Software Project Management for Software Assurance

A DACS State-of-the-Art Report

DACS Report Number 347617

Contract Number SP0700-98-D-4000
(Data & Analysis Center for Software)

30 September 2007

PREPARED FOR:

Air Force Research Laboratory
AFRL/IFT
525 Brooks Road
Griffiss AFB, NY 13441-5700

PREPARED BY:

Elaine Fedchak
Thomas McGibbon
Robert Vienneau

ITT Advanced Engineering and Sciences
775 Daedalian Drive
Rome, NY 13441

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* 30 September 2007	**2. REPORT TYPE** State of the Art Report	**3. DATES COVERED** *(From - To)* N/A

4. TITLE AND SUBTITLE Software Project Management for Software Assurance A DACS State of the Art Report	**5a. CONTRACT NUMBER** SP0700-98-D-4000
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER N/A
6. AUTHOR(S) Thomas McGibbon, Elaine Fedchak, Robert Vienneau	**5d. PROJECT NUMBER** N/A
	5e. TASK NUMBER N/A
	5f. WORK UNIT NUMBER N/A
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) ITT Industries, Advanced Engineering & Sciences, 775 Daedalian Dr., Rome, NY 13441-4909	**8. PERFORMING ORGANIZATION REPORT NUMBER** DAN 347617
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Technical information Center DTIC/AI 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060	**10. SPONSOR/MONITOR'S ACRONYM(S)** DTIC
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release, Distribution Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report presents information on how the need for software assurance affects software project management. The impact of software assurance on the tasks and concerns of a project manager are addressed, first in terms of discipline, then in terms of life cycle. It includes a discussion of the business case for justifying software assurance activities, the impact of software assurance on project management tasks and concerns involved in planning, and on project monitoring and tracking during development. The additional activities and project management concerns are identified and presented as they arise with each phase of a development life cycle, and as a work breakdown structure. This report also contains resources and pointers to enable the reader to continue to track the state of the art, beyond this static document.

15. SUBJECT TERMS
SOFTWARE ASSURANCE, SOFTWARE SECURITY, SOFTWARE PROJECT MANAGEMENT, SOFTWARE ENGINEERING

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Thomas McGibbon
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	86	**19b. TELEPHONE NUMBER** *(include area code)* 315-334-4933

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION

There is a growing awareness of the importance of software assurance in the systems we rely on, and concurrently a growing consensus that achieving software assurance requires changes to the way software is developed. There is a sense that pushing assurance as early in the life cycle as possible is not only necessary, but also cost-effective. There is not, however, useful guidance for the project manager who agrees with these assessments to learn how to apply these insights.

A survey of similar products from different providers found that the least secure product carried a six times higher business risk than the most secure one, highlighting the fact that the security quality of a product can vary drastically depending on who designed and implemented it [Flechis 2004].

1.1 Purpose

The purpose of this report is to collect and present information on how the need for software assurance affects software project management. The impact of software assurance on the tasks and concerns of a project manager are addressed, first in terms of discipline, then in terms of life cycle. For example, one of the key elements of project planning is developing a schedule. The project manager knows that there needs to be time in the schedule for software assurance, but may not know what to schedule: when, who, or for how long.

A second goal of this SOAR is to provide tools and resources for quantifying the effects of software assurance on software development, both in terms of planning (cost estimation and budgeting), and in terms of overall cost-effectiveness and return on investment.

1.2 Intended Audience

By Role.
Software project managers. The primary audience for this report is software project managers. These are the people responsible for planning an effort, for tracking progress against the plans during the development, and for juggling resources as necessary to meet objectives.

Secondary audiences for the information in this report are those with whom software project managers interact, for example:

- Mid-level and senior managers in software organizations, to understand (and therefore, support) the requests of the project manager.

- System engineers, to see where and how software assurance concerns impact system-level decisions.
- Software Acquisition managers

By Environment.
The DACS primary audience is the US DoD community, and so the project managers we are addressing are DoD software project managers, including contractors. However, most of what is presented here is equally applicable to similarly sized and scoped software developments in other Government departments and agencies, DHS, FAA, NASA, etc., and other developers of software-intensive systems.

1.3 Scope

The scope of this report is to focus on what project managers need to know to incorporate software assurance into their software development efforts. This is not intended to be a guide to software project management, but rather a cataloguing of the differences between how it has been done before, and what needs to be done now. The tools familiar to project managers: work breakdown structures, Gantt charts, cost models, etc., are just as applicable, but may need additional steps or different interpretations.

Software Assurance.
By software assurance, we mean being able to show that software does “what the specification calls for and nothing else” [Redwine 2007]. This encompasses activities throughout the project life cycle, from initial concept to retirement.

According to the Committee on National Security Systems (CNSS) Instruction No. 4009, National Information Assurance Glossary, software assurance is defined as “the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its life cycle, and that the software functions in the intended manner” [CNSS].

By software assurance, we do not mean information assurance nor do we mean information security. Those are related aspects of the overall problem. We also do not mean security functionality implemented in software (access control, etc.), although such security functionality may be implemented as part of achieving software and/or system assurance.

Software Environment.
The type of software development we are primarily envisioning is where the software is a part of an overall system. In particular, these are software-intensive systems, meaning that most of what the system does is provided by the software components. This does not mean that the information in this report is not applicable to other types of software – most of it *is* applicable – only that where there are differences or where assumptions are made, this report is describing software-intensive systems. For example, embedded software

development, such as on a chip in a remote sensing system, may not be concerned with usability testing the security aspects of the user-interface.

Relationship to Other Documents.
<u>SwAssurance SOAR.</u> The DACS and the IATAC[1] have recently developed a comprehensive state of the art report on software security assurance, *Software Security Assurance: A State-of-the-Art-Report* [SwASOAR]. That report is aimed primarily at software developers, and includes presentation and discussions of methods, tools, and techniques that are emerging or in use.

What we are doing in preparing this SOAR is to take those methods, tools, and techniques, and ascertain what a project manager needs to implement or support them. Wherever applicable, therefore, this document references the appropriate section(s) in that document, using the format [SwASOAR, Sect.#].

