Case

The purpose of an assurance case is to provide convincing justification to stakeholders that critical system assurance requirements are met in the system's expected environment(s). Should a set of system assurance claims about the system be expressed, these claims need to be incorporated into the system requirements. An assurance case is the set of claims of critical system assurance properties, arguments that justify the claims (including assumptions and context), and evidence supporting the arguments. The development of the assurance case results in system assurance requirements that are then flowed to the system architecture and the product baseline. The assurance case is generated by the systems engineering technical activities applied to the assurance requirements, and provides evidence of the growing technical maturity of the integration of the system assurance requirements for use within event-driven technical management.

In systems engineering, the activities for developing and maintaining the assurance case enable rational decision making, so that only the actions necessary to provide adequate justification (arguments and evidence) are performed. Assurance case planning identifies and justifies what approach will be taken (e.g., programming language selection, trusted source selection) and what evidence must be collected to justifiably achieve the required assurance. This planning includes cost and technical trade-offs, and integration into the risk mitigation process, the work breakdown structure, and the programme/project schedule. Developing an assurance case clarifies the interaction between functionality, cost, schedule, security, safety, dependability, and other "-ilities" (reliability, maintainability, etc…), so that appropriate trade-offs can be made through risk management.

In systems engineering, assurance case development and maintenance should be executed as part of the stakeholder requirements definition, requirements analysis, architectural design, and risk management processes. The assurance case supports efficient system development while reducing overall risk, by applying various existing expertise/disciplines/processes to address the prioritized assurance requirements.

The assurance case need not be a separate document; it may be distributed among or embedded in existing documents. Even if there is an "assurance case" document, it would typically contain many references to other documents. Regardless of how the assurance case documentation is implemented, there must be a way to identify all the assurance claims, and from those claims trace through to their supporting arguments, and from those arguments to the supporting evidence. For example, an organisation might maintain a list of system requirements, tagging specific ones as assurance claims. Each claim might have a hyperlink to the argument that justifies why the claim will be met (e.g., a risk-mitigation plan) as well as being flowed to the architecture and implementation baselines. That argument might in turn contain hyperlinks to evidence (e.g., results demonstrating that planned actions were successfully executed). The assurance case shows that the system will meet the prioritized requirements, and that it will operate as intended in the operational environment, minimizing the risk of being exploited through weaknesses and vulnerabilities. During initial project planning, the Work Breakdown Structure (WBS) should include an activity to provide this traceability (see Section 3.3.1, Project Planning).

As a minimum:

1. Assurance case claims, arguments, and evidence must be relevant for the system and its operating environment(s)
2. Claims are justified by their arguments
3. Arguments are supported by their evidence
4. The assurance case must be developed iteratively, be sustainable, and be maintained throughout the system life cycle as a living document. This implies that the assurance should be developed in a way that makes it easy to change, that tools should be used to maintain it, and that it should be divided into smaller modules so changes can be localized.
5. The assurance case must be delivered as part of the system, to be maintained during system sustainment (Note: Some systems fail to do this, potentially invalidating the assurance case)

Many industry documents include in the term “assurance” the qualities of safety, security, and dependability. There are existing standards, best practices, and processes that support safety and dependability assurance. This guidebook assumes those processes pre-exist, but they may not be formally labeled or completely integrated into an assurance case. For example, the systems engineering risk management plan and systems safety programme plan typically include assurance information and already share some level of integration, but security is not visible within their assurance considerations. The primary value provided by the security version of the assurance case described here is that these existing risk management and safety plans use claims, arguments, and/or evidence in their assurance case.

Security-related system assurance should be integrated into the systems engineering activities to be cost-effective, timely, and consistent. This approach is emphasized due to the emergence of intelligent, proactive, and persistent adversaries, combined with new and increased risks created through networked weapons systems as well as information technology (IT) systems. The assurance case uses an established best practice to integrate security with other disciplines and their designs, verification and validation. It must be understandable, at a high level, by the relevant stakeholders. Figure 2-1 illustrates the typical assurance case framework.

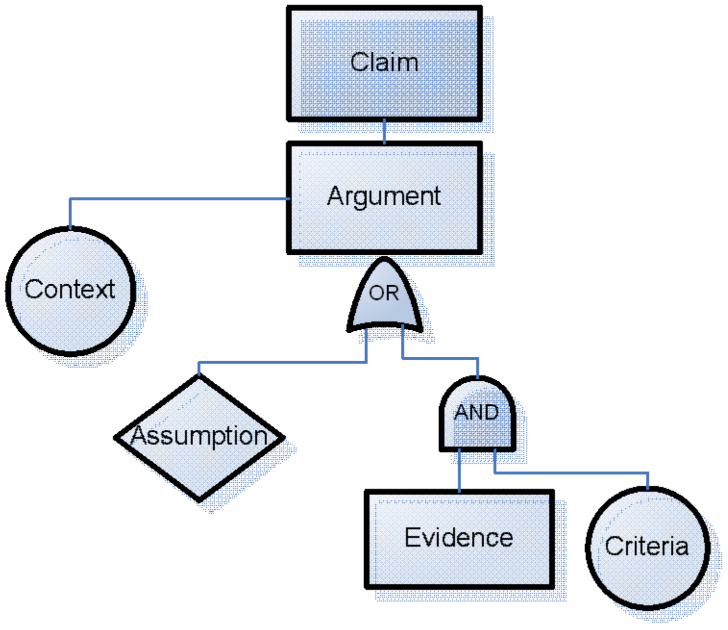


Figure 2-1 Assurance Case Framework

Different disciplines use the same word with different meanings, especially terms such as “assurance,” “integrity,” and “dependability.” (See the glossary for definitions used in this guide.) For example, a system may be required to implement a function controlling hazardous processes. Yet that system may also include anti-tamper (AT) mechanisms that protect this function by destroying the element when it is tampered with. Developing an assurance case can reveal that the AT mechanisms would disable control of the hazardous processes, and aid in ensuring that both requirements are met (e.g., the AT mechanism would need to enable a safe shutdown process).

2.1.1 Claims

Claims identify the system’s critical requirements for assurance, including the maximum level of uncertainty permitted for them. A claim must be a clear statement that can be shown to be true or false–not an action. Claims should be precise and clear to the relevant stakeholders. Claims may be initially identified during the requirements identification process, by contract, or by other means. Claims are often initially identified by stakeholders including users, suppliers, and system integrators.

Claims are usually rendered into sub-claims to help simplify the argument or evidence needed to support the claim (see Figure 2-2). For example, a claim might be that “unauthorized users cannot gain control over the system”; this claim might be subdivided into sub-claims that such control cannot be gained physically, through a network, through subversion of a system element’s supply chain, or by social engineering. Claims might be subdivided by normal and off-normal functionality, or by different portions of the architecture.

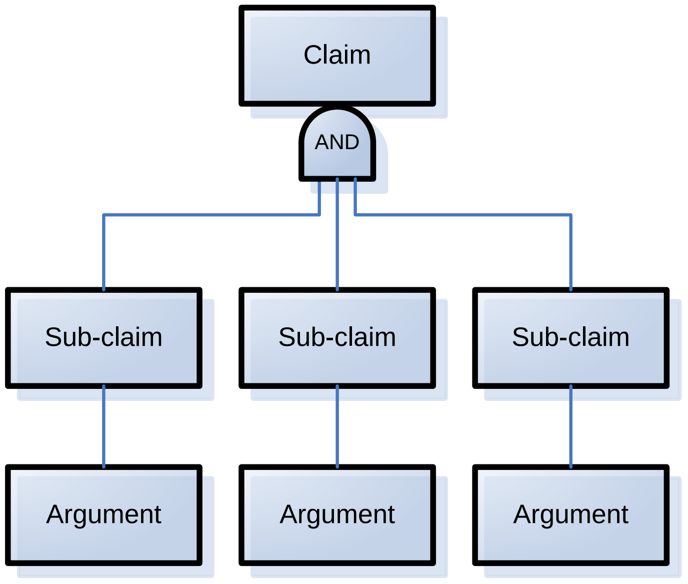


Figure 2-2 Assurance Claim Construct

Claims should identify the required level of confidence. For example, claims should identify if the claim needs to be made high assurance (comprehensive evidence that the claim is met, such as through mathematical proofs and similar means), or merely medium assurance (significant effort is made to reduce the likelihood of failure, e.g., using tools and techniques that significantly reduce the likelihood of weaknesses and vulnerabilities).

2.1.2 Arguments

An argument in an assurance case is a justification that a given claim (or sub-claim) is true or false. The argument includes context, criteria, assumptions, and evidence. Arguments link the SA claim to the evidence. The initial assurance case, except for pre-existing evidence, will generally not have actual evidence but will propose what evidence must be produced to justify the argument that the claim is met. Thus, the initial assurance case allows management to plan and justify system life cycle tasks and postulate what effect(s) might result from not providing certain evidence.

Arguments should be clear, consistent, well-reasoned, and complete (e.g., cover the entire claim or sub-claim). Arguments will typically decompose claims or sub-claims into lower and lower level arguments that eventually connect to assumptions and evidence (see Figure 2-3). The argument will include context (the environment or conditions under which it is effective), which is shown as a separate component in Figure 2-3 to emphasize its importance. The sub-arguments of an argument are themselves arguments. To build an argument, evidence must meet criteria as discussed below. Some assumptions and evidence may be used in more than one argument.

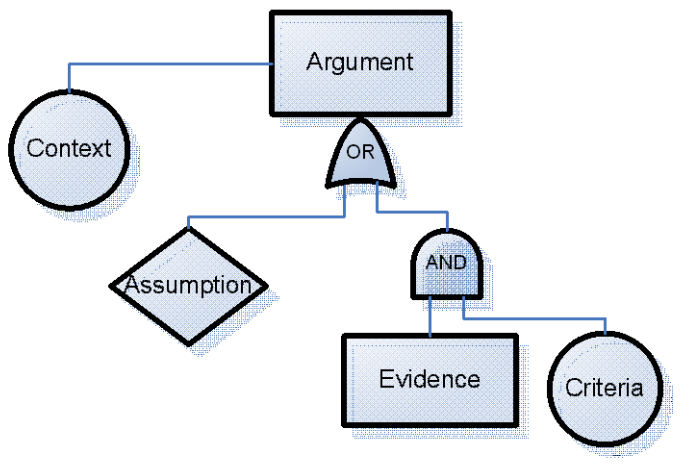


Figure 2-3 Assurance Argument Construct

For rigor, claims, arguments, and evidence should be reviewed at major phase transitions in the system life cycle. Such reviews should ensure that arguments continue to be valid, and that any unnecessary and/or duplicate arguments are removed.

Arguments are typically broken down into further sub-arguments, sub-sub-arguments, and so on, until they lead to a set of assumptions and evidence. Arguments are often, though not exclusively, shown using a graphical notation; examples include Goal Structuring Notation (GSN), Adelard Safety Case Development (ASCAD), Toulmin structures, and fault trees (which are used in fault-tree analysis). Arguments should be reviewed to ensure that they cover all plausible possibilities. Arguments are typically developed using both top-down and bottom-up approaches:

- The top-down approach breaks the claim down into smaller, more manageable arguments to justify the claim
- The bottom-up approach identifies potential assumptions and evidence that may be used to create the leaves (end points) of the argument
- A gap analysis (comparing the top-down and bottom-up results) may be used to identify what additional assumptions or evidence are needed to support the argument (and thus justify the claim)

An argument is only relevant in context, and criteria determine which evidence may be used. There is always the risk from unknowns (including “unknown unknowns”). Defense-in-depth measures, and wider margins for element performance measures, may assist in surviving through these unknowns.

2.1.2.1 Context

Context explains the environment or conditions (including state or mode) under which the argument is effective. For example, the argument’s context may be that the system is operated in an environment where physical attacks are prevented by other means. Context may be viewed as being derived from the aggregate of the assumptions and evidence. Conversely, the assumptions or evidence may be viewed as needing to be sufficient to justify the argument within its context.

2.1.2.2 Criteria

An argument should include criteria for evidence. These criteria will filter the evidence to only evidence that is applicable and adequate for the argument and claim. For example, if an argument requires “thorough testing,” then criteria for “thorough” might be “100% functional testing, software branch testing, and fuzz testing.” Criteria are used to determine whether certain evidence acceptably supports the argument.

2.1.2.3 Assumptions

Assumptions, combined with evidence, support an argument. An assumption should be justified and verifiable. Assumptions would typically be reviewed and confirmed as early as practicable with stakeholders, and need to be monitored throughout the life cycle to ensure that assumptions continue to be valid.

Assumptions should be justified by some documented rationale (e.g., the installation documentation clearly forbids connection to the Internet and that monthly scans and system audit logs for the existence of connections would detect such connections or that a connection was made). A stakeholder or reviewer would need to review and confirm assumptions. Clearly identifying and labeling assumptions speeds the review process. Identifying assumptions prevents a typical integration error (“I thought you were doing that”).

Over time, some assumptions may be validated sufficiently to become arguments supported by evidence. Conversely, key findings disproving an assumption may seriously jeopardize approval of the assurance case, which is why assumptions must be monitored throughout the life cycle..

2.1.2.4 Evidence

Evidence is information that demonstrably justifies the arguments.

To be adequate:

- Evidence must be sufficient for its use in the assurance case arguments, including both its quality and its provenance.
- The stated context and criteria apply to each piece of evidence.
- Where inputs, states, or condition can vary, evidence must cover all possibilities or have a sufficient sample to justify the argument.
- Evidence must be uniquely identified so that arguments can uniquely reference the evidence.
- Evidence must be verifiable and auditable.
- Evidence should be protected and controlled by the configuration management (CM) process.

1) Sample Evidence

- Hardening of development environment (e.g., access controls and identification and authentication (I&A) for CM)
- Development tools that prevent weaknesses/vulnerabilities (e.g., safe languages, safe language subsets, secure libraries, etc.)
- Certification of trusted supplier
- Static analysis tool results for design and code (e.g., avoidance of vulnerability-prone features, unhandled exceptions, valid/invalid data flow including control of tainted data, conformance to coding standards)
- Mathematical proofs and results of proof-checkers
- Assurance testing (e.g., fuzz testing)
- Modeling and simulation results
- QA results
- Red team results
- Certification of compliance against various specifications
- Independent evaluation of a product’s security (e.g., Common Criteria)
- Hardening of system before entering operational environment
- Personnel clearances (of developers, operational administrators, and users)
- Proof of personnel training/education on developing assured systems
- Proof of personnel training/education for operational administrators/users on how to operate securely and respond to incidents
- Verification/test results showing that the target environment’s defenses are adequate to protect the system being developed
- Test results demonstrating the adequacy of countermeasures
- Security regression test results (these may be part of a safety regression test, and should occur as part of the sustainment upgrade process before deploying upgrades)

- Development process designed to improve quality (e.g., higher Capability Maturity Model Integration (CMMI) level, large-scale or in-depth review/evaluation/peer review)
- Field history
- Security incidents

Evidence can be used regardless of whether an element is custom-developed, or is a COTS/government off-the-shelf (GOTS) element, though the specific kinds of evidence available will vary. Evidence is collected throughout the life cycle and verified at each review.

2.1.2.5 Building the Assurance Case

Section 3 describes various life cycle processes, and they often generate both arguments and evidence that are necessary to build the assurance case. Subsections named “Building the Assurance Case” are included to emphasize assurance case activities and evidence developed during that part of the life cycle.

3 General System Assurance Guidance

This section follows the top-level outline of ISO/IEC 15288:2008, Systems and software engineering – system life cycle processes, to ease adding system assurance activities to established systems engineering development practices. ISO/IEC 15288:2008 distributes planning and strategy development throughout its processes; to conform to ISO/IEC 15288:2008, this guidebook does the same.

Occasionally, references may be made to the Institute of Electrical and Electronics Engineers (IEEE) 1220, Standard for Application and Management of the Systems Engineering Process (chapter 6). Organisations may use other life cycle documents such as EIA 632, Processes for Engineering a System, or ISO/IEC 12207:2008, Systems and software engineering – Software life cycle processes. To use this guidebook, organisations should map their life cycle to ISO/IEC 15288:2008; such mappings are often available.

Table 3-1 Guidebook Correlation to Standards

Guidebook (Section 3)	ISO/IEC 15288 (Section 6)	IEEE 1220 (Section 6)
3.1 Agreement Processes	6.1	No match
3.2 Organisational Project-Enabling Processes	6.2	No match
3.3 Project Processes	6.3	No match
3.4 Technical Processes	6.4	6.1- .8

3.1 Agreement Processes

The agreement processes (ISO/IEC 15288:2008, Section 6.1) consists of:

- The acquisition process, used by organisations for acquiring products or services from external sources, and
- The supply process, used by organisations for supplying products or services to external sources

A supply chain is the set of organisations, people, activities, information, and resources for creating and moving a product or service (including its sub-elements) from suppliers through to an organisation’s customers.

An untrustworthy supply chain can lead to loss of assurance. Suppliers (or suppliers of suppliers) may provide counterfeit parts, tamper with their elements intentionally, provide elements with significant weaknesses and vulnerabilities, and/or reveal confidential information. To prevent this, a clear chain of custody must be identified, approved, maintained, and audited.

The sub-tiers within the supply chain may be critical to the assurance of the final system, so evaluation of the supply chain sub-tiers may be required in both the acquisition and supply processes. The acquisition and supply processes, when assurance requirements are added, affect the system lifecycle of those elements and potentially of their supply chain.

Thus, when performing system-level risk management, consider the elements that make up the system, their supply chain, and the supply chain's impact on assurance.

Today's complex acquisition environment has affected the supply process significantly, with a number of trends that have affected our ability to address assurance needs:

1. Suppliers continue to globalize, establishing overseas activities and outsourcing many parts of the system life cycle including product development, requirements and development efforts, test and evaluation (T&E), support service delivery, manufacturing, etc.
2. It has become cost-prohibitive and untimely to develop systems completely in-house in most circumstances. Systems have become extremely large and complex, requiring the use of many elements developed by various specialists. Leveraging of commercial off-the-shelf (COTS) products and government off-the-shelf (GOTS) products has become accepted practice. The use of commercially available elements (including proprietary and open source software products), standards, and associated development and verification tools has also become the norm.
3. Long acquisition times conflict with the shortened release times of some suppliers. This may force repeated re-evaluation of supplier selection, since decisions made early in the life cycle regarding supply may or may not meet the requirements at the time of the supplier product or service usage. This is especially true in cases where integration efforts may take many years.
4. Technology innovation in many cases is driven by new, small, and/or foreign entities that often have not gone through any supplier evaluation process, making programme risk assessment more difficult. Thus, using "best of breed" solutions may introduce significant assurance risks, yet never using "best of breed" solutions may introduce the risk of delivering inferior systems.
   Many of today's products have long operating lives of 20 or more years. During their lifetimes many elements that need to be replaced experience problems with diminishing manufacturing sources. The selection of alternate suppliers or alternate elements invites threats to system assurance.

In support of these trends, the PM and the SE must address the assurance needs of their system while leveraging existing supply management infrastructure. The individuals addressing acquisition and supply processes may not currently have the knowledge to address assurance requirements. They must consider not only the products and services being delivered but also their sub-elements and modifications thereof (some of which may be COTS or GOTS), as well as the tools used to develop and verify the products and services delivered. They should also routinely assess to ensure there is a process for “operational agreement” once the system is live.

In many cases, custom elements (such as application-specific integrated circuits (ASICs) and custom software) may require extra assurance measures. This is because the element developer will typically know or can determine how the element will be used operationally. As a result, they may have the opportunity to divulge confidential information, or to tamper with the element to predict a targeted attack on the system. In addition, weaknesses/vulnerabilities or code tampering are less likely to be noticed and reported by others.

In contrast, an off-the-shelf (OTS) element tends to be developed and tested for generic application. The people in the supply chain of OTS elements will often not know how the element will be integrated into the larger system, and thus are less likely to be able to reveal system-specific confidential information. However, an OTS element may have serious weaknesses/vulnerabilities, either unintentional or intentional, which can compromise the system in its operational environment.

3.1.1 Acquisition Process

The acquisition process is one in which a product or service is obtained in accordance with the acquirer’s requirements. It is used by organisations, including acquisition/procurement offices, systems integrators, and acquirers of many COTS/GOTS elements, for acquiring products or services.

3.1.1.1 Assurance Considerations in Acquisition of COTS & GOTS

Because of cost, time to market, and competitive advantage, systems are increasingly composed of COTS, GOTS, and integration elements. These elements tend to be developed and tested for a generic set of applications as well as large market segments (e.g. banking, federal government, healthcare, etc). The members of the supply chain of OTS elements usually will not know all the possible use cases in which these elements are integrated.

Thus, system development typically involves evaluating OTS alternatives. For assurance, appropriate resources need to be allocated for evaluating the assurance characteristics of the OTS elements. For those OTS elements that are ubiquitous, public information is often available to help determine the likelihood of serious weaknesses/vulnerabilities, and various mitigation options also tend to be available. This information must be considered on a case-by-case basis, to determine which options (custom or OTS) are best for the system being developed. In general, acquisitions should prefer acquiring commercial OTS (COTS) elements instead of building new government OTS (GOTS) elements. In some cases, however, risk analysis may determine that a GOTS approach is best.

Because of the nature of OTS products, the acquirer typically has no influence on current and limited influence on future OTS products. Acquirers should seek OTS solutions that use certified (or best practices) implementations of standards-based interface and should implement a defense-in-depth approach:

- *Standards-based interfaces.* To enable the use of commodity elements, standard interfaces with competing implementations should be evaluated. This evaluation enables switching suppliers, if needed (e.g., because of product weaknesses). In addition, the review process of open standards can remove weaknesses that would otherwise remain in an interface.
- *Defense-in-depth approaches.* Defense-in-depth approaches are the layering on and overlapping of security as well as system assurance measures. In many systems, its resistance to attack is no greater than its weakest link. Using a defense-in-depth strategy, if one defensive measure fails there are other defensive measures in place that should continue to provide protection. Defense-in-depth can be applied inside an OTS element, as well as when combining OTS elements. OTS elements that implement defense-in-depth strategies will tend to be better at countering unintentional weaknesses. Where practical, use two or more levels of defense against primary threats (e.g., suppliers that are least trusted).