Other similar-seeming reports have different objectives, and are written for different audiences:

- DHS SwAssurance Common Body of Knowledge – is aimed at those in academia and elsewhere who define education, training and certification for software professionals, to identify what needs to be taught to address the gap in software practitioners' knowledge about developing secure software. Portions of this document are referenced throughout this report as [SwACBK, Sect#].
- DHS Security in the Software Life Cycle – is aimed at practitioners, and provides specific tools and techniques for doing software assurance. Portions of this document are referenced throughout this report as [SSLC, Sect#].
- DHS Software Assurance Landscape Document, currently under development, is to be focused on organizations working in this area, and identifying gaps in work being done, from the point of view of DHS. This document, when available, will be a useful adjunct to the resources listed in Section 7 of this report.

A longer list of related documents and efforts can be found in [SwASOAR 1.6].

1.4 Assumptions and Constraints

Assumptions.
A primary assumption is that readers of this document have a grounding in traditional software project management. So, for example, it will not be necessary to explain the trade-off triad of schedule, cost and functionality, nor what a cost model is.

The corollary assumption is that readers of this document believe that software assurance is not only a serious issue but also an achievable goal, and so are willing to invest in new ideas and new techniques for their projects' processes.

[1]IATAC is the Information Assurance Technology Analysis Center, another DoD Information Analysis Center (IAC).

Constraints.
This is a rapidly evolving field. Much of what is included as being state of the art is taken from currently active working groups. As their work evolves, so too, does the state of the art. Another primary source, the Build Security In portal [BSI], adds documents and resources frequently.

The software engineering field is not static either. As new processes, tools, and techniques are developed and adopted, astute project managers will need to assess how or if they can be adapted to incorporate software assurance. For example, there is considerable debate about whether or not agile methods can produce secure software [SwASOAR 5.1.8.1 and Appendix F].

The information presented in this report is intended to be descriptive, rather than prescriptive. *Thou shalts* are the province of standards and regulations. Wherever the occasional *should* or *needs to* sneaks into the text, it *should* be read as "this is good advice," or "the intended effect of the technique will be lost without this."

1.5 Organization

The remainder of this report is organized as follows. Section 2 presents necessary background information, some definitions, and discussion of the business case for justifying software assurance activities. Section 3 covers the impact of software assurance on project management tasks and concerns involved in planning. Section 4 covers the impacts on project monitoring and tracking during development. Section 5 walks through a generic development life cycle, identifying the additional activities and project management concerns that arise with each phase. The final sections provide resources and pointers to enable the reader to continue to track the state of the art, beyond this static document. The appendix contains a work breakdown structure which identifies the additional activities that are needed for software and systems assurance.

2 DEFINITIONS AND RATIONALE

"Secure Software Project Management is the 'systematic, disciplined, and quantified' application of management activity to include the 'planning, coordinating, measuring, monitoring, controlling, and reporting' that ensures the software being developed conforms to security policies and meets security requirements" [Abran 2004] [SwACBK 11.1].

Software assurance can not be achieved by careful software engineering alone. Good software engineering is essential for producing secure software, but it is not sufficient. This is because most software engineering has been oriented towards functionality, quality, or reliability. It can be argued that for software, security is an essential aspect of quality and reliability – but this has not always been considered the case, so most quality- and reliability-oriented software processes omit many of the activities necessary for security. If the software is 100 per cent reliable but relies on a component for which a vulnerability is found and exploited, then the software is no longer secure.

The BSI portal has a Project Management content area. The overview article, *Security and Project Management*, states that "Project managers should consider the additional communications requirements, linkage among life cycle activities, and the potential usage environment as these items relate to security needs" [Ellison].

2.1 Definitions

It is our intention to focus on project management for software development that results in secure software, rather than on managing the development of security mechanisms and infrastructure for a system, also known as information assurance. This distinction, however, blurs somewhat in practice. It is also true that more work has been done, for a longer time, and been published, regarding the latter.

Software assurance: The commonly accepted definition is "a level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its life cycle and that the software functions in the intended manner" [CNSS 2006].

Information assurance: All of the mechanisms, infrastructure, policy, practices and monitoring that protect the information in a system.

The place that the distinction becomes blurry is in how the requirements identified for securing the software are implemented. Many requirements for software assurance are satisfied by security functions.

Application security: In practice application security is, in large part, the combination of patch management and vulnerability management best practices, configuration parameters, and technical defense-in-depth safeguards and countermeasures implemented after the application is installed (rather than designed into the application itself), such as application layer firewalls, security gateways, and access controls. Application security focuses mainly on the administration of countermeasures during the deployment and operation phases of the software life cycle.

2.2 Business Case

The importance of having a business case for software assurance can be seen by the inclusion of a Business Case working group in the DHS Software Assurance programs that support the National Strategy to Secure Cyberspace [DHS BCWG]. The goal of this working group is to "[d]evelop a business case analysis to support software security throughout life cycle practices." The Working Group's objective for their Oct 2006 meeting was to analyze the results of a CIO Executive Council survey and discuss their impact on the direction and work products of the Business Case WG going forward. The survey results were then used to introduce and compile information for the WG fact sheets. They also reviewed the *Symantec Internet Security Threat Report: Trends for January 06–June 06*, *Volume X* [Symantec 2006].

A project underway at Carnegie Mellon University's CyLab[2] is likely to provide useful data, once completed. The project title is *Economic Incentive to Improve Software Security*, and the investigators are Ashish Arora and Rahul Telang. They describe their project as follows:

> "A key component of understanding the economic incentives for software vendors to invest in developing more secure software products is the users' willingness to pay for more security. A key hurdle to greater security is that there is little quantifiable evidence on how more secure software would fare in the market place. The proposed research is the first step to developing an empirically grounded framework for analyzing this problem" [Cylab].

Costs of Non-Secure Software.
Even if good, reusable cost of security data is not yet available, there are certainly plenty of news stories and estimates of what the lack of security costs businesses and organizations. The following news clips are representative. In the first two, the numbers are huge, but the losses are aggregated. The next three clips report costs of individual incidents to individual organizations.

Malware Responsible for $55 Billion in Losses Worldwide (16 January 2004)

> Businesses worldwide lost an estimated $55 billion due to computer worms in 2003, according to Trend Micro. Losses in 2002 were between $20 and $30 billion, up from $13 billion in 2001. Trend Micro predicts that figure will

[2] CyLab is a university-wide, multidisciplinary initiative involving faculty, students, and staff at Carnegie Mellon University, aimed at creating a public-private partnership to develop new technologies for measurable, available, secure, trustworthy, and sustainable computing and communications systems and to educate individuals at all levels.

increase again in 2004; the company also believes that blended threats will continue to be the attack of choice [SANS 2004].