In addition, the SE and the PM may need to consider the following factors before choosing a particular OTS product:

- Does the supplier have product evaluation criteria defined for the reduction of potential weaknesses in COTS?
- If a product defect/bug list can be made available, consider examining it. If independent defect/bug lists can be made available, consider examining them too. Do they show that the supplier actively works to prevent weaknesses from being released in their product, and that they place a high priority on repairing important vulnerabilities if found later?
- What are the maturities of the product and the underlying product technologies? An immature product, or a product based on poorly understood technology, is more likely to have vulnerabilities.
- What is the track record/history of the product or organisation regarding vulnerabilities? How often do they occur in released products? What were the impacts and how long did they take to be resolved?
- Is it prone to vulnerabilities compared to its competitors?
- Are there publicly known and uncorrected weaknesses or vulnerabilities?
- What is the organisation’s track record regarding correction of vulnerabilities (including speed and thoroughness)?
- Does the product have certifications that justify its assurance? If it has received widespread reviews (including peer reviews or testing), what are their results?

- What mechanisms exist or can be added to prevent the use of counterfeit parts/software?
- What is the supplier's strategy to reduce vulnerabilities?

3.1.1.2 Assurance Issues in Managing System Integrators

System assurance is not just about assuring the final product; assurance must be addressed throughout the entire acquisition cycle. The system integrator plays a key role in assurance. The acquirer must keep this in mind while managing the integrator.

The acquirer and the system integrator must collaborate to increase assurance throughout the supply chain.

Consider using the following approaches:

- *Limit the information available to the supply chain.*
  - In many cases, the risks from commodity elements can be reduced by preventing the supplier of the product or service from knowing the ultimate user or the use. This approach can reduce the ability of the supply chain to induce a targeted attack. This may be difficult for services or future maintenance of a product (particularly where the Internet or physical address of a customer may give away this information).
  - Limiting systems design and operational information reduces the risk of revealing confidential information and also reduces the opportunities for intentional subversion from the supply chain. One method for doing this is to design the system so that the elements with confidential information are separate and can be added after the other elements have been received from the supply chain. For example, a generic SmartCard implementing JavaTM could be purchased from a supply chain, and specific algorithms/applets could then be implemented in-house by trusted personnel (limiting exposure of confidential information). Another example might be to intentionally use FPGAs or microcode to implement certain functions, so that the suppliers provide generic functions and cannot reveal confidential algorithms.
- *Standards-based interfaces*. To enable the use of commodity elements, the use of standard interfaces with competing certified and or best practices implementations should be encouraged. Open standards, in particular, should be preferred, enabling easier switching of suppliers, if needed. (See also Section 3.1.1.1, “Assurance Considerations in Acquisition of COTS & GOTS”)
- *Defense-in-depth approaches.* Defense-in-depth approaches are the layering on and overlapping of security as well as system assurance measures. It is assumed that the strength of any system is no greater than its weakest link. Using a defense-in-depth strategy, should one defensive measure fail there are other defensive measures in place that continue to

provide protection. Use two or more levels of defense against primary threats (e.g., suppliers that are least trusted).

2) Identify Suppliers

PMs and SEs should aid the acquisition community in the identification and selection of their suppliers. This becomes even more critical when addressing new technology innovations, products, and service methodologies.

Potential assurance issues include not only unintentional vulnerabilities, but intentional vulnerabilities. Because intentional vulnerabilities can be introduced at various points in the supply chain, protecting the supply chain is crucial.

In some cases, the PM and the SE generally lead the identification of potential suppliers, based on system requirements and market research. At least some of these potential suppliers are typically previously unqualified suppliers (including those who are small or new to the market).

3) Select a Supplier

Acquirers (including all acquirers in the supply chain) must consider assurance issues when selecting their suppliers. This will often require a better understanding of their supply chain, internal evaluation of suppliers, and/or external supplier evaluation programmes (including those for particular market sectors). Selecting a supplier includes consideration of the people, processes, products/services, and the tools. System assurance requirements for services (e.g., systems integration) are addressed in each of the elements discussed below in support of selecting a supplier.

Suppliers must be considered together with an assessment of their system assurance risks. It is impractical to evaluate in depth all potential risks for all suppliers, so the risks must be prioritized. Generally, careful assessment should be applied to suppliers of key elements, especially custom elements, and/or suppliers of elements whose use is so widespread that it is difficult to ensure that their failure would not cause enterprise-wide failure.

PMs and SEs must consider that the supply chain includes not only OTS products, but also the integrators that use such products to develop systems and systems of systems. For supply chain assurance considerations, the people, processes, products, and tools must be addressed to provide a comprehensive evaluation.

4) People

Although there is limited control over the various organisations and their personnel in the supply chain, consider assessing their reputation and qualifications, as relevant.

- *Reputation.* Does the organisation have a reputation for providing authentic elements (as opposed to counterfeit parts)? Does it have a reputation for producing/providing quality elements? Does it know SA or functional requirements of the system or acquisition processes? Is it a qualified supplier (e.g., an authorized distributor, etc.)?

- *Qualifications.* Do the personnel/organisations used in the development of the products have the knowledge (possibly reflected through training or certifications) of security engineering, assurance methodologies, and the integration of them across the full life cycle, in association with their tasks? (Without this knowledge, they may inadvertently create weaknesses in the system.)

5) Processes

Current processes do not always adequately address system assurance. However, existing life cycle processes such as evaluation and certification processes provide a framework for system assurance, and for the collection of evidence for system assurance.

Consider process questions that may enhance assurance as noted below:

- What system life cycle processes are implemented and what certifications achieved by the supplier that relate to assurance or security engineering practices (e.g., tool-based evaluation of source or binary code, peer review, verification of defensive functions, red teams, etc.)? (See Sections 3.3 and 3.4, Project Processes and Technical Processes, for more detail.)
- How rigorously are they practiced?
- Are there process controls in place to prevent or limit subversion (e.g., configuration management)?
- Will these processes produce predictably assured outcomes?

Examples of process evaluation approaches, or specifications that imply process requirements that can be leveraged for assurance, include ISO 9000, ISO/IEC 27001, Six Sigma programmes, CMMI, SSE-CMM, etc. Supplier-performed processes are typically ineffective for countering the risk of a malicious organisation, but they may help counter malicious individuals or subcontractors, and unintentional vulnerabilities.

6) Products/Services

Assurance requirements for acquisition of products are discussed above (3.1.1.1). When acquiring a service, consider reputation, qualification, and proven processes as discussed above.

7) Tools

Tools have become so critical to implementing the system life cycle that they can intentionally or unintentionally destroy the assurance of a system, and do so in a way that is unobserved by the system's developers. Today many tools are used in the development of hardware and software, and these tools can access the system under development in ways that could result in intentional or unintentional vulnerabilities in a system.

Development tools may introduce vulnerabilities, such as Trojan horses, or buffer overflows into the system. Such tools include source code generators (including model-driven architecture approaches), software compilers, assemblers, hardware compilers (e.g., VHDL), and tools to correct integrated circuit masks. A tool should have passed qualification requirements for use if it eliminates, reduces, or automates a process step without its output being verified. Where possible, opt for tools that are deterministic (i.e., they deliver the same output for the same input in the same environment), since these are much easier to qualify and test. Spot-checking of results, or even careful examination of generated results, may be appropriate in some circumstances.

Some verification tools may not be able to introduce vulnerabilities, and exacerbate the failure to detect errors or weaknesses that they are expected to detect. These include such tools as static analysis tools that automate a system verification process activity – source and binary code analysis focused on development of both intrinsic and defensive function implementations. Other verification tools include type checkers and functional test tools that focus on evaluating elements. Such tools should be used in an optimized manner so as they are able to detect as many vulnerabilities as possible in the system (e.g., not allowing them to modify the “real” system elements). Tools and/or tool vendors may undergo qualification, and/or go through supplier evaluation processes. The limits of such tools must be understood by their users. Innovation is ongoing in this area, so tool users will need to keep up with changes in the industry.

From acquisition through usage of tools, it is important to ensure that the tools are not tampered with or are not introducing errors or weaknesses into the system that can reduce system assurance levels.

8) Building the Assurance Case

Evaluate element suppliers, and select elements based on their ability to support the assurance case, to justify the assurance claims against the system threats. Off-the-shelf element suppliers typically cannot add new information; but they can often provide some evidence to increase confidence in their element. System threats and assurance claims should be used as the basis for deciding on the elements to be supplied and the associated evidence needed for the assurance case. When evaluating element suppliers, consider the risks of both intentional and unintentional vulnerabilities being included in that element, and if there are mitigation approaches available.

When building the assurance case, possible evidence (where * indicates evidence that may help against intentional vulnerabilities) includes: supplier reputation*, track record (defect and vulnerability reports, including uncorrected vulnerabilities), past speed of correction, standard interfaces with open documentation of extensions, transparency of implementation*, CM processes that protect the software (e.g., multi-person review)*, product assessment, and compensating mechanisms. Product assessment can include tool-based assessment (using static and/or dynamic tools), peer reviews, and certifications (such as Common Criteria); such assessments may be done by the acquirer, the supplier, or an independent third party. Compensating mechanisms may include limiting the amount of information available to the supply chain*, defense-in-depth, development processes (e.g., ISO 9000, CMMI, SSE-CMM), patch management processes, and individual qualifications.

Any of this information provided by the supplier should be provided in writing with the appropriate approval. Once a supplier has been selected, the rationale and supporting evidence should be documented so it can be traceable, and so that if issues arise or changes are required, they can be quickly addressed.

3.1.2 Supply Process

The ISO/IEC 15288:2008 supply process focuses on providing an acquirer with a product or service in accordance with the acquirer's requirements. The supply process is used by organisations such as systems integrators and product suppliers. For information on how to work within the framework of assurance in the systems life cycle process, review Sections 3.1, 3.3, and 3.4 (Acquisition, Project, and Technical Processes) for specific guidance. When acquiring sub-elements within a system from other organisations, see Section 3.1.1, Acquisition Process. System delivery must ensure that the intended product is received; see Section 3.4.7, Transition, for more detail.

3.2 Organisational Project-Enabling Processes

Organisational Project-Enabling Processes (ISO/IEC 15288:2008, Section 6.2) support the organisation's capability to acquire and supply system products or services through the initiation, support, and control of projects. They provide the resources and infrastructure necessary to support projects (system and SoS life cycles) and ensure the satisfaction of organisational objectives and established agreements.

The organisation has policy, governance, and technical constraints that affect the systems being developed. This document focuses on the system life cycle, and not the organisational project-enabling processes. System life cycle processes are connected to organisational processes. For example, the requirements analysis and architectural design processes should be enabled by and should capture the requirements and constraints of the environment in which the system will reside, including any policy, governance (including Human Resources), and technical constraints.

3.3 Project Processes

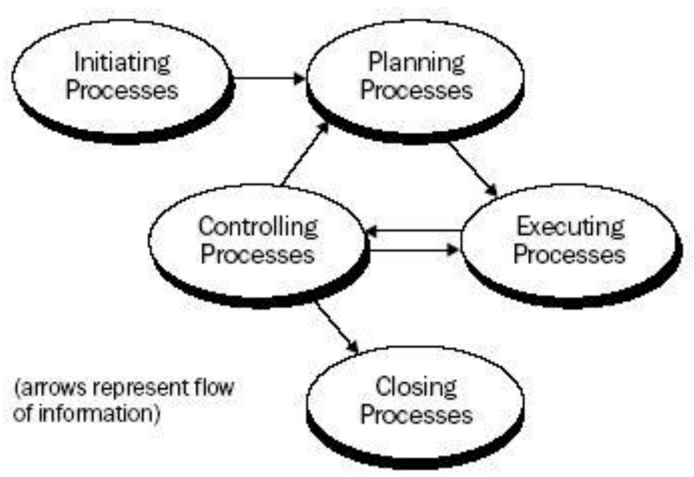
Project Management (PM) with support from Systems Engineering (SE) is critical to the development or acquisition of an assured system. To deliver system assurance, the PM must consider the mission, threats against the mission, system risks, and their relationship with system assurance. The following subsections augment existing PM practices with assurance guidance to manage system risk, allowing the PM to balance the necessary rigor to achieve assurance versus project cost and schedule.

As with the rest of the guidebook, this section mirrors the ISO/IEC 15288:2008 organisation, referencing additional PM and SE processes (ISO/IEC 15288:2008, Section 6.3). Table 3-2 compares SA Guidebook sections with the ISO/IEC 15288:2008, the Project Management Book of Knowledge (PMBOK 2004), and International Council on Systems Engineering (INCOSE) processes.

Table 3-2 Guidebook Project Processes Mapped to ISO/IEC 15288, PMBOK, and INCOSE

Guidebook Subsections (of Section 3.3)	**ISO/IEC 15288 Project Processes (Section 6.3)**	**PMBOK Processes**	**INCOSE SE Processes**
3.3.1 Project Planning	6.3.1	Initializing & Planning	Technical Management
3.3.2 Project Assessment	6.3.2	Controlling	Technical Management
3.3.3 Project Control	6.3.2	Controlling	Technical Management
3.3.4 Decision Management	6.3.3	Controlling	Technical Management
3.3.5 Risk Management	6.3.4	Planning	Technical Management
3.3.6 Configuration Management	6.3.5	Executing	Technical Management
3.3.7 Information Management	6.3.6	Executing	Technical Management

Figure 3-2 illustrates PM processes according to L. Rowland:



Source: L. Rowland, Hawaii Pacific University

Figure 3-1 Programme Management Processes

3.3.1 Project Planning

Project planning (ISO/IEC 15288:2008, Section 6.3.1) is the process used to produce and communicate effective and workable project plans. The project tasks, deliverables, resources, and schedules required to implement system assurance must be incorporated into these plans. Note that the project planning processes of OTS elements are addressed by the OTS suppliers.

3.3.1.1 Define project objectives, constraints, and scope

Ensure system objectives and constraints include assurance as it relates to project performance, quality, cost, and stakeholder needs.

3.3.1.2 Establish a work breakdown structure

The WBS should identify the key operational and/or mission-critical subsystems and elements. These are subsystems or elements that need to undergo greater scrutiny during system development and integration to ensure mission completion while maintaining the confidentiality, integrity, availability, authentication, accountability (including non-repudiation), and auditability of the system. Regardless of how the assurance case documentation is implemented, there must be a way to identify all the assurance claims, and from those claims trace through to their supporting arguments, and from those arguments to the supporting evidence.

The WBS should include:

- Line items to ensure that system assurance is part of all project processes, for example:

- Programme and system threat assessment, including threat scenarios
- Assurance case development and maintenance (see Section 2.2, Assurance Case)
- Peer reviews of design and implementation focused on assurance issues
- Static analysis during implementation
  - Source code analysis during implementation and integration
  - Binary analysis for elements for which source is unavailable
- Implementing countermeasures (e.g., anti-tamper)
- Penetration test/red team activities

- Tasks required to assure non-developmental items (NDI). To achieve the required assurance level for a system, some NDI may require additional assurance efforts.
- Risk management activities related to assurance to address weaknesses, vulnerabilities, and evolving threats, and their impact to the system.

(For software issues in the WBS, see also Fedchak, McGibbon, and Vienneau 2007).

3.3.1.3 Define and maintain a project schedule

Assurance case development and maintenance must be part of the project schedule.

3.3.1.4 Define project achievement criteria throughout life cycle

Ensure that major decision gates require passing measurable assurance criteria. Often these will be existing criteria or measures. Examples of such criteria include:

- Vetted list of potential threats and mitigations
- Design and implementation peer review report highlighting significant assurance-related concerns with associated mitigations
- Results of static analysis, both automated and manual, highlighting resolution of source code and binary analyses issues
- Analysis and resolution of certification and accreditation test results
- Analysis and resolution of Common Criteria evaluation results
- System test results that include assurance-related tests (such as testing system input validation to demonstrate rejection of sample malicious data) and resolutions
- Analysis of System Risk Assessment and hardening results
- Analysis of red team results (e.g. system penetration testing) highlighting resolution/actions to address high/medium risks

3.3.1.5 Define project costs and plan a budget

Ensure that adequate funds are allocated to assurance activities. Such funds should be based on the project schedule, labor estimates, infrastructure costs, and so on. The risk management process considers the assurance risks of a system; the tasks to mitigate those risks must be adequately funded in the budget for the mitigation activities to be effective.

In some cases, some elements may be so critical that high assurance measures must be taken (e.g., formal methods). If high assurance measures must be used, budget accordingly; these measures may cost more in time and money. Where a red team is used, funding should be allocated for addressing problems found by the red team, and not just to perform the red team analysis itself. Such project planning and budgeting tools as COSECMO (Cost Security Model) and COCOMO (Cost Control Model) may be used in support of this effort (Colbert 2006).

3.3.1.6 Establish structure of authorities and responsibilities

Ensure that those tasks requiring assurance expertise are identified and that they will be performed by qualified individuals. Assurance impacts all aspects of systems life cycle as well as many key roles such as architects, developers, programme managers, systems engineers, contracts personnel, system administrators, and others. Identifying the core set of roles that can influence, resource, and shape the assurance of any system/element results in more affectively achieving its targeted system assurance.

3.3.1.7 Define infrastructure and services

Identify the infrastructure and services necessary for system assurance. Ensure that the facilities and communications networks will provide the necessary confidentiality, integrity, availability, authentication, accountability (including non-repudiation), and auditability, and are adequately protected from attack. This includes supporting all necessary access controls. Ensure that IT assets (including CM tools and processes) are adequate for the system life cycle before they are used, e.g., CM tools should be able to identify exactly who made a change, what the change was, and when it occurred (see Section 3.3.6, Configuration Management). Identify changes in policy necessary to address assurance.

3.3.1.8 Plan acquisition of materials

Develop a detailed plan to manage the risks from vulnerabilities and weaknesses that may impact assurance goals for the system. Existing supplier assurance, supplier selection, and/or assured acquisition processes, should be used as part of the decision process regarding COTS/GOTS elements, particularly for a system of unknown pedigree (SOUP). (See also Section 3.3.1.5, “Define project costs and plan a budget,” and Section 3.1, Agreement Process, for more information.)

3.3.1.9 Generate and communicate a plan for technical management

While communicating the plan, ensure that appropriate safeguards are put in place by the technical management team to support assurance requirements.

3.3.1.10 Define project measures

See the assurance case process section for more information on generating, collecting, and using relevant assurance information. For system requirements, see the requirements analysis process.

3.3.2 Project Assessment

The project assessment (ISO/IEC 15288:2008, Project assessment and control, Section 6.3.2) process determines the status of the project. A plan may need to be modified, based on the results of an assessment.

For system assurance:

- Determine the variance between planned and actual results, including the variance's impact on system assurance
- Focus on assessment of elements where there are indicators of potential assurance issues. For example:
  - critical elements with unusually high change rates or fault densities
  - significant changes in supply chain, including change in supplier
  - systems and elements of high complexity
- The assurance case should be revisited when considering a new technology, process, or supplier, to ensure that the technology or process does not enable a new threat or undermine the safeguards already in place to protect the system

3.3.3 Project Control

Project control (ISO/IEC 15288:2008, Project assessment and control section 6.3.2) refers to the direction of the project plan execution and ensures that the project performs according to plans and schedules, within projected budgets, while satisfying technical objectives.

Ensure that corrective and preventive actions, which on the surface might not appear to be related to assurance, do not adversely affect system assurance. In some cases an action that appears to be beneficial can, as a side effect, cause serious impact to the assurance of the system. Examine the assumptions in the assurance case to counter this.

Ensure that corrective actions important to assurance are actually addressed, resolved, and deployed. Customers often complain when a system fails to implement critical functionality, but often do not complain about a vulnerability until it is being actively exploited (which may be too late). When an issue arises in an acquired good or service, including a functional defect, vulnerability, or weakness, constructively interact with the suppliers to correct it.

3.3.4 Decision Management

Decision Management (ISO/IEC 15288:2008, Section 6.3.3) refers to the practice of selecting the most beneficial course of project action. See guidebook sections 3.3.5, Risk Management and 3.4.3, Architectural Design. These processes include considering available options, conducting effectiveness assessments, and making technical trade-offs during the decision-making process.

System assurance needs to be actively addressed as part of the risk management process. Any tradeoff decisions must consider the threat posed by the active adversary. System assurance needs to be a documented requirement, included in any decision-making process.

For each decision situation, the impact of the proposed alternative action and its associated risks must be evaluated, using the assurance case to ensure that the overall system assurance has not been compromised. Once a decision relevant to assurance has been made, its impact must be reflected in the assurance case.

3.3.5 Risk Management

Risk management (ISO/IEC 15288:2008, Section 6.3.4) attempts to reduce the effects of uncertain events that may result in changes to quality, cost, schedule, or technical characteristics. System assurance issues are often not considered during the risk management processes, activities, and work products, resulting in project stakeholders unknowingly accepting assurance risks that can have severe financial, legal, and national security implications. Risk management covers the entire life cycle, including the operation and maintenance processes. A PM, with assistance from an SE, must realize that a system assurance threat is a threat to programme performance. Through risk management, risks (including system assurance risks) are identified and categorized, their probabilities and consequences are determined, a risk mitigation strategy is specified for each risk, the risk status is communicated, and unacceptable risks are acted upon.

Upon determining the organisation’s implementation of risk methods and policy, the existing PM and SE processes for risk management need to be reviewed and expanded to address system assurance risks throughout the system life cycle. This section addresses those risks. Generic risk management is not addressed here.

In the execution of the programme, the PM and SE will need to follow the plan for protecting the system against threats to the assurance goals. As part of this process, the PM and SE will have to reevaluate and update, as needed, the assessment threats, the weakness and vulnerabilities, and any other changes that affect the overall risk mitigation plan (such as costs).