FBI Study Pegs Cyber Crime Losses at $67 Billion (19 January 2006)

An FBI study of 2,066 firms found that 90% had experienced cyber crime events and 64% had experienced financial losses from such events. Worms and viruses caused the most damage despite defenses most organizations had put in place. Average losses were $24,000 [SANS 2006].

Data Loss Proves Costly to Companies (14 November 2005)

A pair of surveys conducted on behalf of PGP Corp. found that companies that lose or mishandle customer data suffer significant financial fallout. A survey of organizations found that each security breach cost approximately US$14 million. A survey of consumers found that twelve percent of the 9,000 people surveyed said they had received a notice that the security of their personal data had been breached. Nineteen percent of people whose information was mishandled immediately closed accounts with that company; an additional 40 percent said they were considering terminating their accounts [SANS 2005].

Report: Fears that a Data Breach Could Ruin Business (April 25, 2007)

A new report from McAfee found that of more than 1,400 IT professionals surveyed, a third fear that a major data security breach could put their company out of business. Despite the fact that 60 percent of respondents said their companies had experienced data loss in the last year, they reported spending just 0.5 percent of their IT budgets on data security. Sixty-one percent of respondents believe data leaks are caused by people within the organization, and 23 percent believe those leakages are of malicious intent [SANS 2007a].

This last clip might provide a contradictory story, but for the editor's response.

TJX Sales Up Six Percent in March: Consumers' Actions Speak Louder Than Words (April 13 & 14, 2007)

Interestingly, sales at TJX-owned stores increased six percent in March, despite the disclosure of a massive data security breach earlier this year. One shopper said he felt that because of the breach, which exposed 45.7 million credit and debit cards, the stores were likely taking greater precautions to safeguard customers' data. For other shoppers, the prospect of the bargains to be had at the stores offset concerns about data theft. In addition, TJX's stock was trading last week at US $28, just US $2 shy of where it was prior to the report of the breach. According to a survey from Javelin Strategy & Research, 77 percent of respondents said they would stop shopping from businesses that suffer significant data security breaches, something that seems to contradict shoppers' actions.

Editor's Note: For publicly traded companies, especially retail, revenue reporting is subject to so many variables that every one of the studies that tried to tie public notice of incidents to stock price or market capitalization is completely meaningless. The retail sales factors that vary each quarter provide swings of much greater magnitude than any of the "effects" attributed to security incidents. Banks that have incidents are a different story, as they can very easily see if they lose an account due to an incident. Many banks say they have definitely seen this effect (including getting the blame for retailers exposures), while in retail it is hard to find any retail business that can tie an incident to sales fluctuation [SANS 2007b].

2.2.1 Return on Investment for Secure Software Development

Being able to forecast a positive return on investment (ROI) in software assurance is an attractive goal. As it is for most project management decisions. Note that a positive ROI for even having computer technology in the workplace is debated by many. The "total cost of ownership" for a PC, an application, a system, a network, or a complex inter-connected system of systems is usually unknowable, even after the systems are built, bought and delivered.

The DHS-CERT BuildSecurityIn web site is developing a business case topic area [BSIa]. That is the best place to start, and will probably remain the best place to go for emerging techniques and information. Another source is McGraw's BSI book [McGraw 2006]. The measurement section (pp. 73-75) refers to the a paper from @stake [Soo Hoo 2001].

One model for calculating Return on Investment is presented in Figure 1 [Paquet 2005].

$$AV = C_{info} + C_{sw/hw} + C_{config} + C_{avail} + C_{assoc}$$

Where
AV = total Asset Value of an information asset
C_{info} = Cost of replacing information
$C_{sw/hw}$ = Cost of replacing software and hardware
C_{config} = Cost of reconfiguration
C_{avail} = Cost of loss of availability
C_{assoc} = Associated costs (loss of data confidentiality and integrity)

$$SLE = AV \times EF$$

Where
SLE = Single Loss Expectancy (financial loss expected from a successful attack)
EF = Exposure Factor of asset (fraction of asset value removed by a particular attack)

$$ALE = SLE \times Pr(attack)$$

Where
ALE = Annual Loss Expectancy
Pr(attack) = Probability of an attack of a particular type in a one year period

$$NPV = \sum_{i=0}^{n} \frac{ALEi}{(l + r)^i}$$

Where
NPV = Net Present Value of a security appliance that stops the annual losses
ALE_i = ALE in the i^{th} year
r = an interest rate.

Total Cost of Ownership (TCO) of a security appliance =procurement cost
+ non-recurring costs+ discounted recurring costs

Return on Investment in security appliance =(NPV of avoiding losses – NPV TCO) / (NPV TCO)

Figure 1: Traditional ROI Calculation Based on Discounted Cash Flows

Even if you can't come up with reasonable-looking ROI numbers to present to upper management to support security spending, you may make a different argument in support of your budget request. Consider this opinion from McGraw: spending money on security is more like insurance than investment [McGraw 2006, p. 153]. (Of course, in the insurance business, insurance companies have actuaries and underwriters who have a lot of ROI data, so they know what premiums to charge and still make money when they have to pay claims. As yet, as for a lot of software engineering, the actuarial tables are still being built.)

Another approach is to liken security to a defensive strategy rather than an offensive one. Then investing in security is like investing in a legal department that keeps the company out of trouble. The legal department isn't asked to calculate ROI to justify their budget. "How do you show ROI on a legal department? How do you know that it was because of a certain blurb in a legal statement that stopped a potential lawsuit? Or better yet, without claiming clairvoyance perhaps, how could you tell that a policy created and enforced by your legal department to define how financial information is handled within your company is going to protect your organization millions of dollars in complete process re-engineering costs in the future when compliance with SOX becomes mandatory?" [MSDN 2006]

3 PLANNING

The basic disciplines of project planning remain the same. The purpose of this section is to highlight those aspects of the planning process which impact the ability to develop secure software.

3.1 Risk Analysis

Risk analysis is the area most affected in developing secure software. First, the number and nature of the risks to be identified and managed are considerably different than in a traditional project. Second, the degree to which security risks are anticipated and controlled affects the security of the software and the system of which it is a part.

Risk management, like project management, includes both planning (discussed here) and monitoring (discussed in next section) activities.