3.3.5.1 Establish risk management strategy

A risk management strategy is a systemic approach to risk identification, assessment, and mitigation. There must be a continuous and iterative system assurance process that delivers due diligence for identifying and assessing risk issues, as well as mitigating issues that are selected as requiring attention. The assurance process, including risk management, is integrated throughout the life cycle, including during operations and maintenance, due to emerging threats and the dynamic nature of today's operational environments. There is a risk of subversion throughout any part of the system life cycle processes, including risk management itself. Consider using the assurance case's claims, arguments, and evidence as a framework for organizing and addressing system assurance risks.

PMs must assume the existence of an intelligent and malicious adversary. Thus, probabilities must be measured differently than for naturally occurring events, as they may not be accurately computable, and they may not even be applicable. Normally, probabilities can be used to accurately represent the likelihood of a natural event occurring. However, an intelligent adversary can react to countermeasures, and choose the time and place to make an attack. Depending on the system, a security breach can have a catastrophic effect on safety. Moreover, highly motivated and well-resourced adversaries must be assumed.

At a minimum, security engineering must provide safety engineering with information about potential security breaches and their probabilities (if computable). Even better would be to develop the security and safety cases jointly to ensure that conflicts between safety and security are resolved early in the development cycle, and that it is possible to take advantage of commonality between safety and security for risk management.

There is a direct relationship between the requirements analysis process and risk management. The requirements analysis process identifies system assurance requirements (among other requirements). The risks of failing to identify or implement such requirements are covered here, in risk management. Note also that the risk management processes impact all other processes.

3.3.5.2 Identify and define the risks

The risk management process must consider the risks on system assurance from the supply/acquisition chain, the system life cycle personnel (including the developers, integrators, operators, and maintainers), and other threat agents. Consider the risks of insertion of both unintentional and intentional vulnerabilities, from both trusted and untrusted sources, throughout the life cycle (including system development, operations, maintenance, and disposal).

Identification of risks should be an open and encouraged process. While the process may lead to an increase in identified risks, the processes to assess legitimacy and prioritize the risks should be just as robust, thereby leading to a manageably-sized database for further mitigation activities. To identify and define risks, programmes should consider using repositories of threats, weaknesses, and vulnerabilities, including such sources as:

- Common Vulnerabilities and Exposures (CVE®): http://cve.mitre.org/ This site provides a list of the standard names for security vulnerabilities and exposures.
- Common Weakness Enumeration (CWE): http://cwe.mitre.org/ This site provides a dictionary of common software weaknesses. CVEs are instances of CWEs.
- Common Attack Pattern Enumeration and Classification (CAPEC): http://capec.mitre.org/
  This site provides a dictionary of patterns that malefactors could use to attack computer systems.

The PM/SE should also consider the specific system requirements including operational environment that may affect the use in the system assurance case.

Identification and definition of risk should include circumstances or events that can potentially harm the system through destruction, disclosure, or modification of data and/or denial of service. Such results can be caused by unintentional or intentional vulnerabilities introduced by authorized system creators, as well as unauthorized individuals and organisations gaining control over the development environment. For example:

1. A threat agent, masquerading as an authorized user, gains access or greater privilege to a system through stolen logon IDs and passwords, impacting system confidentiality:
2. The threat agent may insert malicious code to impact the system operations in many ways.
3. The threat agent may alter data, in transit, impacting data integrity. These data can be reordered, deleted, or modified through the use of such simple techniques as sniffing and replay.
4. A threat agent prevents a system from functioning in accordance with its intended purpose, creating a denial of service attack, impacting system availability. This attack may be achieved through the subversion of a piece of equipment, rendering it inoperable and/or forcing it to operate in a degraded state using a time-delay attack.

As the risks are identified, they should be collected into a risk registry (a list of risks and information about them) for analysis. Developing the risk registry should begin early in the system development process and be a continuing effort.

In general, one risk to be identified and addressed is whether there is a potential for not being able to achieve the necessary system assurance in a timely manner, potentially resulting in a need to risk mitigate the system certification or accreditation, and/or resulting in the system not being used as intended. The use of this guidebook is intended to help reduce this risk, and should be included as part of the risk mitigation plan.

3.3.5.3 Determine the probability and possible consequences

The probability and possible consequences associated with each risk occurrence should be determined. Note, however, that quantitative probabilities are difficult to pre-determine for system assurance. One reason is that intelligent adversaries tend to attack those areas of the system or its development processes that are least defended. If only areas with a high probability of receiving attack are hardened against attack, the adversary is more likely to attack the unhardened areas—which had previously been considered as having lower probability of receiving an attack. In addition, methods for obtaining measures of system assurance are still in their infancy, though there is ongoing work to improve this. Thus, system assurance probabilities will often need to be determined qualitatively (e.g., high/medium/low). In addition, these probabilities will need to be revisited on a situational basis, to consider environment changes or if an intelligent adversary might exploit them as “easier” attack paths. Thus, risk management results need to be less sensitive to the accuracies of the probabilities, typically using a layered set of approaches as discussed below.

Where judging risks is difficult, one possible approach is to make them at least not unacceptable (tolerable) and “as low as reasonably practicable” (ALARP). In employing the ALARP approach, judgments about what to do are based on the cost-benefit of techniques – not on total budget. However, additional techniques or efforts are not employed once risks have achieved acceptable (not just tolerable) levels. This approach is attractive from an engineering viewpoint, but the amount of benefit cannot always be adequately established.

When considering the possibility of supply chain subversion, both intentional and unintentional, consider the risks associated with that supplier (prime, subcontractor, and so on), its infrastructure (e.g., network, configuration or product data management system), and its personnel. These subversions may result from internal or external threats on that supplier. Supply chains quickly become complex, since suppliers have suppliers, which then have suppliers, and so on. See Section 3.1.1, Acquisition, for more information.

Risk management must consider the broader consequences of a failure (e.g., to the mission, people’s lives, property, business services, or reputation), not just the failure of the system to operate as intended. In many circumstances, consequences can vary probabilistically. However, in system assurance, an intelligent adversary can choose its attack to occur at the worst possible time, with the worst possible consequences. An intelligent adversary can even choose to cause multiple simultaneous failures, some of which may be in the environment. Thus, for system assurance, the entire chain of following events must be identified and considered to determine the worst possible consequences. For example, loss of a critical service such as traffic control system failure of a major metropolitan area could result in major accidents, not just a traffic jam. A critical infrastructure control system that shuts down electrical generation could cause not just regional blackouts but deaths due to loss of heat/cooling, and massive property damage due to flood pump failure. Thus, the system assurance case should be sufficiently broad so as to identify not just system (including SoS) failures, but the larger consequences of those system failures such as mission failure, loss of life, physical harm, and major loss of property.

One possible consequence is the reduction in the assurance of the system itself. For example, the system assurance may have been predicated on certain assumptions later found to be false (e.g., due to a breach). The system assurance case arguments should aid in determining the impact and methods for remediation, should this occur.

3.3.5.4 Evaluate and prioritize the risks

The risks should be evaluated and prioritized in terms of probability and consequence. As noted above, prioritization needs to be less sensitive to the accuracies of the probabilities, and consequences must consider the chain of effects that an intelligent adversary could cause.

3.3.5.5 Determine the risk mitigation strategies

For each risk, determine the risk mitigation strategy, including risk avoidance, transfer, mitigation, acceptance (retention), and/or some combination. Consider trade-offs across the enterprise and mission, as well as trade-offs between system risks, balancing between them and the resources available.

In many cases, a single defense mechanism will not be sufficient to mitigate risk. Instead, layered defenses (e.g., defense-in-depth or engineering-in-depth) may be necessary to provide acceptable risk mitigation. Some risks can be avoided or eliminated, for example by changing the software's design, elements, or configuration, or the configuration of its environment. Some approaches may involve a trade-off between performance and security. These trade-offs may be easier to perform by leveraging element(s) threat models, the element's associated risks and its role in the overall system mitigation strategy. (See also Sections 3.4.3, Architectural Design and 3.4.4, Implementation.)

3.3.5.6 Define the threshold of acceptability for each risk, and actions if exceeded

Different stakeholders may have different thresholds of acceptable risk. In some cases, degradation of service to a certain amount may be acceptable.

3.3.5.7 Communicate and maintain the risk registry, risk mitigation actions, and their status

The risk registry information, mitigation actions, and their status may need to be protected from disclosure, since this information could be a significant aid to an attacker. Risk registry is also key to the building of the assurance case as it can contain critical information such as claims, arguments as well as the evidentiary data and other relevant information in support of the assurance case.

The amount of information required for a given risk may increase depending on the criticality level of the system, the importance of that risk, and the stringency of the system's certification and accreditation process. Risk management data, generated as a normal part of a product/service development, may be provided to C&A officials as augmenting evidence of IA control implementation, but is not explicitly required for C&A approval. The verification process may provide some of the objective evidence to justify that a given risk has been managed.

3.3.5.8 Methods for assessing security risk

Several system and software security risk assessment methods exist, some with supporting toolsets; but other equivalent and compatible system and security risk assessment methods, some with supporting Toolsets of Member Nations, can also be used:

- Microsoft's threat modeling as described in Swiderski and Snyder 2004
- Microsoft's ACE (Application Consulting and Engineering) Threat Analysis and Modeling (Avery and Holmes 2006)
- European Union-funded Consultative Objective Risk Analysis System (CORAS) and Research Council of Norway-funded Model-Driven Development and Analysis of Secure Information Systems (SECURIS) (Hogganvik and Stølen 2006)
- PTA Technologies' Calculative Threat Modeling Methodology (CTMM)
- Trike
- CMU SEI's OCTAVE (OCTAVE 2001)
- Siemens' and Insight Consulting's Central Computer and Telecommunications Agency (CCTA) Risk Analysis and Management Method (CRAMM)

While not primarily intended for software, the *Security Considerations for Voice Over IP Systems, Information Security Handbook: A Guide for Managers, and Risk Management Guide for Information Technology Systems* may also be useful for performing system security risk assessments.

3.3.6 Configuration Management

Configuration management (CM) (ISO/IEC 15288:2008, Section 6.3.5), traditionally a best practice for engineering a high-quality system, plays a key role in establishing one of the most important elements in protecting against exploitation. The key is to employ strict control and detailed auditing of activities related to the ongoing processes. Leveraging such CM resources as tracking databases and spot inventory checks as well as a labeling system(s) that provide specific target tracking, aging, and other sufficient pertinent data to categorize and track all assets should be used. An adversary will find it much more difficult to exploit a system that maintains a strong monitoring and report structure.

Configuration management is a process to establish and maintain the integrity of all identified outputs of a project or process, and make them available to concerned parties. This includes the CM of the information captured for the project as well as the CM used by the supplier for the product as it was being developed. In addition, to support system assurance, the CM process must establish and maintain confidentiality, integrity, availability, authentication, accountability (including non-repudiation), and auditability . It must be rigorous commensurate with the criticality of the system, data, and mission, and be flexible enough to enable addressing a wide variety of threats. CM should be tailored with respect to system assurance, which counters maliciousness, reduces uncertainty, and provides objective evidence. Granularity within the CM system should be adjusted to support the assurance case. In addition, the CM system must use techniques, such as non-repudiation, to deter changes that are intended to subvert the system.

For purposes of this guidebook, the “CM system” is the set of processes, procedures, tools, and so on that implements CM. (CM will not be discussed in general in this document; only system assurance aspects that affect or are affected by CM will be.) A CM system must protect its information, such as the current and any previous configurations, as well as an audit trail of the changes made, who made them, and when they occurred. CM systems must normally keep this audit trail information immutable, to support accountability.

The CM process is interlinked with the information management (IM) process. Here, CM is considered as the framework in which the information is contained. Individual information elements are contained in the framework; collections of information are managed through CM. There may be cases in which information is managed (e.g., distributed) but is directly placed inside a CM system (e.g., status reports, temporary results, e-mails, etc.).

3.3.6.1 Define a configuration management strategy

Define the CM process as it pertains to the system life cycle, including how it will address confidentiality, integrity, availability, authentication, accountability (including non-repudiation),and auditability. Determine the assurance-relevant attributes, the timescales for implementation of the CM process, and the tools required to support them. There may be a need to adjust CM requirements or approach based on what tools and techniques are available.

The CM processes should be maximally automated, particularly for electronically stored items, because it is easier to make unintentional mistakes and to maliciously manipulate manual processes. Automation provides the additional benefit of reducing the incentive for circumventing the CM process. Note, however, that approaches that can automatically reconfigure systems without appropriate review of those changes, and their negative impact on the system assurance case, should be avoided where practical.

Strategies for CM processes and supporting tools should incorporate the following:

- Use strong per user authentication. If passwords are used for authentication, they should always be encrypted for transmission over a network, and never stored as plain text anywhere.
- Centralized repositories should be hardened from attack. For example, on the platform supporting the centralized repository, limit the number of other services being run to reduce the risk that these other services could expose the repository to attack.
- Restrict and monitor network access for the CM system.
- Establish quality acceptance criteria to avoid introducing inferior code which is easily compromised.
- Audit the CM administration measurements for metrics on access and updates to the data, to determine if there is unexpected or unusual activity (e.g., activity with unusual times, locations, systems, or people; individuals updating an unusually large number of CIs, etc.).
- Maintain configuration control for operational system and sub-system settings.
- Adjust the granularity of the CM system to more easily provide input to the assurance case.

The CM strategy should identify how to determine who can do what, the roles, attributes, and division of responsibilities. Determine the process used to vet/accept elements for certain uses, e.g., where two-person controls and/or approval by configuration control boards is required, to reduce the risk of insertion of malicious elements.

The strategy should discuss how to perform CM of OTS elements, including COTS, and how changes in OTS elements impact system assurance. Determine how OTS versions are identified and how new patches, bug fixes, and upgraded/new versions of OTS elements are managed. See the acquirer process for information on selecting suppliers.

Identify how to get relevant information from the CM process into the assurance case, including during maintenance.

Patch management's speed should be commensurate with its associated risks. Overly hasty patch management can cause deployed systems to fail to perform their missions and/or insert other vulnerabilities. Conversely, excessive caution in patch management can unnecessarily expose a system to attack by excessively delaying deployment of needed patches. Thus, a CM process for deployed systems must have a triage process to determine which of several patch deployment processes should be used. This triage process should be part of risk management. For example, changes may be fielded immediately or deferred to a later major release.

Systems of systems in which there is an incremental development and deployment model bring in additional complexity. These different systems may have different owners, which results in CM complexities. Should such a situation occur, attempt to establish a relationship with the other entities to define the CM controls for the SoS. CM control may be either hierarchical or distributed, based on the established relationships. It may require a combination of manual and automated approaches. The CM strategy should encompass any impact on the system assurance, based on interoperability within a SoS context. Determine the strategy and plan for horizontal and vertical CM, in particular, how to identify the configuration items that must be protected on behalf of other systems (within the SoS) and vice versa.

3.3.6.2 Identify items subject to configuration control

Configuration items should be uniquely identified to enable configuration control. Identify the configuration items (CIs) that are critical to system assurance, so that they can be protected as necessary. CIs should include defensive functions, build instructions, documentation, the assurance case, tools (including development and CM tools), and the configuration baselines. Delivered systems should include version information sufficient to identify exactly what version was delivered.

3.3.6.3 Maintain configuration information with integrity and security

Protection of configuration item data and meta-data, both in repositories and under modification, must be maintained. Controlled entry points for those needing access to the CM data should be defined.

Physically protect CM systems (e.g., by locking them up) and control access to the system. Ensure that staff cannot add snooping devices (e.g., keyboard loggers or hardware key loggers), reboot systems to unprotected states, and so on, to gain access.

Establish processes to help counter data loss or subversion of a CM repository. Back up repositories, and automatically compare backups with current versions to detect tampering or data corruption. Consider having developer systems compare centralized repositories with their claimed changes as an integrity check. For example, consider using CM tools that support cryptographically “immutable” chains, so an attacker must create “new” entries (which are more visible) instead of silently modifying old information. Establish a process for reconstructing a repository when subversion has been detected. When subversion occurs, determine root causes, and perform corrective actions to address them, possibly including changes to the CM processes. These are to counter the risk of attacks targeted at the CM repositories, directly or indirectly (e.g., via CM administration functions or other services), whether internal or external.

For OTS CIs, detailed element information may not be available. For example, source code and details about hardware and Internet protocol cores are often considered proprietary information that is not released.

Audits should be used to ensure that the system assurance has been maintained by comparing the expected baseline with the actual deployed system configurations.

3.3.6.4 Ensure changes to configuration baselines are managed

Manage all configuration changes, both of configuration items and of baselines (configurations of configuration items). This includes ensuring that they are properly identified, recorded, evaluated, approved, incorporated, and verified.

To prevent unauthorized changes, the CM system should practice least privilege. For example, changes should be permitted only after considering the role, attribute, or identity of the requestor, along with the configuration item or baseline being modified.

Record every change made, including who made the change, when, and exactly what changed. This information must not be forgeable, and must be protected to support accountability. For example, if it is later determined that a particular developer or supplier is malicious, it must be possible to quickly identify the changes made by that developer or supplier (many CM tools have an “annotation” or “blame” function to support this). As noted earlier, recording this information should be automated as much as practical so that it is easy to use. All changes to each baseline should be traced to the previous baseline(s), and maintained with integrity.

Where possible, report to users “last change from you was (time stamp) from (source location)” information when a new change is submitted, so that users can immediately notice any discrepancy. Track modification requests and changes, especially vulnerability reports, so that any vulnerability can be shown to have been addressed (if it has been).

Attackers can introduce malicious information via a CI that is not managed or controlled by the programme or system owner. Programmes should be able to identify the suppliers of the system and its elements. In addition, they should have a way of contacting the supplier for each CI to address any possible issues that may arise.

Suppliers of OTS elements may make updates to their elements without providing detailed information about those changes to their customers. These changes can adversely impact assurance. Mitigation approaches should be used to counteract this potential impact. Potential approaches include:

- Developers may enter into agreements with their suppliers to obtain additional information (in particular, for critical elements) where necessary to support system assurance.
- Documentation and audits may be required, where necessary.
- Architectures may be created to mitigate this lack of information (see the architectural design process section).
- Additional verification processes may be used to determine that updated elements meet necessary requirements (see the verification section).
- Require/develop standards/specifications that enforce required attributes.
- Identify suitable substitutes.

As business case and relationship supports, customers/developers may work with their suppliers to obtain additional information (in particular, for critical elements) where necessary to support system assurance.

Update notifications, including those delivering vulnerability repairs, vary from supplier to supplier, depending on their update programmes. If the frequency is out of sync with system needs, either too frequent or not frequent enough, it is the responsibility of the PM/SE/System Administrator to monitor the update OTS update programme and determine the appropriate policies and procedures for OTS maintenance.

3.3.7 Information Management

Information management (IM) (ISO/IEC 15288:2008, Section 6.3.6) provides relevant, timely, complete, valid, and, if required, confidential information to designated parties during and, as appropriate, after the system life cycle (ISO/IEC 15288:2008). The IM process is interlinked with the CM process. CM is the framework in which individual information elements are contained; collections of information are managed through the framework.

3.3.7.1 Define the items of information that will be managed and the IM strategy

Information is defined broadly and includes information about information (e.g., meta-data). It may require special protection from disclosure, to prevent aiding system compromise by the active adversary. Some information may need to be designated as requiring special protection and may be marked that way.

Marking examples include:

- Proprietary
- Classified (Confidential/Secret/Top Secret)
- Critical Programme Information (CPI)
- FOUO (For Official Use Only)

Information examples include:

- System assurance case
- Source code
- Software documentation
- Information about suppliers (particularly those justifying suppliers' trustworthiness)
- Information from suppliers regarding their products/services
- Programme plans, such as programme protection and the information security plan
- Certification and accreditation (C&A) documents
- Threats (including raw and analyzed data)
- Anti-tamper techniques being used
- Residual system vulnerabilities
- Bug Reports

The strategy for IM should be coordinated tightly with the CM strategy.

3.3.7.2 Designate authorities and responsibilities regarding origination, generation, capture, archiving, and disposal

Some information that impacts assurance must have controlled and limited access. The designated authorities must be cognizant of their responsibilities to safeguard the confidentiality of such information. Do not forget to identify the authorities required for information maintenance (including archiving and disposal).

3.3.7.3 Define the rights, obligations, and commitments regarding retention, transmission, and access

Access to receive some information may require ensuring that only authorized recipients receive it. More sensitive information, including information that impacts assurance, may need to be encrypted, both when transmitted and when at rest. Such information may need to be authenticated for its integrity and/or non-repudiation (e.g., with digital signatures). It may be useful to limit the information about the system, its elements, and its configuration, since such information is useful to an attacker (who could use this to identify likely vulnerabilities). Changing some information may require special processes, e.g., two-person controls.

Review the necessity for proprietary, vendor-unique interfaces or formats, which may limit the ability of preventing change or move to another element (e.g., due to copyright, patent, rights). Otherwise, if a serious vulnerability appears in that product, or the supplier stops supporting it, it may be very difficult to switch suppliers. See the agreement process for more information on how to manage suppliers.

3.3.7.4 Define the content, semantics, formats, and medium

Define the encryption algorithms and formats for encrypted data. Consider the issue of key recovery (which permits emergency access to encrypted data), and the trade-off between enabling key recovery and keeping the information confidential. In cases where imprecise semantics could result in system compromise, the use of formal methods to define semantics may be worthwhile. There may be a need for additional filtering mechanisms (and/or outright forbidding) for certain data formats that have a propensity for carrying attacks (e.g., image formats such as graphic interchange format (GIF)).

Protect the physical medium used to store important information (e.g., laptop storage devices, USB sticks, backups). In some cases (particularly for backups) they should be both physically protected and encrypted at rest. For mobile media, consider policies that require filtering information so that sensitive information is not included unless necessary (e.g., blanking out personal information, anti-tamper information, assurance case information, and test results).

3.3.7.5 Obtain the information

Obtain information (commensurate with its importance) using methods that reduce the risk of invalid or manipulated information. Examples include using trusted parties, validating with alternative sources, using digitally signed data (and checking the signatures). Avoid using sources that have an incentive to provide misleading information.

The process of obtaining information (e.g., using search engines with certain keywords, or carefully reviewing specifications on specific products) may intentionally reveal important characteristics of the system to unauthorized parties. Revealing such information could eventually lead to the compromise of such systems. (See also Section 3.3.6, Configuration Management.)