"In general, a software assurance risk is based on the criticality and sensitivity of the information processed by the software as well as vulnerability and threat information. An initial step in risk assessment is identifying the risk. Using a risk-based categorization scheme can facilitate software assurance risk identification and is useful in standardizing the results of assessing potential security risks for software intensive systems. Such a security categorization scheme is based on a software-intensive system's criticality to the organization's mission and the sensitivity of the information that it processes" [SwACBK 13.3.4].

The diagram in Figure 2 shows the steps in risk management that are taken across the life cycle. These steps are generic in that they don't specify what the risks are, but rather, what actions or types of analysis are needed. Therefore, this can be used as a guide for a complete risk management, including security risks [Pfleeger 1998].

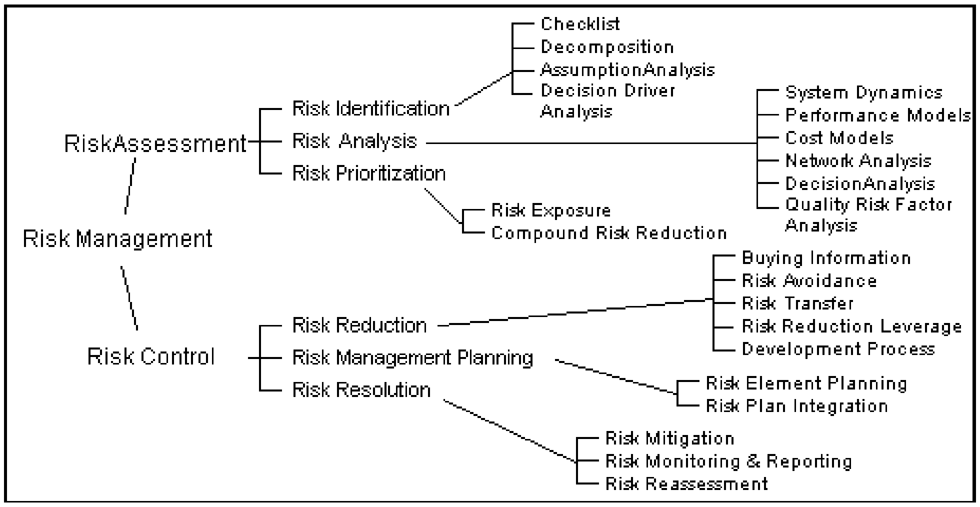


Figure 2: Steps in Risk Management

The only thing different in the beginning is being able to identify risks that affect software security. Typically, a project manager identifies risks to a successful project as schedule risks, cost risks and technical risks. At this stage of the planning, however, there is also a need to consider the potential risks to the product – i.e., the security of the software in its operational environment. Managing for software assurance may cause you to look at your assets differently. The software itself is a target, not just a means for achieving some other goal. In addition, the information processed by the software is also a target, and must be secured. This is where the distinction between software assurance and information assurance becomes somewhat blurred. However developing software that is more secure in itself results in better security for the information it processes.

One of the steps shown in the diagram as a component of risk identification is a checklist. There are many examples of risk checklists available; one example can be found at the website maintained by R.S. Pressman & Associates, Inc. [Pressman]. The checklist approach can be used to identify security risks, by adding items like those in Table 1 [Gilliam 2003].

Table 1: Potential Items for Inclusion in a Software Security Checklist

1 Introduce a walkthrough, security audit review or a formal security review in every phase of the software life cycle development.
2 Establish security metrics during the software life cycle and a trace matrix for security requirements.
3 Determine stakeholders, and elicit and specify associated security requirements for each stakeholder.
4 Determine context and potential usage of software product along with the operating environment and specify requisite security requirements.
5 Make available to programmers, developers, reviewers and test teams the vulnerabilities and potential exposures associated with programming languages and operating systems before the architectural design phase.
6 Set up security parameters for access to services such as ftp service where anonymous ftp is allowed but with write only and no read or list to the incoming directory and read only for outgoing directory.
7 Check for sources of software security risks such as inconsistencies in requirements and in design, reusable programs and other shrink-wrap software. Use of requirements tools, modeling tools, etc. can aid in this area.
8 Avoid the use of unsafe routines such as sprintf(), strcpy/cat(), gets and fgets in coding.
9 Check the security of any middleware in the program.
10 Check for architectural-specific vulnerabilities and how data flows through the code.
11 Check for implementation-specific vulnerabilities such as Race Conditions, randomness problems and buffer overflows.
12 DO NOT allow programmer backdoors or unauthorized access paths that bypass security mechanisms.
13 Avoid storing secrets like passwords in the code or using weak encryption schemes.

14 Identify all points in the source code where the program takes input from users.
15 Identify all points in the source code where the program takes input from another program or un-trusted source.
16 Investigate all sources from which input can enter the program such as GUI, network reads, etc.
17 Check Application Program Interface (API) calls to security modules or interfaces.
18 Investigate secure connections. Verify that they actually are secure and connect as indicated to the systems to which they are intended to connect.
19 Investigate software built-in extensibility features.
20 Review software complexity and look for alternatives to reduce the complexity.
21 Investigate the security of the data when passed from application servers to databases.
22 Avoid default or other improper configurations that may open the door to attackers.
23 Default to "highest security" needed, and require validation and approval for deviations.
24 Establish tools to be used for various stages of the life cycle that will be used for assessing security.
25 Perform security testing for unit and system integration.
26 Potentially, establish a security risk rating criteria and document the rating of the software product within the organization. Using a risk assessment tool can benefit this area.

Once security risks are identified, they are treated like the other risks in the risk management process, they are analyzed, decomposed, and prioritized, and mitigations are identified/proposed/planned/defined.

One of the steps shown in Figure 2 as being a component of risk prioritization is risk exposure. In order to conduct a useful analysis of the identified risks, it is necessary to consider the likelihood of their occurrence. That is where an understanding of the threats to software (and information) comes in.

It would be prudent to include people with specialized knowledge of information security and software security in the risk identification and analysis processes, in the same way as you may already use people who are specialists in risk assessment to do risk identification and analysis.

3.1.1 Threats

An effective risk analysis requires knowledge of the threat environment. Chapter 2 of the Software Assurance Common Body of Knowledge document provides a good introduction to the dangers facing software. It covers types of attackers, attack motivations, and attack methods as well as threats arising from non-malicious (unintentional) events [SwACBK 2].