3.3.7.6 Define information maintenance actions

Ensure that any personnel and processes involved in archiving, re-recording, or transforming the information will not modify or redistribute it inappropriately. This is especially important to consider if these actions are done by third parties. In many cases, a system developer or operator must consider the acquirer’s data protection requirements, so that the acquirer’s data are not compromised.

3.3.7.7 Retrieve and distribute information as required

Ensure that only the appropriate information is provided to authorized parties. In particular, ensure that providing information of one kind does not accidentally reveal information of another kind. In some cases, information may need to be filtered out or intentionally separated.

3.3.7.8 Provide official documentation as required

Mark official information so that it is difficult to alter, e.g., digitally sign official documents or use holographic seals. Or, at least ensure it is difficult to alter or receive while in transit, e.g., using encrypting and authenticated Web servers (e.g., SSL/TLS, OpenSSH, etc.).

3.3.7.9 Archive designated information

The archival systems themselves must be protected from attack. See also Section 3.3.6, Configuration Management.

3.3.7.10 Dispose of information

Unwanted information may still contain sensitive information, so it should be destroyed at the end of its retention period using appropriate disposal methods (e.g., shredding paper, melting/burning hardware, etc.). For more information on disposal, see ISO/IEC 15288:2008, PMBOK (2004), and INCOSE (2005).

3.4 Technical Processes

This section mirrors the ISO/IEC 15288:2008 Section 6.4, Technical Processes. Table 3-3 maps guidebook subsections against ISO/IEC 15288:2008 Section 6.4 and IEEE 1220 Section 6. Correlations to other guides and standards will be added as the need arises.

Table 3-3 SA Guidebook Technical Processes Mapped to ISO/IEC 15288 and IEEE 1220

Guidebook Subsections (of Section 3.4)	ISO/IEC 15288 Technical Processes (Section 6.4)	IEEE 1220 (Section 6)
3.4.1 Stakeholder Requirements Definition	6.4.2	6.1 Requirements Analysis
3.4.2 Requirements Analysis	6.4.3	6.1 Requirements Analysis 6.2 Requirements Validation
3.4.3 Architectural Design	6.4.4	6.3 Functional Analysis 6.4 Functional Verification 6.5 Synthesis 6.6 Design Verification 6.7 Systems Analysis
3.4.4 Implementation	6.4.5	No match
3.4.5 Integration	6.4.6	No match
3.4.6 Verification	6.4.7	6.4 Functional Verification 6.6 Design Verification

Guidebook Subsections (of Section 3.4)	ISO/IEC 15288 Technical Processes (Section 6.4)	IEEE 1220 (Section 6)
3.4.7 Transition	6.4.8	No match
3.4.8 Validation	6.4.9	6.2 Requirements Validation
3.4.9 Operation (and Training)	6.4.10	No match
3.4.10 Maintenance	6.4.11	No match
3.4.11 Disposal	6.4.12	No match

The technical processes (systems engineering process), as noted in the ISO/IEC 15288:2008 standard, is used to:

- Define the requirements for a system.
- Transform the requirements into an effective system.
- Permit consistent reproduction of the system where necessary.
- Use a system to provide the required services.
- Sustain the provisions of those services.
- Dispose of a system when it is retired from service.

The technical processes as they relate to system assurance are tied to the criticality of a given system and its elements. Mission/business need determines the criticality for a system. In particular, what is the effect of the loss or degradation of that system?

A threat leads to a risk. Criticality helps determine the response to the risk. Assurance supports risk management. A notional example of how a system's criticality may be determined is shown in the table below.

Table 3-4 System Criticality

Criticality	Definition	Tolerance	Example
Highest	The compromise of a system or elements within a system results in direct catastrophically degraded mission/business capability of system/mission failures.	Seconds to minutes	Denial of service of an on-line Securities Transaction Systems
High	The compromise of system or elements within a system potentially results in seriously degraded mission/business capability of system.	Minutes to hours	Residential electrical power loss or brown-outs based on an IT compromise

Medium	The compromise of system or element potentially results in partial or identifiable degradation to mission/business capability of system.	Hours or days	Loss of function of traffic lights in a city due to an IT system compromise
Low	The compromise of system or element potentially results in an inconvenience.	Days	

A higher level of criticality requires a higher level of assurance. This distinction must be reflected in the requirements.

3.4.1 Stakeholder Requirements Definition

The purpose of the stakeholder definition process is to define the requirements for a system that can provide the services needed by users and other stakeholders in a defined environment. (ISO/IEC 15288:2008, Section 6.4.2). There is a relationship between the criticality of the system and the level of confidence necessary to achieve system assurance.

The following considerations enhance the requirements process to address system assurance:

1. Identify specific stakeholders or stakeholder classes who have an assurance perspective. In general, stakeholders or stakeholder classes, while they may not express requirements in terms of system assurance, should be able to identify their tolerance for failure, degradation, compromise or loss, e.g., degraded modes of operation.
2. Identify threats, vulnerabilities, or weaknesses. The stakeholders should use current or previous experience in dealing with similar systems to help with the identification. The stakeholders then need to define appropriate requirements to address the threats, vulnerabilities and weaknesses identified.
3. Identify mitigation and/or compliance requirements, such as anti-tamper requirements
4. Consider the operational environment in which the system will reside, and that environment’s impact on system assurance, when defining the set of stakeholder requirements.
5. Provide a framework of requirements to validate that the system does what it is defined to do and nothing else, in its intended operational environment.

After the requirements have been collected, the next step is requirements analysis.

3.4.1.1 Building the Assurance Case

See Section 3.4.2.4, Building the Assurance Case section of Requirements Analysis Process.

3.4.2 Requirements Analysis

The purpose of the requirements analysis process (ISO/IEC 15288:2008, Section 6.4.3; IEEE 1220) is to derive measurable technical system requirements specifying the characteristics the system is to possess, based on the elicited stakeholder needs.

3.4.2.1 Define the system functional boundary

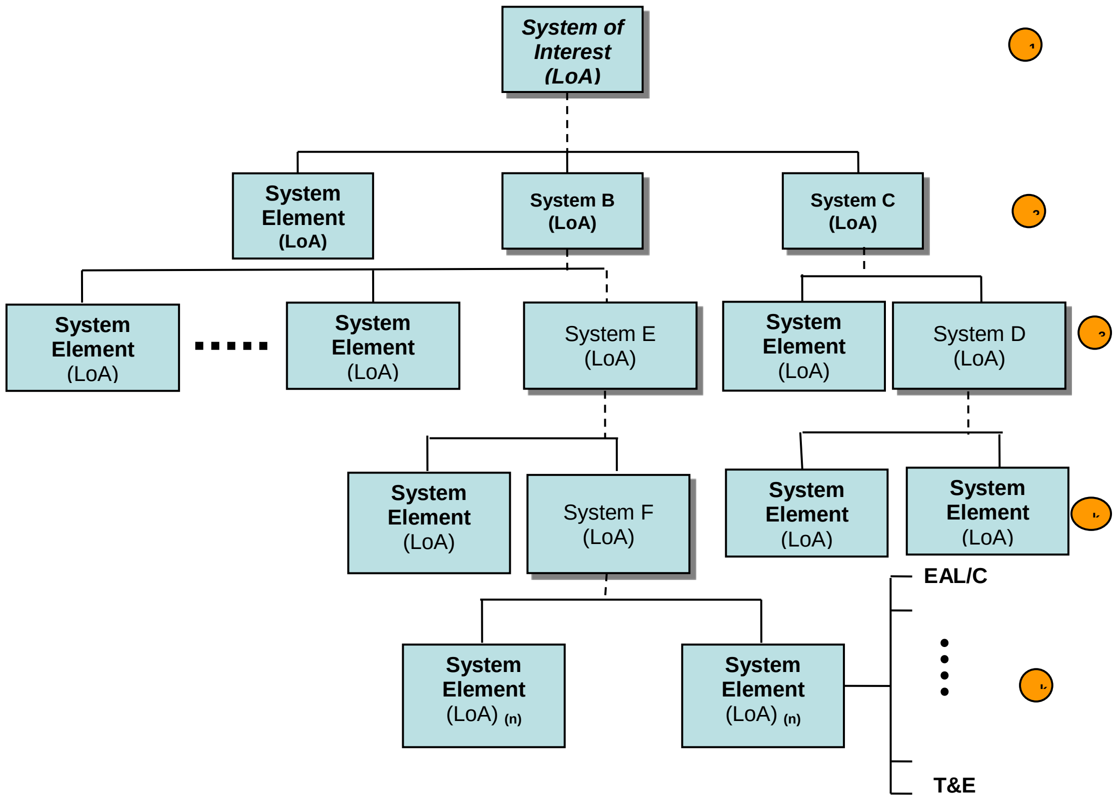
Clearly define the system boundary, and in particular identify what is and is not in the scope of the system, based on stakeholder needs. To develop defense-in-depth, individual systems need to implement assurance in coordination with protection mechanisms that are provided within its supporting infrastructure. The system will leverage many services from the environment, and for assurance it is important to identify which services are being leveraged vs. which services are being provided by the system itself. For example, does the system leverage existing cryptographic functions, operating systems, hardware platforms, I&A mechanisms, audit, and/or other network security services? Or, is the system providing them? For assurance purposes, the system must often be robust in the presence of attacks from portions of its environment, and these portions should be identified. The system may depend on properties of the environment, but these dependencies must be documented and reasonable.

3.4.2.2 Define required system functions

As the functions are defined, they must be further broken down into lower-level sub-functions (Figure 3-2), so that an analysis of alternatives can be performed (including build vs. buy decisions). These functions should include both:

1. Intrinsic functions, which directly deliver mission functions
2. Defensive functions, which secure a system and in turn deliver assurance

Some functions may be using COTS elements. In many cases the use of COTS elements should be encouraged, but if the COTS element can affect system assurance, the requirements on the COTS elements must be included in the system requirements. Issues to consider include additional functionality in the COTS that might negatively affect assurance, patching/upgrades (including whether the supplier controls this process), and default security settings. Building the system breakdown structure aids the SE by making explicit the specific functions, their inter-connectivity, associated criticalities of sub-functions, and the reasoning behind the criticality decisions. This breakdown structure provides a more manageable approach to building assurance into the system, thus assisting in providing the framework for developing the state of the system.



Note: Reference to Level of Assurance (LoA) of System of Interest, System and System Elements is to address notional view of a LoA. LoA for a System of Interest is a derivation of System and System Element level LoAs and the composability of those systems.

Figure 3-2 System Function and Level of Assurance Breakdown Structure

3.4.2.3 Identifying critical functions

A function is a critical function if it is (1) an intrinsic function that if compromised jeopardizes the mission, or (2) a defensive function that is protecting a critical function (which may be intrinsic or defensive).

The functional WBS should be augmented to factor criticality and the derived assurance criteria into the functional breakdown. This can be done by identifying the critical functions in the work breakdown structure.

Any assurance functions should be included in the Functional Baseline. The Functional Baseline should identify which functions have been determined to be critical.

Work to minimize the critical functions within a system in terms of number, size, and complexity. This minimizes cost, time to delivery/time to market, performance impact, and risk. By minimizing critical functions, the assurance case and traceability tend to be easier to manage. Note though that systems that are not known to be explicitly targeted should not be completely ignored as targeted systems may depend on non-targeted systems to accomplish their mission.

A single point of failure should be avoided if possible. Ideally, the system would implement layered levels of criticality and element redundancy, supporting graceful degradation while accomplishing the mission. Ideally, the failure of a single function might degrade system performance but not cause the loss of the system or the mission.

Define and extend the assurance case claims to identify derived system assurance requirements for all critical functions. Make sure that threats, intentional and unintentional vulnerabilities, and weaknesses are addressed.

3.4.2.4 Building the Assurance Case

The requirements definition and analysis process is where the assurance case begins to be addressed and where system claims are defined and identified as a subset of the system requirements. The functional breakdown of the system is performed, aiding in the identification of the critical functions in the system elements that need to be assured. The claims decomposition must be coordinated with the functional breakdown of the system, which leverages both the top level and technical requirements of the system.

- Engage stakeholders to establish the system top-level assurance claims under the constraints of budgets, time-to-market, implementation feasibility, as well as the ability to prove the claims. These claims should be directly linked to the assurance related the defined system requirements.
- Correlate the top-level claims with both the general and technical system requirements to review the accuracy of the claims.
- Decompose the top-level claims down into sub-claims and lower level sub-claims, and arguments. Consider whether adequate evidence is obtainable for the claims. Claims development is an iterative process similar requirement definition. And as such this effort may take several cycles.
- Bound the claims, sub-claims, and arguments in parallel with function criticality to make the assessment of threats, potential weaknesses, and associated vulnerabilities manageable.
- Link any assumptions made at each level of decomposition to sub-claims and arguments.
- Obtain stakeholder approval on the claims breakdown, assumptions, and the types of evidence to be collected.
- Restructure any claims that cannot be justified by collectable evidence.

3.4.3 Architectural Design

Architectural design is an important process that bridges identification of requirements to implementation. Unfortunately, this process is sometimes performed inadequately in a rush to production to meet time constraints. In addition, it is easy for a programme to become dominated by its features or feature sets, especially in a time-critical operational environment, and non-functional requirements are sometimes shortchanged.

This section describes practices in architectural design that can improve assurance. The organisation of this section is based on ISO/IEC 15288:2008 (Section 6.4.4), but some additional detail is extracted from IEEE 1220 (Section 6.3–6.7), which provides more detail in some areas.

3.4.3.1 Define appropriate logical architectural designs and methodologies

When identifying the architectural elements and their interactions, consider the following general issues:

- Attempt to separate elements that are highly critical from those that are not, and attempt to make the critical elements easier to assure by making them smaller, less complex, and more isolated from impact by other elements. Consider separating data, limiting the data and control flows, prioritizing the critical flows, and ensuring that such flows can be predictably and securely delivered (with confidentiality, integrity, availability, authentication, accountability (including non-repudiation and auditability). Assurance efforts may need to focus on these critical “intrinsic” elements and flows.
- Separation and information flow are key. Separation mechanisms also include encryption, physical separation (e.g., different cables or machines), and mechanisms incorporated in the central processing unit.
- Include defensive elements (elements whose job is to protect elements from each other and/or the surrounding environment, or to monitor elements). This particularly includes defensive elements that examine inputs from potentially untrustworthy sources, so they can limit inputs to valid ranges and formats. Determine what defensive functions can be leveraged across multiple elements or the entire system. Assurance efforts may need to examine the defensive elements necessary to protect and monitor the critical elements.
- Identify elements and their interfaces to accommodate requirements, including not just functionality but also protection from abuse cases (as determined in the requirements).
- Individually secured elements are not enough; the composition of the elements and their interconnections also matter. Understanding the interrelationships between elements and their linkages will help in addressing potentially weak areas in the design.
- Use defense-in-depth measures where appropriate, i.e., use multiple independent layered mechanisms to protect a system’s critical functionality to make it more difficult for attackers to succeed.
- Beware of maximizing performance (including throughput) to the detriment of assurance. Consolidation often improves performance (compared to

methods such as data/process separation), but users are not aided by high-performance subverted systems.

- Support some level of randomization and non-deterministic assignment of resources to help limit effectiveness of attacks that rely on the use of architectural knowledge (e.g., buffer overflow).
- Support identification and measurement of a trusted baseline so corruption can be identified.
- Support identification of corrupted and tainted resources and functionality, and recovery mechanisms to reestablish the system based on a trusted baseline.
- Collect additional information on assets, threats, and weaknesses of the architectures under consideration.
- Use the refined assurance case to derive additional requirements and design constraints.

Some specific approaches to consider when developing the architecture are:

- *Least privilege.* Elements should ideally be granted the least necessary privilege. Mechanisms for implementing least privilege include mutually suspicious elements (i.e., elements are designed to treat each other as untrusted or with limited trust), limited interfaces (instead of permitting broad direct access to a process or data, e.g., data in a database), limiting file system privileges, languages that enforce limited privileges, and so on. Access control mechanisms must be identified and be non-bypassable and tamper-proof. There are many mechanisms for addressing access controls depending on the development platforms used, including role- or attribute-based mechanisms. These in turn typically tie back into identity management.
- *Isolation/containment.* Isolation/containment mechanisms such as sandboxes, jails, layered interfaces, computer language security managers, element wrappers, containers, virtual machines, firewalls that are internal to the system, and so on can limit the damage that a less-trusted element might impose on the system. This may be especially important when integrating COTS or GOTS elements (in some cases GOTS may have a lower risk of subversion, but in such cases there is still a risk of unintentional vulnerabilities).
- *Monitoring and response for both legitimate and illegitimate actions.* Systems should detect both legitimate and illegitimate actions (including some faults), be able to record them, and potentially respond to the actions. A response may report, notify personnel (e.g., alarms), log in an audit record, block, hinder, or slow activity, perform recovery, or make other effective responses. This requires elements that can detect, record, and respond to such events and, typically, a communication path for their interconnection. Many of today's COTS products and platforms provide both audit and access control mechanisms that can be leveraged for both

the system design and development. Where possible, consider using such existing mechanisms, in part because using them enables use of many industry tools (which link to these mechanisms) for monitoring the system in its operational environment.

- *Tolerance.* Systems should include tolerance for some bad events, including security violations. In many cases, it is better to provide graceful degradation or reduction in service to a minimal level, instead of a denial of service. Possible tolerance approaches include intrusion tolerance (including tolerance to partial attacker success), fault tolerance, robustness (input tolerance), damage isolation or confinement, continuation of service although possibly degraded, redundant operations/procedures (which may only provide reduced service), recovery of the system to a legitimate state (including "self-healing" approaches such as periodic restarts from safe states), and disaster recovery. An example of intrusion tolerance is a security gateway appliance that offers firewall, intrusion prevention, anti-virus, and intrusion detection services that operate as independent services. A failure of one feature does not impact operation of the other security features. Additionally, if the hardware-based features like intrusion prevention fail, the system will revert to software intrusion prevention services. Isolation/containment mechanisms can be leveraged to aid tolerance. In addition, during deployment and maintenance, it is feasible to configure solutions to obtain both disaster recovery and continuation of service – reduced or full.
- *Identification and authentication mechanisms.* Identities should be defined and known for the entities involved; and their privileges or authorizations must be known or able to be established. This often requires some I&A mechanisms; more critical functions may require stronger mechanisms such as multi-factor authentication (e.g., a smartcard plus password) or integrating into a public key infrastructure (PKI) instead of simple password mechanisms. Operational environments may already provide some I&A mechanisms; normally in such cases they should be leveraged, but ensure that they provide the level of strength necessary to meet the need for system criticality or determine how such mechanisms can be strengthened.
- *Cryptography.* Cryptography is a foundational security element that is often needed to implement I&A, confidentiality, integrity, availability, authentication, accountability (including non-repudiation), and auditability. Although they have been widely used in the past, some cryptographic algorithms (such as Data Encryption Standard (DES)) and protocols (such as Wireless Equivalency Privacy (WEP)) are inadequate for today's needs. Consult with security experts when applying cryptography, since it can be easily misapplied, by architects and developers, without detection by system developers.
- *Deception.* While it provides no guarantees, deception may be useful for aiding concealment, intelligence gathering, confusing the attacker, and

wasting an attacker's resources. Use this type of resource to protect a system against intentional or unintentional attacks during deployment and maintenance. Examples include fake elements (e.g., honeypots and honeynets), falsified data, misleading service name/version number reports, and nonstandard ports.

- *Employ interface standards or standard elements*. In some areas there are standards for element interfaces and/or widely used elements. Using interface standards and/or standard elements for which an assurance case has been established often makes it easier to gain assurance, since typically the work to gain assurance can build on others' work. In addition, using interface standards may ease switching from one implementation to another if the first implementation has serious assurance problems.

9) Anti-Tamper Techniques

Anti-tamper techniques should be considered. Anti-tamper mechanisms can be divided into tamper-evident, tamper-resistant, and tamper-respondent mechanisms; systems may have multiple mechanisms. Examples of anti-tamper mechanisms include:

1. Tamper-evident (break-once) seals
2. Hard-to-forge images (e.g., holographs)
3. Protected enclosures (e.g., closed via epoxies, one-way screws, etc.)
4. Binding epoxy that encapsulates key elements (e.g., critical integrated circuits)
5. Protective packaging (coatings such as laminates)
6. Algorithms adjusted to have constant power use (to minimize emanations), countering power analysis techniques (including simple and differential power analysis)
7. Soft architectures – i.e., building generic hardware and/or software, and as late as possible specializing the system with specific software or configuration data and/or techniques. Examples include use of FPGAs, reconfigurable computing, and separately protected software plug-ins. This reduces the opportunity for tampering or unauthorized disclosure
8. Tamper-respondent enclosures (e.g., if someone drills into it, zero all data)
9. Use of multiple on-board monitors to determine if results are as expected, and respond (such as disabling device, removing sensitive data, and/or report incident) if results differ
10. Use of multiple mechanisms as part of a defense-in-depth approach

Other anti-tamper techniques include trusted processing, physical tamper detection mechanisms, the use of intrinsically tamper-resistant hardware elements/architectures, physical disabling/removing of access lines/ports, and encryption key hiding/management.

Some anti-tamper techniques can limit testability (such as time to perform diagnostics and ease of maintenance), maintainability (such as time to verify and validate fault and repair tasks/actions), or reliability (false alarms caused by anti-tamper checking). Consider trade-offs as well as process changes such as when or how to implement the anti-tamper techniques. Ensure that protected elements are not subverted before being protected by anti-tamper mechanisms, since such mechanisms tend to hide information that would expose such subversion. Also, ensure that the anti-tamper mechanism cannot be exploited by an adversary (i.e., an adversary should not be able to trigger a denial-of-service attack through the anti-tamper mechanism).

3.4.3.2 Partition and allocate identified system functions

ISO/IEC 15288:2008 reminds PMs and SEs to “Generate derived requirements as needed for the allocations.” When allocating requirements to architecture elements, it may become immediately obvious that certain elements/elements are much more critical than others; if they can be isolated from less-critical elements, assurance is easier to obtain.