There are a number of programs and efforts devoted to threats, to identifying, understanding, classifying, standardizing, and tracking them. These include several efforts sponsored by DHS. Others are products of commercial and international consortia and working groups. Closely tied to understanding threats is understanding vulnerabilities, and there are related programs for those as well.

- CVE – Common Vulnerability Enumeration. CVE aims to standardize the names for all publicly known vulnerabilities and security exposures [CVE].
- CWE – Common Weakness Enumeration. The objective of CWE is to provide a unified, measurable set of software weaknesses that will enable more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code [CWE].
- CME – Common Malware Enumeration. The objective of CME is to provide single, common identifiers to new virus threats and to the most prevalent virus threats in the wild to reduce public confusion during malware incidents [CME].
- SAMATE – Software Assurance Metrics And Tool Evaluation. One objective of the DHS Software Assurance Tools and R&D Requirements Identification Program is the identification, enhancement and development of software assurance tools. SAMATE is a software assurance tools taxonomy [SAMATE].
- CAPEC – Common Attack Pattern Enumeration and Classification. The objective of this effort is to provide a publicly available catalog of attack patterns along with a comprehensive schema and classification taxonomy [CAPEC].
- OVAL – Open Vulnerability and Assessment Language. OVAL is an international, information security, community standard to promote open and publicly available security content, and to standardize the transfer of this information across the entire spectrum of security tools and services [OVAL].

There are similar types of efforts going on in the non-government world. Two examples from the open source community are:

- WASC Threat Classification – The Web Application Security Consortium (WASC) Web Security Threat Classification is a cooperative effort to clarify and organize the threats to the security of a web site [WASC].
- OWASP – Open Web Application Security Project. OWASP supports many projects, including the Top Ten consensus list of the most critical web application security flaws, and the Guide to Building Secure Web Applications [OWASP].

These efforts are generally developing searchable on-line databases. The details are massive, overwhelming, and constantly growing. The audiences and intended uses of these databases are not primarily software project managers, of course. Some of the goals are to promote information sharing, and to influence tool developers, etc. The reason they are mentioned here is that they can help direct a risk analysis. Just knowing the overall classifications, without diving into the details, can guide risk identification, so that you know you have covered all the types of things that need to be considered. Later in the life cycle, these databases can be used again for developing requirements, doing test planning, etc. The standardization promised by these efforts will also help you communicate with information assurance and security specialists as your project evolves.

Threats are categorized by source:

- Malicious insider
- Non-malicious insider – intentional or unintentional
- External attacker

By attacker capability/motivation:

- Script kiddie for kicks
- Skilled hacker for notoriety
- Criminal for profit
- Nation-state for political or military objective

By type (each affects a different security property):

- Sabotage compromises availability
- Subversion compromises integrity
- Interception compromises access control
- Disclosure compromises confidentiality

By what is attacked:

- Code
- Other artifacts of development
- Installation parameters
- Through interfaces during use

Or by when in the life cycle the threat occurs.

Another source of help for threat modeling is in the security-enhanced development methodologies that are discussed in [SwASOAR 5.1.8].

- CLASP, alone or as plug-in to RUP
- Microsoft Trustworthy Computing SDL
- TSP-Secure
- AEGIS
- RUPSec
- Secure Software Development Model (SSDM)
- Oracle Software Security Assurance Process
- Waterfall-Based Software Security Engineering Process Model
- McGraw's Seven Touchpoints for Software Security
- Security Extensions to MBASE

3.2 Size and Cost Estimation

Perhaps the most daunting question a project manager has when faced with developing secure software is, “How much more is it going to cost me to do this?”

There is and has been much work going on to help managers answer this question. Not all of it is aimed specifically at software assurance; much more is concerned with information assurance and system or network security. But since those areas have a head start, their results can be used as starting points.

An example is the CONIPMO model. This is an extension to COCOMO concerned with the cost of securing network infrastructure. The stated motivation for this work is equally applicable to software assurance:

> “To establish adequate budgets, management must be able to determine how much time and effort is really needed [to put adequate defenses in place]. The management must understand what portions of these budgets should be considered investments versus project costs. To provide managers with the ability to prepare such estimates, more accurate cost estimating models are required.” [Reifer 2007]

The cost of security is not something that can be calculated for one project and then reapplied intact. The effects of software assurance on cost and effort depend on how much assurance is required. The Evaluation Assurance Levels (EAL) in the Common Criteria (CC) define a range of activities and requirements that become more numerous and expensive as the assurance level increases. The cost to develop and certify a system at EAL7 is significantly more than one at EAL2, enough to make use of formal methods look cost-effective.

COSECMO.
COSECMO is an extension to COCOMO that is specifically targeting the cost of software assurance. It is still under development at the Center for Software Engineering at USC. Their model development process for developing extensions to COCOMO includes several iterative steps that feed back into model refinements. Once the initial model is formulated and tentative parameters defined, the model is validated against both expert judgment and project data. The COSECMO model is currently in this phase of development. Therefore, the final model may not conform exactly to the factors described in the existing documents.

This model is the closest to providing exactly what a software project manager needs for estimating the cost of software assurance, although still, the line between security of the software and the security functions that protect the system is indistinct.

COSECMO adds project scale factors to COCOMO II. Factors include maturity, risk, flexibility, teamwork and precedentedness. Not all of these factors are strictly security or assurance factors, but all will affect the cost of assurance.

The COSECMO developers are defining a Security Driver Rating Factor for Development (they also have factors for operational and physical security) that will measure/predict the effect of processes for development and validation of security-critical software. The following is a list of what COSECMO includes in the security driver during software development. The added work is broken out by security level (driver SECU), that ranges from None to the Evaluation Assurance Level 6 (EAL6) as defined in the Common Criteria. Each higher level adds to what was included in the prior level. At each level, the additional work is categorized by life cycle stage.

Low (no EAL equivalent)
- No security requirements
- No protection other than provided execution environment

Nominal (EAL 1-2)
- Requirements - Informal security requirements formulated for system
- Design - Analysis of security functions using
    - Informal functional and interface specification
    - Descriptive high-level design
    - Demonstration of corresponding pairs
- Testing - Developer tests implementation of requirements
    - Black box testing
- Life cycle controls - Simple Configuration Management (CM) with version numbers

High (EAL 3&4)
- Requirements - Fully defined external interfaces
    - Informal security policy modeling
- Design - Security enforcing high-level design
    - Informal low-level design description
- Testing - Independent testing of all functional requirements
    - Inspection of COTS source code if available
- Life cycle controls - CM with unique referencing
    - Detailed delivery and installation procedures
    - Identification of security measures for life cycle

Very High (EAL 5)
- Requirements - Semi-formal functional specifications
    - Formal security policy modeling
- Design - Semi-formal high-level design
    - Modular implementation
    - Wrapper & dynamic analysis for COTS and open-source
- Testing - Evidence of coverage for all developer test results
    - Testing of high-level design
    - Independent vulnerability analysis
    - Independent validation of analysis
- Life cycle controls
    - Partial automation of CM
      –with authorization control, problem tracking, and detection of modification

    - Developer-defined life cycle model
      –with well-defined development tools

Extremely High (EAL6)

- Requirements - Fully defined external interfaces
    - Informal security policy modeling
- Design - Semi-formal high level explanation
    - Structured implementation with reduction of complexity
    - Secure container for COTS and Open-source
- Testing - Analysis of coverage of tests
    - Ordered functional testing with tests of low-level design
    - Covert channel analysis
- Life cycle controls - Compete automation of CM
  –with coverage for developer tools
    - Standardized life cycle model
      –with compliance to implementation standards

[Colbert 2002]

A further complication of security in terms of effort is the effect that increased security has on other cost drivers. For example, high security software will be more complex. The COSECMO model includes adjustments for the drivers Reliability, Complexity, Multi-Site, Documentation, Tools, Size and Risk on security level.

An example of the estimated impact of security by assurance level, once all the drivers are factored in, is shown in Table 2 [Colbert 2002].

Table 2: COSECMO Percent Added Effort

System Size (KSLOCS)	Assurance Level					
	Nominal EAL1-2	High EAL3	Very-High EAL4	Extremely-High EAL5	Super-High EAL6	Ultra-High EAL7
5	0	20	50	125	312	781
10	0	40	100	250	625	1560
100	0	60	150	375	937	2344
1000	0	80	200	500	1250	3125

These results are based on analysis of existing data, with a calibration constant of 2.5. As the effort continues, they are currently validating the model parameters against additional projects.

Others – ask Pete (NASA).
Ask Pete is a software project planning tool which incorporates methodologies for cost and schedule estimation through COCOMO II; corporate risk and resource investment through the use of the Control Level methodology; and the need for external verification and validation activities through the NASA IV&V criteria. Ask Pete incorporates the following tools:

- COCOMO II using SLOC or Function Point Estimates

- NASA Glenn Research Center's (GRC) Software Development Procedure and Control Levels
- NASA's IV&V Criteria
- Plan templates
- CMM Checklist

Results of an ask Pete session are available in an external database file that can be used by other applications, for further planning and risk management activities. Ask Pete was developed at Glenn Research Center with funding from NASA Office of Safety and Mission Assurance and GSFC IV&V Facility. The application is freely available and can be obtained from the Ask Pete web site at http://tkurtz.grc.nasa.gov/pete [Kurtz 2001].

COTS.
There has been a lot of research into developing estimates and models of the cost of development using COTS. Some of these now include security implications [Minkiewicz 2005].

3.3 Scheduling

Scheduling in project management is a straightforward task that becomes a juggling act. The project management body of knowledge area on scheduling summarizes the scheduling process as:

- Define tasks
- Sequence tasks
- Estimate resources needed for each task
- Estimate duration of each task
- Combine all these into a schedule

There are a number of standard and familiar tools and techniques available to manage schedule development.

- Gantt
- PERT
- MS Project

Schedules, like most other management activities, require monitoring and adjustment throughout the development life cycle. Although you start out following your plan, you need to monitor project performance against the estimates in the plan and adjust it as necessary. Changes to (almost) anything in the project will affect the schedule. The concept of critical path is used to identify which activities affect the schedule directly, and which can grow or shrink without affecting the overall progress.

Additional Activities.
The primary effect that managing for software assurance will have on scheduling is allocating time for the additional activities that security requires. For scheduling purposes these would be the same activities that were added for cost estimation purposes.

Scheduling-specific issues for the additional activities include both sequencing and staffing. The additional reviews, for example, must be inserted at the appropriate places in the life cycle, especially if the review uncovers problems or changes that must be made.

In terms of staffing, some of the additional activities may require the use of specific people with specific skills (that are not traditionally part of the software development team). Two examples of these, needed at different points in the life cycle, are risk analysis experts, and penetration testers. The schedule therefore needs to accommodate them. They may need to be borrowed from another organization, and so that must be coordinated with their managers.

Additional Reviews.
There needs to be a security/assurance focused review at every phase of the life cycle where work transitions from one type of analysis/development to another. These do not need to be formally separate reviews, but security issues must be included with the rest of the review. Therefore time and resources must be scheduled for each of the following:

- Security requirements review
- Security architecture review
- Security design review
- Security interface review
- Security code review
- Security test readiness review
- Security integration review
- Security release review
- Certification & Accreditation
- Assurance Case review

Additional Types of Testing.
Similarly, there are additional types of testing needed to properly assure that the software is secure. These must be planned and allocated within the overall test schedule. Examples include:

- Black-box testing
- Penetration testing
- Abuse case testing

More information on testing implications for secure software is contained in [SwASOAR 5.5].

Scheduling Resources.
The lists of additional activities found in COSECMO and other models, and in the security-enhanced life cycle processes/methodologies can be used to compile the set of activities that must be scheduled, as can the work breakdown structure in Appendix A of this SOAR.

3.4 Staffing/Communications

For software assurance, "the competence of personnel is very important. Competencies include:

- Technical skills, especially knowledge of security
- Knowledge of techniques for achieving low defect density
- Domain knowledge, i.e., understanding of the application area
- Communication skills
- Personal attributes, such as integrity" [SwACBK 11.1]

As can be seen from the cost drivers in COCOMO, staff ability has the greatest impact on project cost, after size.

"The levels of organizational and individual experience and proficiency can mean that the result is anything from a substantial increase in hours needed if new or a low to modest increase if highly experienced and skilled" [SwACBK 11.2].

"Always important, properly staffing a project with requisite skills is even more important and difficult for secure software projects. A central group with in-depth security expertise may help stretch this scare resource across all the projects in need" [SwACBK 11.6.1].

Trustworthiness.
Software assurance requires consideration of the insider threat. It is important to know who your own workers are. But you must also know (or mitigate not knowing) who (and where) your subcontractors are.