Derived requirements often imply a verification approach, and may include measures of performance (MOPs) associated with assurance, such as:

- Use of certified elements (obtaining certification data as evidence).
- Static analysis tool results (when analyzing source or binary).
- Security or defensive functionality testing results.
- Fuzz testing requirements, possibly with specification of valid/invalid input ranges.
- Define acceptable input, as minimally as possible, and require the element to reject all input that fails to be acceptable. Avoid defining “unacceptable” input, even though this is logically equivalent, because attackers can often create malicious inputs that do not match a pattern of unacceptable values.
- Encryption for both data at rest and in motion, including encryption strength.

3.4.3.3 Analyze the design to establish element criteria

This analysis process must consider assurance, to determine the necessary criteria for each element and how the elements must be combined. The analysis must ensure that the architecture can meet assurance requirements, including a determination that it addresses all identified abuse cases. Use system assurance requirements, design constraints, and system assurance critical scenarios for architectural trade-off analysis and document the results. Include the following in the architectural analysis:

- Identify and evaluate architecture vulnerabilities.
- Allocate system assurance requirements to architectural elements.
  - Consider defensive architectural elements that allow an assurance requirement to be allocated solely to that architectural element.

  - Examine intrinsic and defensive functions.
- Perform Failure Mode Effects and Criticality Analysis (FMECA), fault tree analysis (FTA), anti-tamper analysis, and other applicable techniques to evaluate the architecture system assurance properties.
- For interfaces evaluate the filtering restrictions that the elements must enforce. For example, for data interfaces, identify the legal values (so that all illegal values will be rejected/trapped); for electrical connections, identify where voltage limits must be enforced.
- Update and refine the system assurance case with claims, arguments, and expected evidence for the selected architecture.
- Protect architectural information from potential adversaries by preventing public access to it.

As issues are identified, designs may be modified, and/or the identified issues are input to the risk management process (so steps may be taken to prevent, reduce the impact of, or accept the risks). Often, specific elements will be more critical, and these should be identified so that special efforts can be made to assure them, as discussed notionally in the requirements section of this document. Care must also be taken that critical protective elements, for example, authentication, are not bypassed and that needed separations among system elements are preserved. Some existing documents identify criticality levels that may be useful for identifying element design criteria (e.g., from catastrophic to no effect).

A variety of analysis techniques exist, depending on factors such as the type of system, technologies used, and issues being analyzed. These include:

- Threat analysis, including:
  - Threat analysis involves identifying a threat, modeling a set of possible attacks, assessing probability, potential harm of the attack, and trying to minimize or eradicate the threats
- System analysis, including:
  - Failure Modes and Effects Analysis (FMEA)):, but focused on assurance issues, e.g., both unintentional vulnerabilities and element subversion: http://www.fmeainfocentre.com/
  - Failure Mode Effects and Criticality Analysis (FMECA): http://www.fmeainfocentre.com/presentations/fmeca.pdf
  - Failure Modes and Impacts Criticality Analysis (FMICA): http://vva.dmso.mil/Special_topics/Risk/Risk.htm
  - Fault Tree Analysis (FTA): http://reliability.sandia.gov/Reliability/Fault_Tree_Analysis/fault_tree_analysis.html
  - Vulnerability trees: http://www.glam.ac.uk/socschool/research/publications/technical/CS-03-2.pdf

Architectures can be examined to determine if the principles of least privilege, limited data access, redundancy where appropriate, heterogeneity where appropriate, and so on, have been successfully applied. Where appropriate, consider adding “test interfaces” (to allow insertion or extraction of test data) and/or built-in tests, but ensure that these interfaces are properly protected so that they are not exploitable. Develop tests (including those for assurance) before implementation to increase the likelihood of correct results. Consider the design’s extensibility, particularly to determine the difficulty of supporting probable future changes, and to reduce the difficulty of maintaining assurance in those changes.

Critical elements may need to be hardened or have their criticality managed. Techniques such as graceful degradation, isolation, reducing single points of failure/multi-pathing, modularity, diversity, and use of interchangeable standards-compliant elements should be encouraged to significantly reduce the number, size, and/or impact of critical elements. Residual risks inherent in the use of less-assured element products can then be better managed. Isolation techniques can be applied at many levels, and the isolation mechanisms themselves need to be assured so that they cannot be easily subverted.

A result of this analysis is an argument that demonstrates that the assurance claims (requirements) have been met, and what evidence will demonstrate that the argument is valid (e.g., the framework of an assurance case). These design criteria are allocated to the appropriate system elements.

Design criteria may take other forms (e.g., requiring the use of static analysis tools on source code and/or binaries, fuzz testing, formal proofs of design and/or code, peer reviews, taint checking. See Section 3.4.3.2). Such criteria eventually link to the system verification process.

3.4.3.4 Determine which system requirements are allocated to operators

“Operators” include administrators as well as ordinary users. From an assurance perspective, operators must be granted the privileges they need to perform their tasks. In instances where these privileges are critical, information must be secure. Consider limiting the access of operators (possibly by separating their roles), and/or logging their activities, since not all people are trustworthy. Constantly asking users for permission to perform tasks should be avoided; users may not have the information to reliably make that determination, and over time may choose to automatically accept risks without thinking. See Section 3.4.3.1 regarding access control, including role-based and attribute-based access control.

3.4.3.5 Determine whether hardware and software are available off-the-shelf

As noted in ISO/IEC 15288:2008, these elements must satisfy the design and interface criteria. Off-the-shelf elements should be considered for an eventual make/buy decision. In addition to their functional requirements, however, there is a need to perform a supply chain analysis and for software, perform source code review(s), depending on the criticality of elements, to sufficiently assure them. (See Section 3.1.2, Supply Process).

As noted earlier, where interface standards exist, strongly consider using such standards. Standards tend to be examined and vetted through large groups, reducing certain kinds of risk. Where practical, limit element use to such standards so that elements can be replaced later with another standards-compliant element.

3.4.3.6 Evaluate alternative design solutions

ISO/IEC 15288:2008 notes that these alternative design solutions should be modeled "to a level of detail that permits comparison against the specifications expressed in the system requirements and the performance, costs, time scales (including time to market/deployment), and risks expressed in the stakeholder requirements."

It is a good idea to identify and review various architecture alternatives. Examining the set of critical elements (to potentially reduce their size or number) may be especially helpful in identifying alternatives. Architectural alternatives include "build vs. buy," and leveraging partial or full solutions. In all cases, consider the effect on assurance of these alternatives, including whatever mitigation techniques may be necessary.

3.4.3.7 Define and document the interfaces

ISO/IEC 15288:2008 notes that these interfaces are both "between system elements and at the system boundary with external systems." This documentation should include information on the assumptions necessary for assurance.

Identify opportunities for new standards, and include assurance considerations in those new standards. One challenge to using external elements is to know their relevant properties, so standards for such properties can be helpful. These include product-level assurance properties, expressing reference models/standards/requirements in modeling language, system interdependencies with the supporting infrastructure, susceptibility to external environment (e.g., EMI), and identifying methods for validating compliance with requirements/standards, using industry-developed tools. For example, it would be good to have standard isolation models, so element developers do not depend on access (es) they are unlikely to receive in real-time system operation.

3.4.3.8 Specify the selected physical design solution

ISO/IEC 15288:2008 notes that the physical design solution should be specified "as an architectural design baseline in terms of its functions, performance, behavior, interfaces and unavoidable implementation constraints."

3.4.3.9 Maintain mutual traceability between architectural design and system requirements

This traceability must be a living document, including traceability for assurance requirements, so assurance can be maintained through changes in requirements or implementation.

3.4.3.10 Building the assurance case

While performing architectural design, update the assurance case to include the necessary arguments (which justify why the architectural design will support meeting the assurance claims):

- The architectural design should identify the interfaces to untrusted sources; justify in the assurance arguments that this is the complete set of such interfaces. For these interfaces:
  - Identify the full set of elements that connect to or implement these interfaces.
  - Justify why the measures taken to harden these elements from attack are adequate, and
  - justify that whatever input or output filtering the elements will perform is sufficient.
- Some assurance arguments may be based on a defense-in-depth approach. In these cases, identify the primary hardening methods and explain why (1) an attacker would have to defeat all of the methods to succeed (instead of a subset) and (2) why this would be much more difficult than defeating just one method.
- Some assurance arguments employ element hardening, which often includes creating and enforcing least-privilege limits on elements connected to external interfaces. In these cases, explain why these limitations would significantly reduce the effectiveness of an attack or successful penetration.
- Externally-connected elements often cannot completely filter all possible malicious input. In this case, the assurance argument should demonstrate that the other elements that process the input will appropriately respond to malicious input (this is particularly a challenge for multimedia formats).
- Develop the assurance case, including its arguments, in a way that makes it easy to change. Use tools to maintain the assurance case, and divide up the assurance case into smaller modules so changes can be localized.

3.4.4 Implementation

During the implementation process (ISO/IEC 15288:2008 Section 6.4.5), the system capabilities, interfaces, and constraints are transformed into system elements (such as fabricated hardware (HW) and coded software (SW)) that satisfy requirements and the architectural design. It is a demonstrated fact that even a well-designed system can fail if its key elements are implemented poorly, which underscores the criticality of system assurance considerations during implementation.

No single assurance technique can prevent vulnerabilities during implementation. Therefore, various techniques are necessary, each of which has some probability for detecting and avoiding various vulnerabilities that would otherwise be inserted during implementation. Selected detection techniques should be sufficiently different so that they are not correlated in their likelihood of addressing a given type of vulnerability. Performing multiple techniques identified herein will decrease the overall probability of vulnerability insertion during implementation.

3.4.4.1 Implementation strategy (and integration strategy)

It is important to follow documented strategies for both implementation and integration. While the Implementation Process (Section 3.4.4) and the Integration Process (Section 3.4.5) are treated separately, the strategies required for both have many overlapping considerations for system assurance. Consequently, they are both treated in this subsection.

The Implementation Strategy and the Integration Strategy can either be part of the updated project plan or be generated as separate documents prior to implementation and integration. Specific considerations for system assurance should be included:

- The balance between assurance and system performance (carefully considering both).
  - Avoid high performance at the cost of subversion.
- Include mechanisms for revelation, isolation, and diagnosis of faults.
  - Implement elements and integrate the system for testability.
- Consider implications of fault-tolerant techniques that may disguise important warnings (e.g., of attack) or create vulnerabilities (e.g., if some elements ignore bad data but others do not, creating an inconsistent state).
- Assurance of custom-designed HW and SW for implementation/integration
  - Secure the development and integration environments and tools by appropriate isolation (physical security and access measures), even when the environments are Unclassified.
  - Include assurance-specific checklists into the SW peer review process.
  - Emplace tools to identify unintentional/intentional vulnerabilities.
    - ♦ (Identify expected outcome of tool usage, and what to do with it)
- Assurance of reuse HW and SW, including COTS and GOTS
  - Concentrate on the strategy for SOUP, as its ability for adaptation to adequately assured reuse may constrain the overall design.
  - Adaptation of reuse SW conducted with regard to applicable security and privacy standards.

  - Emplace tools to identify unintentional/intentional vulnerabilities.
    - ♦ Identify expected outcome of tool usage, and what to do with it.
- How system elements received from suppliers and/or retrieved from storage will be checked/certified for assurance purposes (e.g., they might be verified against acceptance criteria specified in an assurance agreement).
- Clarification of assurance-related issues to be addressed during the SETRs (Systems Engineering Technical Reviews) occurring during implementation and integration. Examples of the relevant SETRs might include:
  - Systems Requirements Review (SRR)
  - Systems Functional Review (SFR)
  - Preliminary Design Review (PDR) (down to the Configuration Item (CI) level)
  - Critical Design Review (CDR) (down to the element and unit level)
  - Test Readiness Review (TRR) (down to the lowest level of algorithms and boards)
- Protection of system boundaries and interfaces.
- Protection of the development and integration labs.
  - Establish a routine examination of all tools introduced into the labs.
  - Prevent “dumpster diving” by monitoring trash for unintentional “releases” (destroy trash).
- Appropriate system assurance training for staff
  - Provide counterintelligence briefings for staff who will interface with outsiders, as part of the supplier assurance effort.
  - Provide adequate training on assurance techniques and tools for effective application (e.g., difficulties for implementation and integration arising from maliciousness).
- Adequate CM-related access control and tracking measures
  - Limit privileges to authorized personnel.
  - Require two-person integrity, as needed.
  - Implement repository subversion countermeasures.
  - Configure HW and SW elements with regard to applicable security and privacy standards.

Tools are a key enabler for assurance. Although they are not a panacea, they provide both insight and an approach to reducing the risk of vulnerabilities. Industry provides some tools for the identification and assessment of unintentional as well as already known intentional vulnerabilities (e.g., viruses). Tools and techniques for detecting not-yet-identified malicious code (for example, by analyzing its behavior) still have limitations although continued efforts are being made to improve heuristics. In some cases tools for proving properties of designs or programme may be appropriate. Organisations should evaluate, as early as practical, the various options to determine if such tools are adequate for their specific purposes, and include this analysis in their strategy.

3.4.4.2 Constraints imposed by the strategies on the design

It is important to prioritize and handle the unavoidable constraints of implementation and integration that influence any system assurance requirements.

Difficulties for implementation and integration arising from maliciousness should be covered within the Implementation/Integration Strategy, including their effects in (potentially) constraining the design. For example, element and system realization (both SW and HW implementation/integration) can include malicious logic that causes:

- Loss of confidentiality
- Loss of integrity (e.g., changing information into invalid results or producing the wrong results)
- Loss of system availability (a critical vulnerability can cause entire system/mission failure)
- Possibility of repudiation

The adaptation of reuse SW must be conducted with regard to applicable security and privacy standards. This may lead to constraints on the system design. In fact, the strategy for adequate assurance of reused HW and SW elements in general (COTS, GOTS, and especially SOUP), may imply limitations in potential reuse, which in turn may constrain the overall system design.

Protection of system boundaries and interfaces may also lead to design constraints. Specific techniques should be employed to verify that interface breaches are prevented through adequate design and access controls (see Architectural Design).

3.4.4.3 Countering threats in implementation

Malicious SW or HW is difficult to counter because it is hard to detect. While code is being developed and HW elements are being fabricated, the following things should be kept in mind by the Lead/Chief Systems Engineer in guiding and overseeing the development team:

- Vulnerable elements can be extremely small relative to the size of the overall system. Today’s software and hardware systems are often large and complex, with many interactions. Yet it may take only a few lines of malicious code or hardware logic to cause great damage, and these elements can be distributed across the system, making vulnerabilities easy to hide and difficult to find.
- Effects are typically nonlinear. It is important to remember (especially with digital systems) that a slight change in input can yield massive effects.

- Malicious implementation can appear valid. Publicly known examples have shown that it is easy to create code that is malicious yet at first glance appears to be legitimate.
- Elements may lack transparency. Many commercial elements are delivered in forms that are difficult to analyze (e.g., binary-only forms for software and IP cores for hardware). For external services (e.g., software-as-a-service), there may not even be a binary that can be examined. This makes attacks easy to hide.
- Malicious implementation can pass tests but then fail when most critical. An intelligent adversary can develop attacks that will occur when the adversary desires—when it is the worst possible moment for the system. This requires considerations that are fundamentally different from typical safety analyses (which depend on probabilistic behavior).

The implementation process should incorporate frequent threat updates to ensure that countermeasures are developed based on the intent and specific threat agent capabilities that may change throughout the system life cycle. For different kinds of threats, different countermeasures and avoidance techniques apply, for example:

- **Threat: Unintentional vulnerability inserted during development**

  Countermeasure/Avoidance Techniques:

  - Prefer development tools/languages that prevent or warn of common vulnerability and error types. All programming and hardware description languages have constructs that are undefined, imperfectly defined, implementation-dependent, or difficult to use correctly. Both unintentional and intentional vulnerabilities may result (intentional vulnerabilities can masquerade as unintentional errors). The language should aid the design and development of reliable systems. The language should be designed to avoid error-prone features and to maximize automatic detection of implementation errors as soon as practicable. Risky properties of languages include failures to protect against boundary overflows and null pointers, or constructs that are easily misinterpreted. Some common constructs of C and C++ are prone to misuse, making the risk of including vulnerabilities especially likely in these languages. When using languages that are especially risk-prone, use additional techniques to reduce their risks. These include other techniques identified in this guidebook (e.g., static analysis tools, peer reviews/inspections, additional training, and specially skilled personnel). In some cases, a secure subset of a language may be appropriate. If switching to a language unfamiliar to the developers, include the necessary training for that language. It may be more cost-effective to retrain existing developers in a more-secure language than to attempt to implement additional necessary measures in a less-secure language. There is existing international standards work to develop guidance for avoiding vulnerabilities through language selection and use; if available, consider it. Before selecting tools/languages, be sure to include these additional measures in the total costs, since in some cases these additional measures may exceed the "savings" through using less-secure languages.

- Enable HW/SW compiler "warnings" as much as practical. These warnings should be enabled as soon as possible, especially for custom code, because "warnings" are often hard to address after implementation begins (due to the potentially overwhelming number of warnings to be addressed). Since compilers add new warnings over time, implementers may need to reevaluate which warnings are appropriate to include or add during maintenance.
- Train developers to recognize common mistakes that lead to vulnerabilities (e.g., buffer overflows, cross-site scripting, and integer overflows. Sources of such lists include CWE and the Open Web Application Security Project (OWASP) top ten.)
- Gracefully handle out-of-bounds data (SW) and signals (HW), e.g., reject such inputs and present an indicator that input has been rejected. Out-of-bounds data and signals should be logged for the purposes of detecting trends (such as attack indicators).
- Use tool-enforced HW and SW implementation guides to help avoid common mistakes and anomalies.
- Use static and dynamic tools to identify vulnerabilities. Vulnerability identification tools generate false positives and false negatives; therefore, use a suite of tools and emphasize eliminating the riskiest constructs.
- Select development tools with proven track records of low defect-insertion rates. (Some development tools can insert hard-to-find defects.)
- Perform peer reviews of implemented custom elements that search for vulnerabilities. (It may be useful to include security engineering in the peer review.)
- Perform periodic testing of elements for functionality and reliability requirements. Some additional test such as fault insertion may be key to detect malicious SW or HW threats.

➢ **Threat**: **Intentional vulnerability inserted by a solo authorized developer**

Countermeasure/Avoidance Techniques:

- Use CM tools to restrict access to software elements based on "need to know" and audit access.
- Use development languages that enforce limited access. (Programming languages like Java have mechanisms such as "private" and Security

Manager. In contrast, C++'s version of "private" is easily subverted by a malicious authorized developer through mechanisms such as pointer manipulation.)

- Use tool-enforced HW and SW implementation guides. (Tool-enforced implementation guides can prevent some hiding techniques.)
- Use tools to search for already-known as well as not-yet-identified malicious code. (Tools and techniques for detecting not-yet-identified malicious code (for example, by analyzing its behavior) are not mature.)
- Integrate advanced security testing techniques that can be used to identify zero-day vulnerabilities as well as embedded malicious code. Advanced security testing techniques may include fuzzing and automated source code analysis that can reduce vulnerabilities and deter adversaries by introducing the possibility of discovering covertly inserted code.
- Use paired development and perform peer reviews of implemented elements.
- Ensure that what is reviewed is the same as what is used (see CM section).
- For hardware fabrication, limit the access and controls provided to operators and/or include monitors (such as cameras) to increase detection of subversion.
- Implement frequent audit controls of processes/procedures.
- Test sampled hardware parts for critical test parameters.

➢ **Threat: Intentional vulnerability inserted during development by a solo unauthorized person**

Countermeasure/Avoidance Techniques:

- Enforce CM processes during development (see CM section).
- Ensure that the SW reviewed is the same as what is used (see CM section).
- Ensure limited access during hardware fabrication.
- For mass production, include CM procedures for difficult-to-forge markings on hardware elements (so that it will be difficult to swap them later).
- Use development languages and tools that enforce limited access.

➢ **Threat: Intentional vulnerability inserted during development by a supplier or supplier's supplier.**

Countermeasure/Avoidance Techniques:

- Use tools to search for already-known malicious code (e.g., viruses). (Tools to search for not-yet-known malicious code are extremely immature.)

- Perform additional checks and defensive measures at the implementation level for data and signals entering and leaving the element.
- The implementation process has difficulty countering this threat. Therefore, it is primarily countered by other system life cycle processes, such as architectural design, as well as supply.
    - ♦ Architecture and design will identify such techniques as sandboxing to reduce impact of malicious elements through containment.
    - ♦ Supply management processes will reduce the risk of malevolent products.

The above examples of threats and countermeasures are a sampling of the many possibilities and are not intended to be a comprehensive list.

Many implementation processes include implementation guides, which may also be termed “coding standards” or “style guides.” For assurance, these need to include information that helps counter vulnerabilities (e.g., counter the use of insecure constructs), and not be merely formatting guides. The guide should identify the subset of the language that is acceptable for use, and it should include the rationale for that subset.

Ensure that any assessment or evaluation of an OTS element is relevant to its intended operation in the system before depending on that assessment or evaluation.

Implementation issues are encountered cyclically within incremental and spiral development efforts and within the execution of system/element Engineering Change Requests (ECRs) in general. The level of rigor needed, vulnerability, and closure hazards of ECR implementation should be addressed by assurance engineering, and integrated into the systems engineering tasks of ECR analysis, design, implementation, integration, and testing. Trouble Reports and the follow-on Change Requests should be monitored for evidence of problems in assurance-critical functions.

Systems engineering personnel also get involved during the implementation process with assurance-related requirements that need to be addressed during the late life cycle V&V processes. The SE should verify that a test procedure exists to test each assurance-critical requirement, function, and signal or message. The SE should also develop regression analysis and test planning to ensure that a given element in new development/modification cannot adversely affect another element.

3.4.4.4 Record evidence that system elements meet assurance requirements

Document the successful execution of the assurance-related elements of the Implementation Strategy. In particular, record the assurance claims that can be made with respect to the implemented system elements, the arguments that justify those claims, and the evidence that supports the arguments.