"People who are intelligent, highly skilled, and knowledgeable about secure software may be hard to find and recruit, and require careful management to ensure retention. In addition, care needs to be taken to avoid personnel security problems. More than routine background checks on personnel producing software where security is an important concern occur in commercial as well as government organizations. Personnel need to be resistant to succumbing to attempts at corruption or recruitment, to conceal problems, or to disclose sensitive information. Different levels of background checks may be desirable depending on a person's role. For example, one might have additional levels of checks on personnel who do ethical hacking" [SwACBK 11.6.1].

A recent news story illustrates this:

Navy Computer Sabotage Draws One-Year Prison Sentence (April 5, 2007)

> A former government contractor has been sentenced to one year in prison for sabotaging Navy computers after his company's bid for another project was not accepted. Richard F. Sylvestre has pleaded guilty to one count of damaging protected computers; he could have faced up to 10 years in prison. Sylvestre's company at the time, Ares Systems, had a contract to maintain computers for the Navy's 6th Fleet in Naples, Italy. Sylvestre admitted to placing malicious code on the Navy computers. The computers were used to help submarines navigate and avoid collisions with undersea hazards and other submarines. Sylvestre has also been ordered to pay a fine of US $10,000 and will serve three years probation following his release from prison. He has repaid the Navy US $25,000 for damages [SANS 2007c] [McGlone 2007].

And this comment on the story, by Stephen Northcutt, provides further emphasis:

> "It is important to memorize a few stories like this one, and share them with others, because most organizations do not give enough attention to the insider threat. It is natural to want to trust your own people. Richard has had access to DoD systems since at least year 2000 as the link below shows, so you have to wonder what else he has done to reduce the security of our nation's computers: http://www.defenselink.mil/contracts/contract.aspx?contractid=1808" [SANS 2007c].

Training.
Developers need to know how to develop securely. For coding, this is being addressed – there are a number of checklists being compiled. A table of them can be found in [SwASOAR 5.4.1]. Some of the major information assurance and network security training organizations are beginning to offer training for developers, to reduce the problems that their traditional audiences have to deal with. For example, the SysAdmin, Audit, Network, Security (SANS) Institute offers the Web Application Security Workshop for application developers and security professionals. This is training for developers to learn how to implement security measures during the software development life cycle [SANS 519]. Although this particular course focuses on developing web applications, similar offerings are becoming available for other types of applications, languages and environments.

Developers also need to know about security for requirements. They need to know what non-functional security requirements are, how to elicit them, how to analyze them, and how to implement them. For design, they need to know security principles, such as least privilege, and know about security design patterns.

As the level of security required on a project increases, developers need to add formal methods to their skill sets.

These needs may be addressed by training your staff in use of security-enhanced methodologies (CLASP, TSP secure, RUPSec, …), and/or tools.

The DHS Software Assurance forum has a working group for education and training. One of the Workforce Education & Training Working Group's goals is to have software security and assurance successfully included in higher education, workforce training, and elsewhere via propagating the Software Assurance Common Body of Knowledge and systematizing software system security principles and guidelines [SwACBK].

Certifications.
The SANS institute has recently announced a secure programmer certification.

> "SANS Institute has announced the first certification examinations for programming professionals to gain Certified Application Security Professional (CASP) status. The examinations cover four specific programming language suites: (1) C/C++, (2) Java/JSP, (3) Perl/PHP, (4) .NET/ASP. The exams are designed to enable reliable measurements of technical proficiency and expertise in identifying and correcting the common programming errors that lead to security vulnerabilities.
>
> The secure programming certification exam provides a focused approach for programming professionals who want to identify the gaps in their secure coding skills and knowledge, and it will allow employers of those programmers to differentiate their organizations and help increase their competitive advantage by employing programming professionals who have successfully demonstrated their technical skills through certification. For the exam to be viable, it needs to be updated continuously to reflect changes in languages as well as newly discovered methods of code or execution compromise. The test developers intend to work closely with the SANS vulnerability tracking team and the Common Weaknesses and CVE program at MITRE to ensure that the Secure Programming Skills Assessment (SPSA) exam is measuring what it needs to measure at all times.
>
> The lack of trustworthy standards and certifications has been a challenge for software buyers and software developers; this exam will be part of a comprehensive secure coding improvement strategy that will help both buyers and sellers of software" [Paller 2007].

The certifications are offered for the following initial set of languages:

- Secure programming skills using JSP and Java
- Secure programming skills using C/C++
- Secure programming skills using PHP and Perl
- Secure programming skills using ASP.NET and .NET

The Information Systems Security Association (ISSA) maintains a list of certifications [ISSA]. Most of them are the more standard information assurance certifications, which are applicable for network security analysts. Of note, however is the Software Security Engineering Certification (SSEC) from Security University.

> "Security University Software Security Engineer Certification is a number of classes that make a Software Security Engineer Certification. This certification is for anyone interested in securing software from flaws and bugs, with how to break code, and best practices for checking your code, to penetration testing your code. These classes and certification are new and will provide consistent, extreme

> hands-on software security labs and classes with trademarked escalating workshops and performance based training for security, IT professionals and now coding /developers" [Security University].

The International Council of Electronic Commerce Consultants (EC-Council) offers the EC-Council Certified Secure Programmer (ECSP) and Certified Secure Application Developer (CSAD) certifications [EC-Council].

Many of the standard software engineering certifications, such as Microsoft Certified Application Developer (MCAD) and Solution Developer (MCSD) are adding coverage of security [SwACBK 3.3].

Communications.
Most competitive businesses know that it is a good idea to keep business information private. There are known risks and costs associated with leaks of proprietary data. What software assurance adds is the need to protect information about software from a different set of potential adversaries. Unlike competitors who may be after the "how it works" or "how it is going to look and feel" scoop, software assurance adversaries are after "how it can be made to not work" or "how it can be made to divulge information." Release of a software product at the end of the development life cycle used to mark an end to the need to keep some of this information private. Such a milestone does not affect the hacker's interest, however, and so design and other types of information that may normally be releasable after launch may still need to be protected.

Depending on the culture that you are coming from, such reticence may be a new concept for the project team.