Record any non-conformances introduced or discovered during implementation, and initiate action plans to resolve those issues that affect assurance.

3.4.4.5 Package and store system elements appropriately

Once implemented, it is important to package and store the system elements in accordance with prior assurance agreements. Part of the task of protecting system boundaries and interfaces includes packaging and storing the implemented system elements, prior to their transition to operations, by containment and conveyance methods that achieve the required level of assurance.

3.4.4.6 Building the assurance case

While performing implementation and integration[1], update the assurance case to complete the arguments as well as to add some of the evidence needed to justify the assurance claims:

- Complete the claims, sub-claims and arguments with a final verification.
- Begin collecting the evidence on the interfaces to trusted and untrusted sources, including evidence of implementation of defensive functions and secure interfaces. For interfaces to untrusted sources, identify the hardening and/or filtering approaches used.
- For defense-in-depth approaches, collect arguments and evidence. Evidence collected should support the justifications (arguments) such as the traceability to defensive functions.
- Adjust the arguments and evidence as appropriate where there is a changing (dynamic) enterprise environment. This is critical when leveraging existing enterprise infrastructure functions (e.g., network infrastructure, encryption, or identity and access management), to confirm that the infrastructure continues to support the assurance case.
- Collect evidence from static analysis. Such evidence may include results from inspecting for weaknesses, such as buffer overruns or not validating inputs.
- Review claims and arguments for feasibility. In some cases, it may be determined that some evidence is not collectable, triggering a need to update the claims and arguments.

[1] Many of the issues in building an assurance case are common to both the implementation and integration processes, so this document has a single section that covers both processes.

3.4.5 Integration

During the integration process (ISO/IEC 15288:2008 Section 6.4.6), a system capable of being verified against the specified requirements of architectural design is assembled and integrated. Hardware, software, firmware, and other system elements are combined to form complete or partial system configurations that create a product specified in the system requirements. It is essential during integration to ensure assurance of both elements and their interfaces. In today's environment, this integration will often also need to consider the integration of systems of systems.

It is arguable whether today's systems, and systems of systems, are more developed or integrated. As systems become more complex, their elements should be more modular, better defined, and more easily integrated, in order to ensure the tractability of the assurance task. In the interim, the integration process requires even more careful attention to system assurance.

Since a system is susceptible to threats posed by vulnerabilities and malicious actions during integration, a strong integration strategy should be defined and followed. Some elements may present additional risks; those risks should be assessed in terms of their impact on the system (or SoS), and mitigated (e.g., through isolation and containment mechanisms) during integration. System elements should be integrated with solid interface control descriptions and detailed assembly procedures, using clearly specified integration facilities. Further considerations on this defined strategy are given in the next section.

3.4.5.1 Integration strategy

It is important to follow documented strategies for both implementation and integration. While the implementation process (the previous major subsection of this guidebook) and the integration process (the current major subsection of this guidebook) are treated separately, the strategies required for both have many overlapping considerations for system assurance. Consequently, they are both treated in the Implementation section. (For details, see "Implementation Strategy" and "Constraints Imposed by the Strategies on the Design" in the Implementation section.)

3.4.5.2 Obtain system elements

If a supplier has not provided sufficient information to confirm the operation of its element and the ability to assure it, then the assurance case for the integrated system may be more uncertain. More of this concept is discussed in the Supplier Assurance section.

3.4.5.3 Ensure that system elements have been verified

Ensure that any assessment or evaluation of an OTS element is relevant to its intended operation in the system before depending on that assessment or evaluation.

3.4.5.4 Integrate system elements

For different kinds of threats, different countermeasures and avoidance techniques apply, as follows:

- **Threat: Malicious substitution of elements and the unintentional use of incorrect elements during integration**

  Countermeasure/Avoidance Techniques:

  - Ensure that assembly of the system is consistent with the architectural design.
  - Perform an integrity analysis (and corresponding C&A) of integrated products.
  - Evaluate COTS/GOTS product vulnerabilities.
  - Assure the supply chain by evaluating suppliers' assurance cases. Do not take the state of supplied elements at face value; rather, ensure that the system is not threatened by the state of its integrated elements.
  - Identify and analyze public domain software to be integrated (including modifications to previously approved products).
  - Assure integrity of SW versions and HW items through adequately robust CM tools.
  - For mass production, include CM procedures for difficult-to-forge markings on hardware elements (so that it is difficult to swap them).
  - Perform tests as required to ensure the functionality and reliability of critical parts. Some additional test such as fault insertion may be key to detect malicious SW or HW threats.

- **Threat: Malicious or unintentionally incorrect system composition**

  Countermeasure/Avoidance Techniques:

  - Ensure that assembly of the system is consistent with the architectural design.
  - In integration facilities, limit the access and controls (possibly including monitors, such as cameras, to increase detection of subversion).
  - Revisit the "least privilege" criteria and test at a system level.
  - Ensure (perhaps through adequate architecture) that the security model is correctly implemented and that received transactions are expected and validated.
  - Ensure clear and detailed build-and-release procedures (including checks and balances) through adequately robust CM tools.

    Detailed Consideration: At each level of integration including the system level, it will become clear whether the various elements developed and bought off the shelf can be configured in the same pattern that the architecture specifies. Systems often enter integration as incomplete conglomerations of code and parts that require finishing activities.

Integration engineers may generate their own concepts of how to perform this finishing job, which in turn may weaken the assurance case (because there is no documented trail from architectural decisions through design and implementation to integration).

- Document all integration decisions (that finish the design-to-architecture fit) and relate them back to previous decisions. Update and evaluate the architecture, if necessary.
- Include integration engineers in the systems engineering process to affirm and reinforce the assurance case and to ensure configuration and integration information is well-documented and evaluated.
- Use the CM system to document and track architectural changes made to finish the integration (to support evaluating the assurance case).

➢ **Threat: Malicious or unintentionally incorrect interfaces**

Countermeasure/Avoidance Techniques:

- Assure that assembly of the system is consistent with the architectural design.
- Test all interfaces between elements.
- Perform network security testing and accreditation.

  Detailed Consideration: An interface is rarely as simple anymore as one module calling another. Integration touch points exist throughout the system, between software and hardware, among programmable logic devices, and among major subsystems including databases, security systems, transaction monitors, and operating systems.
- Check each touch point during integration. The correct functioning of intermediate and final interfaces cannot be assumed (e.g., the functions of a well-defended element may be completely undone by a database with significant exposures).

  Detailed Consideration: Interface control descriptions must be comprehensive (including knowledge of element interoperability and data transfer protocols) in order to avoid such problems as can be caused by buffer overflows. Also, for multi-layer stacks (a typical communications and networking architecture pattern), good architecture will ensure that the layers underneath operate as they should; but in a "trust, but verify" posture, it is important for layers above not to over-rely on this security.
- Integration engineers must verify all interface controls (which includes checking for the vulnerabilities of buffer overflows).
- Integration engineers should validate the specifications and agreements that document the data transfer in a multi-layer stack, including both explicit and implicit interfaces throughout the stack. In many cases, upper layers will need to validate their layer's data, since lower layers will not have the necessary information to do so.

- Examine or develop a security agreement when one sub-system exchanges sensitive information with another sub-system at a different location, particularly if it has different organisational controls.

3.4.5.5 Record integration information

Document the successful execution of the assurance-related elements of the Integration Strategy. In particular, record how security properties and behaviors have been maintained. More importantly, record any non-conformances due to integration activities, and initiate action plans to resolve those issues that affect assurance.

Document any further justifications for system assurance that have been identified during integration, together with supporting evidence.

Changes made during integration may affect assurance, so they need to be documented and related back to previous decisions, to update the architecture if necessary, and evaluate the assurance implications of that change.

3.4.5.6 Building the assurance case

See “Building the Assurance Case” text in the Implementation section (Section 3.4.4.6).

3.4.6 Verification

The verification process (ISO/IEC 15288:2008, Section 6.4.7) is needed throughout the development of the system to ensure that the design solution has met the system requirements and to provide assurance that the system is being built “right.” A “verified” system is able to demonstrate that it has been implemented correctly and is compliant with stakeholder system requirements, as documented. Requirements should be traced to implementation and verification using a requirements verification matrix. The system assurance requirements should be tagged with attributes assigning them to/as SA. In general, this implies a need to conduct some form of assessment, such as developmental testing and evaluation.

The verification process activities must be conducted to provide objective evidence as the basis for the assurance arguments. For additional information on verification approaches see ISO/IEC 15408.

3.4.6.1 Define verification strategy and plan

The Verification Strategy and Plan must be enhanced to address any additional tools and techniques specifically needed to verify system assurance. Some verification techniques for system assurance may require additional element information than would otherwise be available; for example, when integrating COTS, access to source code may be necessary. The Verification Strategy and

Plan generic approaches may be modified to incorporate consideration of system assurance. Depth and breadth of verification should be commensurate with risk.

In particular, ensure that the strategy and plan for assurance verification can be executed by individuals with the appropriate skill sets for the assurance tools and techniques. Such skill sets, which are specialized and less commonly available, include understanding common weaknesses and security engineering practices, including more specific skill sets such as analysis of assurance tool output to determine their meaning and associated impacts, ability to identify probable attack scenarios, reverse engineering techniques, and malicious code analysis.

3.4.6.2 Identify and communicate constraints on design decisions

Highly critical elements imply the need to strongly limit their size and intrinsic complexity, so that thorough verification can take place with those elements. The availability of assurance tools and techniques that can be applied may significantly impact the design. For example, many static analysis tools only focus on a few weaknesses and do not cover many others (which would need to be countered in other ways); some development tools/languages facilitate vulnerabilities that are difficult to avoid and for verification tools to detect; there are no verification tools available for certain development tools/languages.

3.4.6.3 Ensure verification system is available and conduct verification

Verification should be done throughout the life cycle, following the verification strategy and plan. Ensure that the system, to perform verification, is available when needed (and potentially earlier for risk management):

- Generic verification tools that may be used in special ways for assurance or to help enhance assurance (e.g., generic bug-finding tools, testing that covers all branches, functional tests created to verify assurance assumptions).
- Security-oriented static analysis tools, e.g., those focused on identifying vulnerabilities. Many of these specifically search for common vulnerabilities (e.g., for software, those identified in the CWE). These may analyze binary and/or source.
- Security-oriented dynamic analysis tools. These include fuzz testers, which send large quantities of “random” data to see if the system fails in some catastrophic or insecure way. This may also include creating specialized attack scenarios as tests.
- Penetration testing and red team (aka security assessment). These involve teams to examine and attempt to penetrate the system. Identify procedures needed. Determine what is and is not in-bounds for such verification (e.g., is social engineering permitted), and how the issues not in bounds will be addressed instead. Minimize the limitations placed on red teams and penetration tests—since such limitations decrease the utility of the results—

and ensure that those limitations are communicated only to those who need to know. After all, attackers do not limit themselves in the same way.

- Peer review and desk checking. Ensure that assurance-specific issues are covered in peer reviews, including that any checklist used includes such issues. These issues should cover common design weaknesses and language-specific dangerous constructs, such as handling malicious inputs, preventing or handling buffer overflows, etc.
- More specialized verification tools may be used, particularly for high assurance, such as theorem checkers/provers and formal model checkers.

For additional information on verification for software assurance, see the SwACBK. Verification should include examination for maliciously inserted functionality. Ensure that what is verified is the element to be actually used in the system (e.g., a malicious developer may attempt to ensure that what is verified is not what will be used). Verification should also examine that the constraints/assumptions on design are actually met if they are required for assurance.

3.4.6.4 Document and deliver verification data

Document and deliver the verification data as objective evidence supporting the assurance argument. The set of evidence, once collected and correlated, should support the broader assurance argument(s) and system assurance claims. This evidence should include the system requirements verification matrix. This evidence should be considered part of the system, and as such, must have commensurate protection, e.g., from tampering, misuse, and/or hijacking.

3.4.6.5 Building the assurance case

While performing verification, update the assurance case to collect the evidence needed to justify the assurance claims:

- Demonstrate that the tools and techniques used to gather evidence are adequate to do so.
- Collect all static analysis and peer review results that support the assurance case.
- Collect test plans, approaches, and the associated test results that support the assurance case; these will typically be those for the critical elements (including defensive functions).
- Collect all penetration/red teams’ test scenarios and results.

3.4.7 Transition

Transition (ISO/IEC 15288:2008, Section 6.4.7) is the process of establishing a capability or service into an operational environment. This process must ensure that the system once transitioned is assured, and does not have a negative impact on the assurance of other systems in the operational environment.

3.4.7.1 Prepare a transition strategy

The transition strategy must consider those aspects necessary for system assurance, as well as prevent other systems at the site from being breached. Issues to be considered include:

1. Configuration of the system in its target environment; consider the integration required with operational enterprise services such as an existing PKI.
2. Assessment and documentation of system assurance impact of any changes made to the system or environment.
3. Test and evaluation of the system as installed at the site with a focus on validation of the system in operation, considering other systems' impact on assurance. For example, consider the impact of the underlying operating environment or the enterprise services provided such as I&A.
4. Training of users, operations staff, and maintenance developers on the aspects of the system necessary for assurance. Consider the training implications for the system interoperating with any SoS, enterprise, or any overarching supporting enterprise services.

3.4.7.2 Prepare operational sites

The operational site must be prepared to receive the system. This may involve the adaptation of the operational site. Ensure that any changes made in support of this adaptation do not negatively impact assurance of the system being deployed, the other systems, or the infrastructure of the operational site. Examples include the installation of upgrade patches to the other systems at the site, or adding network connectivity to the site, either of which can introduce system vulnerabilities. A site survey could also be included prior to delivery to facilitate transition planning. Ensure that any downtime or parallel use during installation does not expose the site to additional vulnerabilities.

3.4.7.3 Deliver the system for installation

The delivery process for the system, and upgrade/patches, must protect the system from tampering, fraud, and exfiltration. For information this is often done through cryptography. The system elements must be protected through the intermediate stages of delivery. This includes such events as tampering of the system while in temporary storage or being moved between development and operational site.

3.4.7.4 Install, demonstrate, and activate the system

During the installation process, the system must be hardened to resist attack. This includes the disabling of unnecessary services as well as leveraging and enabling enterprise-secure mechanisms in support of the system and defensive functions. Where there are choices, use the more secure mechanisms.

When installing patches and upgrades, consider the implication of these changes to the assurance of the system. For example, the operation of the various security mechanisms and defensive functions in support of system operation may be disabled or degraded through patching.

Test and evaluation of the system must include the test and evaluation of its assurance. Validate that the system's assurance is not reduced as it is interfaced to the user/operational site. Demonstrate that the services provided by the system are sustainable by the enabling systems in the operational environment.

Train the users, operations staff, and maintenance personnel to execute and maintain the system's assurance. Keep an open communication channel between the development organisation, transition, operational, and maintenance teams in support of assurance.

After activation, validate and demonstrate that the system assurance has been maintained. For example, the system and its surrounding subnet may be reexamined through the use of network, port, and vulnerability scanners to confirm compliance with security configuration policy in support of the system's assurance.

3.4.7.5 Record data

Ensure that transition information is analyzed for anomalies that may indicate attacks or vulnerabilities. Record and protect such data commensurate with its value to the system and its potential for abuse by attackers.

3.4.7.6 Building the assurance case

While performing transition, environmental chances must be scrutinized to ascertain whether they affect the validity of the assurance case's evidence, arguments, and claims. Update the assurance case to complete the evidence needed to justify the assurance claims and arguments:

- Include evidence that any changes made (to adapt the operational site) have a positive or neutral effect on assurance (3.4.7.2) of the system or on other interconnected systems.
- Verify that the installed system is genuine and has not been tampered with (3.4.7.3). Such evidence could include chain of custody, intact tamper-evident packaging, and original manufacturer's mark (e.g., hologram).
- Verify that patches and upgrades, if any, have a positive or neutral effect on assurance (3.4.7.4) of the system or on other systems at the site.
- Verify that tests relevant to assurance achieved satisfactory results (3.4.7.4).
- Verify that staff training is adequate (3.4.7.4).
- Analyze anomalies in transition information (3.4.7.5) to determine that they have acceptably low risk.

3.4.8 Validation

The Validation process (ISO/IEC 15288:2008, Section 6.4.9) confirms that the system complies with stakeholder requirements when used in its intended environment, based on objective evidence. Validation is performed throughout the development and operation of the system to ensure that the 'right' system is being built, and is often accomplished through operational testing and evaluation. See also the Verification section, since much of the objective evidence generated by verification feeds into validation. Assurance needs must be validated, just as other needs must be validated, throughout the life cycle.

3.4.8.1 Create validation strategy and prepare validation plan

The validation strategy and plan must be enhanced to address assurance issues. The validation strategy and plan would often be iteratively developed in parallel with the system requirements. Validation should occur throughout the life cycle, not just once after verification "completes," because problems are less expensive and faster to fix when discovered earlier. There may be additional stakeholders who are specifically concerned about system assurance; see the Stakeholder Requirements section. There may be a need to modify the scope of relevant teams (e.g., Integrated Product Teams) to address assurance issues, and those teams will require appropriate representation.

Necessary representation may be difficult to obtain in a timely manner, since assurance skill sets are a limited resource. The strategy and plan will need to address this constraint.

Determine who is responsible for validating the impact of system assurance from COTS elements and/or suppliers of concern, and how this validation will be performed. When the programme involves SoS and/or FoS considerations, include in the assurance strategy distributed testing and other appropriate planning for validating the security of the architecture and SoS/FoS interfaces.

The strategy and plan must ensure that the validation process (e.g., by performing a temporary installation of the system) will not interfere with the existing environment, including the assurance of that environment. For example, if the system being validated receives an asset such as private keys, passwords, etc., these assets must be protected and/or destroyed appropriately. (See also Section 3.1, Agreement Process, and Section 3.4.3, Architectural Design, for a discussion of alternatives during the acquisition process.)

3.4.8.2 Ensure that operators and facilities are ready for validation

Ensure that there is appropriate access to relevant information, including physical and logical access, for the validation process. This can especially be a problem where assurance information (such as threats and risks) is highly confidential.

Necessary representation (including operators) may be difficult to obtain in a timely manner because skilled personnel may be a limited resource. Any system has assumptions about its environment, and these assumptions should be validated and revalidated (as the environment changes) so assurance is maintained. Ensure that the enterprise environment is prepared for such validation. Special arrangements may need to be made for testing assurance assumptions, if they occur in the production environment, as this testing may otherwise appear to be an attack on the enterprise.

3.4.8.3 Conduct validation

Conduct validation, according to the validation strategy and plan. When conducting validation, include assurance considerations. For example, review user needs as follows:

- Who are the potential attackers, and how important is subversion of this system to them? Who would the users not trust to control this system?
- What would the attackers' goals be?
- What are the services and their importance/prioritization/criticality to the user? What is the significance of service failure if it is caused by an attack?
- What new attacks and/or previously unanticipated attacks have been identified (including information on similar systems, if any)?
- What impact will new usage modes and models have on system assurance?
- What anomalous behavior has been observed, and does it suggest an impact on system assurance?
- How long can users live without that service, or specific parts of the service?
- How much degradation in service, and which service, can the user live with?
- What is the impact of the systems supporting infrastructure on use and logistics?
- Which reused elements (including COTS) or suppliers raise concerns?

Validate that the system is “secure by default” and “secure in deployment” in its expected environment(s). As changes occur (e.g., in usage modes and environment), ensure that the system and its assurance is revalidated.

The system life cycle and operational environment are dynamic, many systems are complex, and many attackers are intelligent and adaptive. This may result in many “unknown unknowns” that impact system assurance. Thus, “unknown unknowns” must be identified and addressed throughout the system life cycle. Keep track of any new attacks coming in on honeynets, etc. Ensure that networking is being done with System Administrators to see if any anomalous traffic is noticeable that might affect the system. When a previous “unknown unknown” is identified, its assurance impacts must be reassessed and validated, consistent with the resources available and risks to the system.

In some cases modeling and simulation may be useful for validating assurance. This may be accomplished via techniques such as system mock-ups/screenshots and animation of formal models.

3.4.8.4 Isolate the part of the system giving rise to non-conformance

Isolate not only the specific requirement/design/implementation issue that is the root cause of the non-conformance (and thus needs correction), but also identify why and how that non-conformance occurred to facilitate corrective action to prevent its recurrence. Do this to (1) detect similar problems so they can also be detected and corrected, and/or (2) develop preventive measures where possible to prevent its reoccurrence (e.g., through process improvement, or a change in materials, tools, or technology).

When an assurance-based non-conformance is identified, a revalidation process should be used to determine the priority of the service, based on user needs, and determine how to feed that information into the life cycle (e.g., through element change, isolation, and so on). This should include a determination of the impact on the system assurance case with regard to the non-conformance.

Isolating a part of the system for an assurance non-conformance may be difficult, because assurance is an emergent property. An emergent property is a property of the entire system working together, not solely of one particular element. Often assurance issues do not have a single cause, but instead are caused by a combination of factors and system elements, and may involve a chain of many elements. The visible effect may be far removed from the initial causes. The assurance case may help guide identifying the factors involved and what to repair. In some cases it may be necessary to change multiple elements to establish or re-establish assurance.

3.4.8.5 Analyze, record, and report validation data and make it available

Document and deliver the validation data as objective evidence supporting the assurance argument. The set of evidence, once collected and correlated, should support the broader assurance argument(s) and system assurance claims. This evidence should be considered part of the system, and as such, must have commensurate protection, e.g., from tampering, misuse, and/or hijacking. Some assurance information may need to be confidential and specially protected, since it provides information on the system’s weaknesses to attackers.

3.4.8.6 Building the assurance case

During validation, an argument supporting the claim that the system possesses the properties required to meet user security requirements should be developed. Evidence to support this argument should be collected and/or referenced, as well. Update the assurance case to complete the evidence needed, including:

- Data that support the argument that the system meets the stakeholders’ security requirements. (3.4.8).

- Evidence that revalidation has been performed successfully after encountering non-conformance, or when there have been changes in the environment, or in the use of the system (3.4.8.3, 3.4.8.4).

3.4.9 Operation (and Training)

The point at which a system transitions to operation is critical for system assurance success. Often, the engineering team members with closest knowledge of the system and its inner workings are migrating away from the programme (ISO/IEC 15288:2008 section 6.4.10). New team members, many of whom will be held accountable for the correct functioning of the system during operation, are just entering training programmes and learning how to operate and use the system. Thus while ISO/IEC15288:2008 section 6.4.10 is titled Operation, we are including “and Training” to emphasize the importance of training. This emphasis will normally require two forms of adjustment: (1) adjusting the mission process to successfully use the system and (2) modifying the system itself, perhaps despite its implementation, to make it usable by the mission. Modifications may encompass the addition of services, updates, upgrades, changes in configuration, interfaces, etc. The implications of new team members, for example, will have to be recorded and the impact on system assurance considered. All the work to this point will either be fulfilled in assured operation, or compromised through incorrect usage.

3.4.9.1 Training users

The implementation of risk mitigations and assurance tools does not eliminate the need for training. Testers may not understand the results of a tool or how to apply those results without first understanding the kinds of vulnerabilities the tool is addressing. A training plan should be included as part of the strategy in support of systems life cycle support.

Operating a system in an assured manner requires users who have been properly vetted and trained. The system risk assessment should have identified the potential impact of risks from inadequately trained or untrustworthy users.

User training must include information on how to establish and maintain system assurance. The scope of training shall encompass information about the system and its relationship to the operational environment, so that users can maintain system assurance and leverage existing services to do so (e.g., security services, firewalls, and so on). The training shall include both positive aspects (what to do) and negative aspects (what not to do) and why, including avoiding or countering social engineering attacks. Training should include information on what the system is expected to do in typical usage circumstances, in order for users to identify when the system does something unexpected (because this may indicate a potential security breach). Training should include how to handle important non-routine and emergent conditions so users do not compromise system assurance. Training should include how users should report or act upon such non-routine or emergent conditions.

To operate the system, these vetted and trained users must be granted the privileges necessary to perform their roles. This is typically done through some sort of I&A system, then tying those identities to authorizations, per the associated policies (system, security, and enterprise policies). For system assurance to be achieved, some roles may need to be separated, and/or in some cases two-person control may be required.

3.4.9.2 Monitor the system

While monitoring the system, the objective is to ensure that the system is operating securely, as intended. The requirements for monitoring will be determined by the system risk assessment, and may include monitoring for intrusion and monitoring for deviations from expected operation or configuration that affects system assurance. For example, if a system is not supposed to be connected to the general-purpose Internet, passive monitoring for general-purpose network addresses and/or periodic attempts to access the Internet can be used to ensure that this is always the case. Monitoring can employ different tools and techniques which are available (e.g., intrusion detection tools, network mapping tools (to detect unexpected changes), risk assessment, and audit log correlation tools).

3.4.9.3 Modifying the system

For the duration of system operations, the system may need to be modified (e.g., because of changes in need or because of system problems, such as vulnerabilities). Such modifications may be deployed through system upgrades, updates, or changes in fielded configurations. Regardless, a risk assessment or security impact analysis of the changes should be performed in order to ensure system assurance can be maintained.

Because timely modifications to the system may sometimes be impractical, this can mean continued and perhaps unacceptable exposure. For example, a shipboard system might not be modified until the ship returns to port, or it might be performing other critical operations that cannot be interrupted. Deriving several alternatives for handling such an exposure may be required, including working around a problem, or informing the user how the remaining exposure may impact their mission. For all these alternatives, perform an analysis of the impact on system assurance.

System assurance is not static, because usually neither the system nor its mission is static. Information on system modifications and how they impact the system’s assurance, must cycle back into the appropriate engineering team (e.g., development, maintenance, etc.). The arguments and evidence that justify system assurance deteriorate over time unless maintained during operational adjustments.

3.4.9.4 Building the assurance case

Update the assurance case to complete the evidence needed to justify the assurance claims and arguments:

- For all aspects of the assurance case that require training of the operations and sustainment staff, collect, maintain, and audit training records and other documentation to justify that appropriate training is occurring periodically.
- Perform regular audits of operation records, to verify that there is no evidence that the system's assurance has been unknowingly subverted.
- Periodically verify that environment assumptions necessary for the assurance case are still valid.

3.4.10 Maintenance

The maintenance process (ISO/IEC 15288:2008, Section 6.4.11) sustains the capability of a system to provide its service. It monitors system operations and records problems, conducts analysis, acts on recorded problems and analysis results to take corrective/adaptive/preventive actions, and confirms restored capability. To this end, the process involves logistics, system interfaces to field support equipment and other systems, system fault detection and isolation, facilities assessment, computer resources life cycle support, and so on. The maintenance process should sustain, and in some cases improve, system assurance.

3.4.10.1 Maintenance strategy

Prepare and document a maintenance strategy, including schedules and resources required in order to perform corrective and preventive maintenance. This strategy can be part of the updated project plan or a separate document to be included as part of the maintenance process input package. Responsibilities for assurance should be purposefully and appropriately assigned; typically this will be across multiple individuals and multiple disciplines.

The maintenance or system support strategy should be generated early during planning, so that maintenance constraints can be provided as inputs to the requirements process, and to accommodate sustainment issues associated with the intended operational environment and/or interfacing systems. It should provide part of the framework upon which systems engineering and logistics planning are developed. This strategy should be refined throughout the life cycle as more detailed information becomes available.

The maintenance strategy defines the schedules and resources required to perform corrective and preventive maintenance in conformance with availability requirements. It should include:

1. The corrective and preventive maintenance strategy to sustain service in the operational environment in order to achieve customer satisfaction. This should update documentation as necessary. Corrective maintenance may

be identified through modeling and simulation performed during a repair level analysis. (See also Section 3.4, Technical Processes.)

2. The scheduled preventive maintenance actions that reduce the likelihood of system failure without undue loss of services (e.g., suspension, restriction, or reduction of the services provided (to ease assurance)).
3. The number and type of replacement system elements to be stored, their storage locations and conditions, their anticipated replacement rate, their storage life, and renewal frequency. For assurance, identify how they will be protected while they are stored, and how retrieval of them is managed, controlled, and logged.
4. The skill and personnel levels required to effect repairs and replacements. For assurance, determine how to ascertain their trustworthiness, how to ensure that only authorized individuals can perform repairs/replacements, and include a process to audit deviations. Explain how they will receive sufficient training.
5. Configuration management maintenance approach; see Section 3.3.6, Configuration Management.

The strategy should define processes for developing, verifying, and delivering changes that are protected from unauthorized and undetected changes, and ensure that recipients can verify that changes they receive are authorized.

3.4.10.2 Define the constraints on system requirements

Per ISO/IEC 15288:2008, constraints on system requirements are unavoidable consequences of the maintenance strategy.

In some cases it may be necessary for a fielded system to authenticate an update system or vice versa. If the system can be updated in the field, it may be necessary to include a public key in the deployed system that can be used to cryptographically verify that updates are from a trusted source. In addition, it may be necessary to have a process for providing a private key to deployed systems, so that their identity can be verified. There may be a need for some sort of “call home” update system, or conversely, it may be necessary to disable such a system. Evaluate this function and its impact and need for the service on the system. Physical security measures (e.g., holographic images embedded in hardware elements) may be necessary to prevent use of counterfeit parts.

If maintenance must occur with little to no service interruption, identify what maintenance must occur under these conditions, and how much service interruption is acceptable. Often this is done with alternative/backup systems, processes, or pathways; these must be examined to ensure that their implementation and use do not introduce vulnerabilities.

There may be requirements that the system support special maintenance roles, which may be limited in their actions. It may be necessary that the system not be able to perform certain actions because it is physically exposed to untrustworthy parties. An “undo update” function may be important to restore functionality, should an update cause critical functionality to break.

3.4.10.3 Obtain the enabling systems, system elements, and services used during maintenance

As part of assurance maintenance, consider including products and services that monitor/maintain assurance of the elements (e.g., intrusion detection, configuration management audits, audit log examination, risk assessment).

Ensure that any infrastructure used to distribute software or data updates can withstand attack, and is able to scale to the number of users.

If specialized equipment or tools are required for retirement of elements during maintenance (e.g., incinerators for storage devices), obtain them.

3.4.10.4 Implement problem reporting and incident recording

There may be a need for a special problem-reporting procedure for vulnerabilities, to prevent such information from reaching potential attackers (e.g., through cryptography), and to ensure that critical vulnerabilities can be considered immediately.

Some incident recording may need to be specially protected from arbitrary access, since it may reveal significant vulnerabilities or weaknesses in a system.

In some instances, a community (e.g., banking, government, etc.) may require notification of certain types of vulnerabilities or weaknesses (depending on their impact), expected corrective actions, and associated timelines, as a cost of doing business.

Problem reports must be prioritized based on impact; those with a significant impact on assurance should have high priority. Techniques such as using vulnerability trees may aid in this analysis.

3.4.10.5 Implement failure identification

Correction procedures should consider if the reported problem is evidence of an unknown vulnerability and/or of an attack on the system. It may also be evidence of non-compliance with a security policy or controls (e.g., configuration management controls).

Identify the failure and its associated fault. Determine the fault’s impact (i.e., through impact analysis and/or risk assessment), including a determination of whether it is an exploitable vulnerability. In some cases it may be difficult to determine if it is exploitable; if this is uncertain, it should be treated as an exploitable vulnerability.

When a fault is detected, an assessment of the fault should be performed to identify whether it poses a threat and to diagnose its origin. Where appropriate, forensic analysis of the failure should be performed to identify the root cause and to take appropriate action.

If a new failure is identified, correction procedures should involve looking for similar or linked constructs to also repair. This is because a type of fault found in one place may have occurred in other places as well and may have also previously escaped detection. Root cause analysis of such new vulnerabilities is one approach for performing this search. It is also imperative that when exploitation is identified, prompt action is taken to mitigate the consequences among all affected areas.

3.4.10.6 Implement correction procedures

Develop a plan to correct the product, and if necessary, communicate it with relevant parties. In some cases, some functionality may need to be removed to maintain assurance; this situation in particular may require such communication. In some cases, there may be conflicting goals (e.g., functionality vs. assurance, cost vs. speed of change); see Section 3.3.5, Risk Management, for how to adjudicate such issues.

Verify, using test and evaluation, for example, that the corrective action is effective. The correction procedures should not introduce new vulnerabilities or weaknesses, and should avoid or minimize either reducing or interrupting service. The corrective actions should also update any relevant documentation, including the justification and supporting evidence of assurance.

3.4.10.7 Confirm logistics actions satisfy replenishment levels

Based on the maintenance strategy, ensure that spare elements will be protected while they are stored, and that retrieval of them is managed, controlled, and logged. This could include fail-over systems residing on a network ready for operation, as well as substitute elements not yet installed.

3.4.10.8 Perform preventive maintenance

Ensure that software systems are kept up-to-date on their security patches, where practical, instead of waiting for attackers to subvert the system.

3.4.10.9 Maintain a history

- This historical information may be sensitive, and as such appropriate measures may be necessary to protect the confidentiality, integrity, accountability (including non-repudiation), availability, and auditability of the data.

3.4.10.10 Building the assurance case

Changes to the system during system maintenance may have wide-ranging (little to significant) impact on the assurance case. Since each change can involve one or more of the technical processes described in the preceding sections, the maintenance of the assurance case involves following the guidelines for the processes involved in the change. For example, a change which involves a change to code with no change in requirements or architecture design requires the use of the guidelines for the implementation, integration, verification, transition, and validation processes (3.4.4 thru 3.4.8).

During maintenance, update the assurance case to complete the evidence needed to justify the assurance claims and arguments:

- For each change, perform the activities in the applicable processes as described in this guidebook (see sections 3.4.1 - 3.4.9), including updating the assurance case as appropriate.
- Analyze the defect and enhancement requests to determine any deviation that affects the assurance case, or trends that may impact the assurance case (e.g., through threats, vulnerabilities, and security breaches).

3.4.11 Disposal

The disposal process (ISO/IEC 15288:2008, Section 6.4.12) deactivates, disassembles, and removes the system and/or system elements and/or any waste products. The process destroys, stores, or reclaims system elements (including software, hardware, and data). For assurance, procedures must be defined for the safe and secure sanitization and destruction of the system elements, along with ensuring data are destroyed or migrated safely and securely. For example, this may include the destruction of private keys embedded in cryptographic modules.

3.4.11.1 Disposal strategy

In addition to environmental concerns, some software, hardware, and/or data may need to be destroyed or retained. There may be restrictions in terms of who may handle data (e.g., handling caveats for classified information or personally identifying information), and/or there may be a need to destroy hardware, software, or data to prevent reverse engineering, reuse, or potential release. Some elements may need to be handled specially (e.g., cryptographic gear may need to be destroyed to prevent reverse engineering, or have embedded keys that must be erased using special processes). Additionally, some elements may be released for reuse or resale; in these cases, elements must be examined to ensure that software, hardware, or data that should not be released are not thereby accessible or recoverable. For example, if a computer system or storage device is resold, ensure that the storage device(s) is properly erased before release. Laws, regulations, policies, licenses, contracts, and other requirements may impose specific disposal criteria (requirements or restrictions) related to assurance.

A disposal strategy should be developed that identifies the elements (hardware, software, and/or data) to be disposed of, and the approved process for disposal. Particular focus should be placed on the disposal of more sensitive elements or information, system CPIs including classified information, elements that must not be reverse-engineered, and personally identifying information.

One strategy may be to keep a careful record of system elements to ensure that proper techniques are applied during disposal; see Section 3.3.6, Configuration Management.

The disposal strategy should:

1. Sanitize/destroy hardware, software, and data, as appropriate, in all locations it resides. This may be determined by the licensing or other legal agreements. Should some elements (e.g., storage devices and computing equipment) be resold or reused, the licensing may forbid transfer of software or certain software upgrades. This is particularly the case with enterprise-wide software license agreements. Disposal of an operating system (OS) is often tied to the computer hardware on which it was originally installed. When an enterprise donates a computer to a charity or sells it to a third party, often the OS also must be donated.
2. Sanitize/destroy all associated media as appropriate; this includes original media used to load the software or data, backup copies of software or data, and portions of the software or data that may exist in system temp or swap spaces.
3. Sanitize/destroy associated technical manuals and training materials for the hardware, software, and data, as appropriate.

Requirements for clarifying destruction and licensing policies need to be established. Policies and procedures provide standard methodologies for destruction and control the risks associated with improper disposal of software and computing equipment.

The strategy may need to be reviewed by the organisation’s legal counsel, and include a process to track and record compliance. It may be necessary to keep an inventory of serial numbers, dates of purchase, dates of destruction, and means of destruction. By providing a complete disposal audit trail, it is possible to assure that hardware, software, and/or data were not inadvertently exposed.

3.4.11.2 Destroy the system

There are many different elements that may need to be destroyed, or have software or data removed from them. For example:

- Any hardware element that should not be reverse-engineered.
- Magnetic hard drives.
- Flash memory. Note that “overwriting” to flash memory typically does not actually overwrite all the underlying information, due to wear leveling and garbage collection mechanisms built into these devices.
- Volatile memory. Volatile memory elements do not normally retain data after removal of all electrical power sources, and when reinserted into a similarly configured system, do not contain residual data. However, volatile memory may become non-volatile through unknown battery backups, and a highly resourced adversary may be able to retrieve some information.

- Nonvolatile memory elements do retain data when all power sources are discontinued. Nonvolatile memory elements include Read Only Memory (ROM), Programmable ROM (PROM), or Erasable PROM (EPROM and EEPROM). Examples of their use include BIOSes, FPGA configurations, and ASICs.

All of these elements may reside on boards, modules, and sub-assemblies. Naïve approaches to removing software and data often fail. Simply deleting files from the media, or reformatting the media, is often insufficient to completely erase data so that it cannot be recovered; often such data is easily recovered. Approaches for destroying software or data include:

1. Overwriting the data. Overwriting of data means replacing stored data on a drive with other data. There are numerous software products that rewrite media to wipe off deleted files or the entire contents of a computer drive. Many of the products have various levels of overwrite, so be sure to select the option that is compliant with your requirements. As noted above, this often fails with flash memory.
2. Degaussing. This is a procedure that reduces the magnetic field on media virtually to zero by applying a reverse-magnetizing field. Properly applied, degaussing renders any previously stored data on magnetic media difficult to read. Magnetic media can be divided into three types (I, II, and III) based on their coercivity, which defines the magnetic field necessary to reduce a magnetically saturated material's magnetization to zero. The level of magnetic media coercivity must be ascertained before executing any degaussing procedure.
3. Physical force (e.g., pounding with a sledgehammer). Bending, disfiguring, or otherwise mutilating the media may render its contents unrecoverable, but it may be ineffective for disposing data. It is not typically regarded as a best practice.
4. Incineration. This is typically the best way to destroy data (including software) or hardware.

Highly resourced adversaries may be able to recover data from overwritten, degaussed, or physically damaged devices, so processes such as incineration may be necessary in such cases. In some cases, an independent party may need to witness the destruction.

3.4.11.3 Outcome/benefits

Outcome of performing proper destruction of systems is elimination of the risk associated with access to residual hardware, software, or data on the system or its elements. This reduces the risk of revelation of system information that might enable an attacker to penetrate the system, illegal distribution of licensed software, release of sensitive system information to an unauthorized organisation or individuals, enabling reverse engineering, etc.

3.4.11.4 Building the assurance case

Update the assurance case to complete the evidence needed to justify the assurance claims and arguments:

- Verify that all elements requiring disposal are appropriately identified and destroyed in an appropriate manner. Note: For highly critical elements, this often this requires incineration.

ANNEX A

CORRESPONDENCE WITH EXISTING STANDARDS

COMMERCIAL STANDARDS–IEEE, ISO

Institute of Electrical and Electronics Engineers (IEEE) Standards

The IEEE is a leading developer of standards that underpin many of today's technologies. The standards are developed in a unique environment that builds consensus in an open process based on input from all interested parties. With nearly 1,300 standards either completed or under development, it is a central source of standardization in both traditional and emerging fields, particularly telecommunications, information technology, and power generation.

The IEEE also develops standard jointly with ISO/IEC JTC1/SC7, the subcommittee for systems and software engineering standards.

Some of the standards relevant to the system assurance process include:

- ISO/IEC/IEEE 12207-2008, Systems and software engineering — Software life cycle processes.

  This standard establishes a common framework for software life cycle processes, with well-defined terminology, that can be referenced by the software industry. It contains processes, activities, and tasks that are to be applied during the acquisition of a software product or service and during the supply, development, operation, maintenance and disposal of software products.
- ISO/IEC/IEEE 15288-2008, Systems and software engineering — System life cycle processes.

  This standard establishes a common framework for describing the life cycle of systems created by humans. It defines a set of processes and associated terminology. These processes can be applied at any level in the hierarchy of a system's structure. Selected sets of these processes can be applied throughout the life cycle for managing and performing the stages of a system's life cycle.
- ISO/IEC/IEEE 14764-2006, Software Engineering — Software Life Cycle Processes — Maintenance.

  This standard defines the process for performing the maintenance of software.
- ISO/IEC/IEEE 16085-2006, Systems and software engineering — Life cycle processes — Risk management.

  This standard describes a process for the management of risk during systems or software acquisition, supply, development, operations, and maintenance.
- IEEE Std 610.12-1990, Standard Glossary of Software Engineering Terminology

  This standard contains definitions for more than 1,000 terms, establishing the basic vocabulary of software engineering. [This standard is being replaced by

ISO/IEC IEEE 24765, Systems and software engineering — Vocabulary, and by the SEVOCAB terminology database

- IEEE Std 1233-1998, Guide for Developing of System Requirements Specifications

  This standard provides guidance for the development of a set of requirements that, when realized, will satisfy an expressed need.
- IEEE Std 1058-1998, Standard for Software Project Management Plans

  This standard specifies the format and contents of software project management plans.
- IEEE Std 1074-2006, IEEE Standard for Developing a Software Project Life Cycle Process

  This standard provides a process for creating a software project life cycle process (SPLCP). It is primarily directed at the process architect for a given software project. It is the function of the process architect to develop the SPLCP.
- IEEE Std 730.1-1998, Guide for Software Quality Assurance Plans

  The purpose of this guide is to identify approaches to good Software Quality Assurance practices in support of IEEE Std 730. (The standard establishes a required format and a set of minimum contents for Software Quality Assurance Plans. The description of each of the required elements is sparse and thus provides a template for the development of further standards, each expanding on a specific section of this document.)
- IEEE Std 1008-1987 (Reaff 1993), Standard for Software Unit Testing

  The standard describes a testing process composed of a hierarchy of phases, activities, and tasks. Further, it defines a minimum set of tasks for each activity.
- IEEE Std 1063-2001, Standard for Software User Documentation

  This standard provides minimum requirements for the structure and information content of user documentation.

ISO Standards

- ISO/IEC 90003-2004, Software and Systems Engineering – Guidelines for the Application of ISO 9001:2000 to Computer Software

  This standard provides guidance to organisations in the application of ISO 9001:2000 to the acquisition, supply, development, operations, and maintenance of computer software.
- ISO 13335 - IT security management

  ISO 13335 (which started life as a Technical Report (TR) before becoming a full ISO standard) comprises a set of guidelines for the management of IT security, focusing primarily on technical security control measures:

  - ISO 13335-1:2004 Information technology – Security techniques – Management of information and communications technology security – Part

1: Concepts and models for information and communications technology security management explains the concepts and models for information and communications technology security management. (ISO/IEC TR 13335 parts 1 and 2 were combined into the revised ISO/IEC 13335-1: 2004. The original TR13335-2:1997 “Guidelines for the management of IT security - Part 2: Managing and planning IT security” was cancelled.)

- ISO 13335-2, when published, is expected to cancel and replace ISO/IEC TR 13335-3:1998 and ISO/IEC TR 13335-4:2000.
- ISO TR 13335-3:1998 Information technology – Guidelines for the Management of IT Security – Part 3: Techniques for the management of IT Security covers techniques for the management of IT security. This standard is currently under revision and will be inserted into ISO 27005
- ISO TR 13335-4:2000 covers the selection of safeguards (meaning technical security controls). This standard is also currently under revision and will be inserted into ISO 27005
- ISO TR 13335-5:2001 provides management guidance on network security. This standard is currently under revision, being merged into ISO/IEC 18028-1. ISO/IEC 18028-1 will eventually cancel and replace ISO/IEC TR 13335-5:2001.ISO 15408 - Common Criteria

• ISO 15408:1999 describes the Common Criteria for Information Technology Security Evaluation. Products that are evaluated against the Common Criteria have a defined level of assurance for their information security capabilities that is recognized in most of the world. Unfortunately, the evaluation process is quite costly and slow, and is therefore not very widely used apart from the government and defense markets.

• ISO 19770 - Software Asset Management

ISO/IEC 19770-1:2006 promotes the implementation of an integrated set of software asset management processes, using good practices for efficient software management. Contents include Scope, terms and definitions, Field of application, Conformance, Intended usage, Agreement compliance, General Software Asset Management processes, Control environment for Software Asset Management, Planning and implementation, Inventory processes, Verification and compliance processes, Operations management processes and interfaces, Life cycle process interfaces.

• ISO 21827 - Systems Security Engineering Capability Maturity Model

Like other Capability Maturity Models (CMMs), the Systems Security Engineering (SSE) CMM defines the essential characteristics of SSE processes, emphasizing those which indicate process maturity. The model covers the entire systems development life cycle from concept definition to decommissioning. It applies to those developing or integrating secure products/systems, and those providing specialist security services such as security engineering. It was published as ISO 21827 in 2002.

- ISO has reserved the ISO/IEC 27000-series numbering for a range of information security management standards in similar fashion to the very successful ISO 9000-series quality assurance standards.

The following ISO 27000-series standards are either published or planned:

- ISO 27000 will contain the vocabulary and definitions, i.e., terminology for all of these information security management standards
- ISO 27001 is the Information Security Management System (ISMS) requirements standard (specification) against which organisations are formally certified compliant. In October 2005, British Standard BS 7799 part 2 was adopted by ISO, rebadged and released as the new international information security standard ISO/IEC 27001:2005. ISO 27001 is the formal standard against which organisations may seek independent certification of their ISMS (meaning their frameworks to design, implement, manage, maintain, and enforce information security processes and controls systematically and consistently throughout the organisations).
- ISO 27002 will be the new name for the standard currently known as ISO 17799 and formerly known as BS 7799 part 1. This is the code of practice for information security management describing a comprehensive set of information security control objectives and a menu of best-practice security controls.
- ISO 27003 will be an implementation guide.
- ISO 27004 will be an information security management measurement standard to help measure the effectiveness of ISMS implementations.
- ISO 27005 will be an information security risk management standard (will replace the recently issued BS 7799 Part 3).
- ISO 27006 will be a guide to the certification/registration process for accredited ISMS certification/registration bodies.

GLOSSARY

Following is a selected list of SA terms. Where possible, the authors have used standard definitions; in some cases, however, it was necessary to modify the standard definitions to more accurately convey the meaning within the SA context.

Abuse: Intentional or reckless misuse (i.e., use in an unintended way), alteration, disruption, or destruction.

Accountability: The security goal that generates the requirement for actions of an entity to be traced uniquely to that entity. This supports non-repudiation, deterrence, fault isolation, intrusion detection and prevention, and after-action recovery and legal action.

Anomaly: Any condition in the software's operation or documentation that departs from the expected. This expectation can come from the software's documentation (e.g., requirements specification, design documentation, user documentation), its source code or other development artifact(s), or from someone's perception or experience with the software (e.g., an observation of how the software operated previously). (IEEE Std 1012-1986, IEEE Std 1044-1993)

Anti-Tampering (AT): The Systems Engineering (SE) activities intended to prevent and/or delay exploitation of critical technologies. These activities involve the entire life cycle of systems acquisition including research, design, development, testing, implementation, and validation of anti-tamper measures. Properly employed, anti-tamper measures add longevity to a critical technology by deterring efforts to reverse-engineer, exploit, or develop countermeasures against a system or system component (aka element).

Architecture: The fundamental organisation of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution. (ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems) If a system is being developed to fit inside a larger enterprise architecture, the environment includes the enterprise architecture and the system's architecture includes the interfaces to that enterprise architecture.

Argument: A justification that a given claim (or sub-claim) is true or false.

Assurance: See system assurance.

Assurance Case: The set of *claims* of critical system assurance properties (requirements of the system), *arguments* that justify the claims (including assumptions and context), and *evidence* supporting the arguments.

Availability: The degree to which the services of a system or component (aka element) are operational and accessible when needed by their intended/authorized users. In the context of security, availability pertains to authorized services/actions only, and the need for availability generates the requirement that the system or component is able to resist or withstand attempts at unauthorized deletion or denial of service, regardless of whether those attempts are intentional or accidental. (*IEEE Std 610.12-1990*)

Blind Buy: An acquisition in which the identity of the acquirer and/or user(s) is intentionally concealed from the supplier.

Claim: The critical system requirements for assurance, including the maximum level of uncertainty permitted for it.

Component: One of the parts or elements that make up a system. A component may be hardware or software and may be divisible into smaller components. Note: With regard to software, the terms module, component, and unit are often used interchangeably, or defined to be sub-elements of one another. Used in this document as a synonym for “element.”

Compromise: A violation of the security policy of the system, or an incident in which any of the security properties of the system are violated or its dependability is intentionally degraded.

Confidentiality: The property that sensitive information is not disclosed to unauthorized individuals, entities, or processes.

Countermeasure: An action, device, procedure, technique, or other measure that reduces the vulnerability of a component or system.

Critical Component: See critical element.

Critical Element: An element of a critical system whose unintentional or intentional subversion/failure (possibly in combination with others) could cause the loss of essential system functionality or mission failure.

Critical Programme Information: Elements or components of an RDA programme that if compromised, could cause significant degradation in mission effectiveness, shorten the expected combat-effective life of the system, reduce technological advantage, significantly alter programme direction, or enable an adversary to defeat, counter, copy, or reverse engineer the technology or capability.

Criticality: A relative measure of the consequences of a failure mode and its frequency of occurrence.

Critical System: A system determined to have such vital importance that its engineering, production, evaluation, sustainment, assurance, acquisition, and subsequent operation must be governed by focused system assurance activities.

Defense-in-Depth: A strategy that combines people, technology, and operations capabilities to establish protective barriers that span different layers and dimensions of a system in its operational environment in order to isolate that system from potential sources of attack.

Denial of Service: An action or series of actions that (1) prevents access to a software system by its intended/authorized users; (2) causes the delay of its time-critical operations; or (3) prevents any part of the system from functioning.

Dependability: The degree to which the system is operable and capable of performing functionality or of delivering a service that can justifiably be relied upon (i.e., trusted) to be correct. To achieve dependability, the system must be able to avoid service failures that are more frequent or severe, or longer in duration, than is acceptable to users. Dependability may be viewed according to different, but complementary, properties (or “instances of dependability”) required for a system to be considered dependable:

- Availability
- Integrity
- Reliability
- Safety
- Maintainability
- Security

Two other properties are closely related to dependability:

- Survivability
- Trustworthiness

For systems, availability and reliability emphasize the avoidance of failures, safety emphasizes the avoidance of a specific class of failures (catastrophic failures), and security emphasizes the prevention of exploitable development faults (intentional and unintentional) and malicious external faults.

EIA 632, Processes for Engineering a System: Provides an integrated set of fundamental processes to aid a developer in engineering or re-engineering a system.

Element: See system element.

Enabling System: A system that complements a system-of-interest during its life cycle stages but does not necessarily contribute directly to its function during operation. (ISO/IEC 15288:2008)

Engineering-in-Depth (EiD): The employment of systems engineering processes to meet system assurance requirements, usually involving issues of incorporation of elements with different degrees of uncertainty about their relevant behavior or properties and defense-in-depth.

Enterprise: One or more organisations sharing a definite mission, goals, and objectives to offer an output such as a product or service (ISO 15704, Industrial automation systems — Requirements for enterprise-reference architectures and methodologies).

Enterprise Architecture (EA): The architecture is a conceptual blueprint that defines the structure and operation of an organisation. The intent of an enterprise architecture is to determine how an organisation can most effectively achieve its current and future objectives. EA is also variously defined as “…understanding all of the different elements that go to make up the enterprise and how those elements interrelate.”[2]

Evidence: Information that demonstrably justifies the arguments in an assurance case.

IEEE 1220: Standard for managing systems engineering.

[2] E.g., see: http://www.enterprise-architecture.info/EA_Methods.htm.

Information Assurance (IA): Measures that protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality, and nonrepudiation.

Integrity: The quality of a system or component that reflects its logical correctness and reliability, completeness, and consistency. In security terms, integrity generates the requirement for the system or component to be protected against either intentional or accidental attempts to (1) alter, modify, or destroy it in an improper or unauthorized manner, or (2) prevent it from performing its intended function(s) in an unimpaired manner, free from improper or unauthorized manipulation.

ISO/IEC 15288:2008, Systems and software engineering – System life cycle processes : Establish a common framework for describing the life cycle of systems.

Least Privilege: Principle requiring that each subject be granted the most restrictive set of privileges needed for the performance of that subject's authorized tasks. Application of this principle limits the damage that can result from accident, error, or unauthorized use of a component or system.

Maintainability: The ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment (IEEE 90).

Malicious Code: Software or firmware intended to perform an unauthorized process that will have adverse impact on the dependability of a component or system.

Mission Assurance: Identify and mitigate design, production, test, and field support deficiencies that could affect mission success.

Non-Repudiation: In law, non-repudiation implies one's intention to fulfill his/her obligations to a contract. It also implies that one party of a transaction cannot deny having received a transaction nor can the other party deny having sent a transaction. Electronic commerce uses technology such as digital signatures and encryption to establish authenticity and non-repudiation.

Non-Developmental Items: The statutory definition of non-developmental item includes COTS and GOTS.

Programme Protection Plan (PPP): A risk-based, comprehensive, living plan to safeguard CPI that is associated with a RDA programme. The PPP is used to develop tailored protection guidance for dissemination and implementation throughout the programme for which it is created. The layering and integration of the selected protection requirements documented in a PPP provide for the integration and synchronization of CPI protection activities**.**

Reliability: The ability of a system and its parts to perform its mission without failure, degradation, or demand on the support system.

Requirement: A statement that identifies a product or process operational, functional, or design characteristic or constraint, which is unambiguous, testable, or measurable, and necessary for product or process acceptability (by consumers or internal quality assurance guidelines). (IEEE 1220)

Risk: The possibility or likelihood that a particular threat will adversely impact a system by exploiting a particular vulnerability.

Risk: A measure of the inability to achieve programme objectives within defined cost and schedule constraints. Risk is associated with all aspects of the programme, e.g., threat, technology, design processes, or Work Breakdown Structure (WBS) elements. It has two components: the probability of failing to achieve a particular outcome, and the consequences of failing to achieve that outcome.

Risk Management: All plans and actions taken to identify, assess, mitigate, and continuously track, control, and document programme risks.

Robustness: The degree to which a component or system can function correctly in the presence of invalid inputs or stressful environmental conditions, including inputs or conditions that are intentionally and maliciously created (IEEE Std 610.12-1990)

Sandboxing: A method of isolating application-level components into distinct execution domains, the separation of which is enforced by software. When run in a “sandbox,” all of the “sandboxed” programme’s code and data accesses are confined to memory segments within that “sandbox.” In this way, “sandboxes” provide a greater level of isolation between executing processes than can be achieved when processes run in the same virtual address space. The most frequent use of sandboxing to isolate the execution of untrusted programmes (e.g., mobile code, programmes written in potentially unsafe languages such as C) so that each programme is unable to directly access the same memory and disk segments used by other programmes, including the application’s trusted programmes. Virtual machines (VMs) are sometimes used to implement sandboxing, with each VM providing an isolated execution domain.

Secure State: The condition in which no subject can access another entity in an unauthorized manner.

Security: Protection against intentional subversion or forced failure. Security is a composite of four attributes – confidentiality, integrity, availability, and accountability -- plus aspects of a fifth, usability, all of which have the related issue of their assurance.

Software: Software for these purposes shall be taken to mean the programmes, routines, and symbolic languages that control the functioning of the hardware and direct its operation, including but not limited to the genre of items called software, firmware, microcode, source code, object code, machine code, machine language, etc.

Software Assurance (SwA): (1) concerning uncertainty related to properties or behavior of software; (2) grounds for justified confidence in software’s correctness of specification or meeting of specification or requirement.

Software Intensive: A software-intensive system is any system in which software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole. Ref: IEEE STD 1471-2000.

Supply Chain: The set of organisations, people, activities, information, and resources for creating and moving a product or service, including its elements, from suppliers through to their customers.

System: A combination of interacting elements organized to achieve one or more stated purposes. (ISO/IEC 15288:2008)

System Assurance: The justified confidence that the system functions as intended and is free of exploitable vulnerabilities, either intentionally or unintentionally designed or inserted as part of the system at any time during the life cycle.

System Assurance Activities: A planned, systematic set of multi-disciplinary activities to achieve the acceptable measures of system assurance and manage the risk of exploitable vulnerabilities.

System Element: A member of a set of elements that constitutes a system. NOTE: A system element is a discrete part of a system that can be implemented to fulfill specified requirements. A system element can be hardware, software, data, humans, processes (e.g., processes for providing service to users), procedures (e.g., operator instructions), facilities, materials, and naturally occurring entities (e.g., water, organisms, minerals), or any combination. (ISO/IEC 15288:2008). Note that a system element can be composed of other system elements.

System-of-Interest: The system whose life cycle is under consideration in the context of this International Standard (ISO/IEC 15288:2008)

System of Systems (SoS) and **Family of Systems (FoS):** These terms are substantially defined elsewhere. For the purpose of this paper, SoS and FoS require additional diligence in system assurance because of the potential complexity involved. Aspects of SoS/FoS can sometimes improve overall system assurance (e.g., multiple systems provided for recoverability in a FoS), while some aspects can also make assurance more difficult (e.g., complex interoperability among distributed SoS elements).

Threat Agent: Individual or organisation that has an intent to cause harm

Verification and Validation (V&V): The process of confirming, by examination and provision of objective evidence, that:

- Each step in the process of building or modifying the software yields the right products (verification). Verification asks and answers the question, “Was the software built right?” (i.e., correctness);
- The software being developed or modified will satisfy its particular requirements (functional and nonfunctional) for its specific intended use (validation). Validation asks and answers the question “Was the right software built?” (i.e., suitability).

Vulnerability: A development fault or other weakness in a deployed system or exploited with malicious intent by a threat with the objective of subverting or incapacitating (violation of integrity, achieved by modification, corruption, destruction, or security policy violation)—often to gain access to the information it handles—or to force the software to fail (denial of service). Vulnerabilities can originate from flaws on the system’s design, defects in its implementation, or problems in its operation.

Weakness: A flaw, defect, or anomaly in a system that has the potential of being exploited as a vulnerability when the software is operational. A weakness may originate from a flaw in the system’s security requirements or design, a defect in its implementation, or an inadequacy in its operational and security procedures and controls. The distinction between “weakness” and “vulnerability” originated with the MITRE Corporation Common Weaknesses and Exposures (CWE) project (http://cve.mitre.org/cwe/about/index.html).

ABBREVIATIONS AND ACRONYMS

ACE	Application Consulting and Engineering
ALARP	as low as reasonably practicable
ASCAD	Adelard Safety Case Development
ASIC	application-specific integrated circuit
ASR	Alternative Systems Review
AT	anti-tamper
BIOS	basic input/output system
C&A	Certification and Accreditation
CAPEC	Common Attack Pattern Enumeration and
CCEVS	Common Criteria Evaluation and Validation
CDR	Critical Design Review
CI	configuration item
CI	counter intelligence
CIO	Chief Information Officer
CND	Computer Network Defense
COCOMO	Cost Control Model
CoI	Community of Interest
CoP	Community of Practice
COSECMO	Cost Security Model
COTS	commercial off-the-shelf
CM	configuration management
CMMI	Capability Maturity Model Integration
CTMM	Calculative Threat Modeling Methodology
CPI	critical programme information
CR	Change Request
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
DAG	Defense Acquisition Guidebook
DOTMLPF	Doctrine Organisation Training Materiel Leadership Personnel and Facility
E2E	end to end
ECR	Engineering Change Request
EiD	Engineering in Depth
EM	Electromagnetic
EOIR	electro-optical and infrared
EPROM	Erasable PROM

FDCC	Federal Desktop Core Configuration
FMEA	Failure Modes and Effects Analysis
FMECA	Failure Mode, Effects, and Criticality Analysis
FMICA	Failure Modes and Impacts Criticality Analysis
FoS	family of systems
FOUO	For Official Use Only
FPGA	field-programmable gate array
FRP	full-rate production
FTA	fault tree analysis
GIF	graphic interchange format
GOTS	government off-the-shelf
GSN	Goal Structuring Notation
HW	Hardware
IA	information assurance
I&A	identification and authentication
IC	integrated circuit
IC	intelligence community
ICD	Initial Capabilities Document
ICSA	International Society for Computers and Their Applications
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IM	information management
IM	instant messaging
IMS	Integrated Master Schedule
INCOSE	International Counsel on Systems Engineering
INFOCON	Information Operations Condition
IP	Internet protocol
ISO/IEC	International Organisation for Standardization / International Electrotechnical Commission
IT	information technology
ITAR	International Trafficking in Arms Regulation
ITR	Initial Technical Review
LoA	Level of Assurance

MOP	measures of performance
NDI	non-developmental item
OS	operating system
OT&E	Operational Test & Evaluation
OTS	off-the-shelf
OWASP	Open Web Application Security Project
PDR	Preliminary Design Review
PLD	programmable logic device
PM	programme manager; programme management
PMBOK	Project Management Book of Knowledge
PPP	Programme Protection Plan
PROM	Programmable ROM
RCM	Reliability-Centered Maintenance
R&D	requirements and development
RF	radio frequency
RFP	request for proposal
ROM	Read-Only Memory
SDP	Software Development Plan
SA	system assurance
SE	systems engineering; systems engineer
SEP	Systems Engineering Plan
SETR	Systems Engineering Technical Reviews
SFR	System Functional Review
SoS	system of systems
SOUP	system of unknown pedigree
SRR	System Requirements Reviews
SVR	System Verification Review
SW	Software
SwA	software assurance
SwACBK	Software Assurance Common Body of Knowledge
TDS	Technology Development Strategy
T&E	test and evaluation
TEMP	Test and Evaluation Master Plan
TES	Test and Evaluation Strategy
TRA	Technology Readiness Assessment
TRL	Technology Readiness Level
TR	Trouble Report

TRR	Test Readiness Review
VHDL	Very High Speed Integrated Circuit (VHDIC) Hardware Description Language
V&V	verification and validation
WBS	Work Breakdown Structure

REFERENCES

Avery, James, and Jim Holmes. 2006. *Windows Developer Power Tools.* O'Reilly.

Avizienis, Algirdas, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 1(1):11–33.

Colbert, Edward, and Danni Wu, 21st International Forum on COCOMO & Software Cost Modeling. Costing Secure Systems Workshop Report.

Constantine, Larry L., and Scott W. Ambler. 2002. *The Unified Process Transition & Production Phases: Best Practices in Implementing the UP*. BMP Books.

CVE (Common Vulnerabilities and Exposures). http://cve.mitre.org/.

CWE (Common Weakness Enumeration). http://cwe.mitre.org/.

EIA 632: http://www.techstreet.com/cgi-bin/detail?product_id=1145585

EIA 649: http://www.techstreet.com/cgi-bin/detail?product_id=1167865

Equipment Disposal Policy (draft). 2007. March 26. Gannon University.

Fedchak, Elaine, Thomas McGibbon, and Robert Vienneau. 2007. Software Project Management for Software Assurance. N.P.

Hogganvik, Ida, and Ketil Stølen. 2006. *A Graphical Approach to Risk Identification, Motivated by Empirical Investigations. In 9th International Conference on Model Driven Engineering Languages and Systems* (MoDELS 2006). No. 4199 in *Lecture Notes in Computer Science,* pp. 574-588, Springer, 2006).

IEEE 1220. http://shop.ieee.org/ieeestore/Product.aspx?product_no=SS95334

IEEE P1633\AIAA R-013A. Standard for Software Reliability.

INCOSE (International Council on Systems Engineering). 2007. *Systems Engineering Handbook*. v.3. INCOSE.

INCOSE (International Council on Systems Engineering). 2005. *Guide to Systems Engineering Body of Knowledge* (G2SEBoK). http://g2sebok.incose.org/.

ISO/IEC (International Standards Organisation/International Electrotechnical Commission) 9126. Parts 1, 2, and 3. Software Engineering: Product Quality

ISO/IEC 12207:2008. Systems and Software Engineering: Software Life Cycle Processes.

ISO/IEC 14598. Parts 1–6. Information Technology: Software Product Evaluation.

ISO/IEC 15288:2008. Systems and Software Engineering: System Life Cycle Processes.

ISO/IEC 15408–1:2005. Information Technology: Security Techniques. Part 1: Introduction and general model.

ISO/IEC 27001:2008. Information Security Management System Standard.

O'Brien, Frances, and Alvin R. Park. 2003. *Software Disposal: Old Software Never*

Dies. Gartner.

OWASP (Open Web Application Security Project). 2005. *A Guide to Building Secure Web Applications and Web Services 2.0.* The Open Web Application Security Project, Black Hat Edition. http://www.owasp.org/index.php/OWASP_Guide_Project

Pigoski, T.M.; Sexton, J. 1990. Software transition: A case study. In *Software Maintenance Proceedings* (Nov 26-29): 200–204.

PMBOK (*Project Management Book of Knowledge, A Guide to the*). 2004. 3d ed. Washington, DC: Project Management Institute.

Rowland, L. Hawaii Pacific University.

Swiderski, F., and W. Snyder. 2004. *Threat Modeling*. Microsoft Press.

INTENTIONALLY BLANK

AEP-67(B)(1)