"Continuing communications about the importance of security, security procedures, and what to do if approached or suspect others are necessary to maintain a proper level of attention and discipline among personnel. Vigilance and proper processes can reduce the chances of successful subversion. This means implementing separation of duties and privileges, as well as the principle of least privilege. The rule that at least two persons are fully aware of – and completely understand – any given thing on the project must be rigorously planned and ensured. The concept of separation of duties should extend to thorough reviews at all levels, including review of the 'final' version resulting in final approvals by other than the author and accompanied by rigorous (and secured) configuration management throughout, including ensuring the version reviewed and approved is the version delivered" [SwACBK 11.6.1].

Working Environment.
The level of security required on a project will affect the working environment. The amount and complexity of controls needed for access, secure communication, and secure storage increases with the security level. This is addressed in the COSECMO model with the SITE driver. It is similar to the effect that working on a classified effort has. As the classification level increases, the processes and controls are more rigorous, because the consequences are higher. Software information should be protected during development

as if it were classified, and the degree of protection needed depends on the results of the risk assessments that are done at the beginning of the effort.

"Security is an essential issue in establishing secure physical, communications, and computing facilities. Personnel need to be not only highly skilled and knowledgeable about security concerns within their roles but also trustworthy" [SwACBK 11.2].

"Project managers need to supply the environment and resources the staff needs to do high quality work and the needed information, guidance, inspiration, motivation, emotional support, perseverance, and discipline to achieve the highly demanding level of rigor. Additionally, the manager preferably will do this in a fashion that is suitable to the professionals involved and allows everyone to benefit while experiencing an acceptable quality of life" [SwACBK, 11.6.1].

"A development or sustainment environment with at least the level of security required of the product is best for developing secure software." "[Ibrahim 2004] contains a section on the work environment. The work environment needs to not only be secure but also contain the proper tools and equipment for the approach taken to secure software." "Physical security is also a concern, as good physical security is essential to maintaining information (and industrial) security. Commercial needs or customer requirements may call for a higher than normal level of operational security" [SwACBK 11.6.2].

3.5 COTS / Acquisition

Much of the software that ends up in a final product is non-developmental. This applies not only to commercial-off-the-shelf components, but also to re-used (legacy) code, and to basic library functions that come with the programming language or environment being used. It is nearly impossible to build an entire application from scratch without using some piece of code that was acquired from elsewhere. As with the expression "the chain is only as strong as its weakest link," the software product is only as secure as its most vulnerable component. Due to the inescapable need for non-developmental software, there is a lot of work and research into the security/assurance issues involved.

For example, the DHS Software Assurance Forum has a working group devoted to software acquisition. The working group's goal is to enhance software supply chain management by leveraging the acquisition process to ensure safety, security, reliability, and dependability of software. The group's action items are to:

- Develop a Software Assurance (SwA) guide for acquisition officials
  - SwA activities in all phases of the acquisition process
  - SwA Due Diligence questionnaires
  - Sample Request for Proposals and contract language
- Disseminate materials to organizations providing acquisition training and education
- Recommend changes to IEEE Std 1062 and other software acquisition related policies, procedures, standards, and processes [DHS AWG]

According to the SwACBK, "make versus acquire decisions are compounded by security requirements and needs for assurance…Showing that security properties are preserved during the composition of parts from inside and outside the project is, of course, also an obligation" [SwACBK 11.6.3].

The SwACBK devotes an entire section to the subject of acquisition. The audience for that section is people in the acquisition community, more so than developers. But as a developer who is creating software that will be acquired, this guidance is worthy of perusal. Section 13.6, in particular, is written for the supplier.

The topics covered are shown in Table 3 [SwACBK 11.6.3].

Table 3: Acquisition Topics

13.2 CONCEPTS, TERMS, AND DEFINITIONS
13.2.1 Acquisition
13.2.2 Off the Shelf Software (OTS)
13.2.3 Information Assurance Architecture
13.2.4 US NIAP
13.2.5 Security Accreditation
13.2.6 Security Certification
13.3 PROGRAM INITIATION AND PLANNING--ACQUIRER
13.4 ACQUISITION AND SOFTWARE REUSE – ACQUIRER/SUPPLIER
13.5 REQUEST FOR PROPOSALS – ACQUIRER
13.6 PREPARATION OF RESPONSE--SUPPLIER
13.6.1 Scope
13.6.2 Initial Software Architecture
13.6.3 Initial Software Assurance Plan
13.7 SOURCE SELECTION–ACQUIRER
13.8 CONTRACT NEGOTIATION AND FINALIZATION
13.9 PROJECT/CONTRACT MANAGEMENT – ACQUIRER/SUPPLIER
13.10 FURTHER READING
13.11.1 APPENDIX A: NOTIONAL Language for the Statement of Work
13.11.2 APPENDIX B: NOTIONAL Language for Instructions to Suppliers

4 TRACKING

The second component of project management is tracking. Once the project is underway, all the planning is for naught if there is no correlation between the plans and the actual performance.

Tracking typically focuses on work performed (by percentage, for example) versus work scheduled, and money spent versus amount budgeted. This tracking is used to measure progress, assess the health of the effort, and report to management.

There are many standard tools for project tracking in this sense, taught in any project management training, such as PERT, Gantt, and various graphs. A good resource is the Software Project Managers Network / Airlie Software Council [SPMN][3].

An adept project manager must also be cognizant of events that affect the plans, examples of these are requirements changes (manageable versus scope creep) and adjusting for risks that are realized (personnel change).

4.1 Risk Management

"As in scoping the project, possible consequences or risks are a constant concern throughout secure software projects [McGraw 2006, C2]. Product-affecting ones are explicitly addressed in the assurance case, but project-oriented ones not directly affecting the product but rather such items as schedule or costs must also be addressed, including attacks on the project and subversion of products. One essential difference within many secure software projects is the lack of or irrelevancy of probabilities. This means that consequences rather than risks are what must be managed. One attempt to bridge the gap between possible consequences and probability analysis is [Baskerville 2003]. Theoretically, project managers might benefit from knowledge of game theory with its techniques that do not depend on knowing the probabilities" [SwACBK 11.4].

Not much will change to manage assurance risks – the same tools and techniques can be used. A key will be to have identified software assurance risks in the planning stages. What will change is the level of attention that needs to be paid to risk management. Due to the volatility of the threat environment, analysis of software assurance risks that was performed at the beginning of an effort is more likely to become outdated than typical project risks. So the choice of tool/technique must consider the need to re-assess risks as the project evolves. For example, a probability of low may have been assigned to the risk of using some software library component, with an impact of moderate. Then, at some point during the development life cycle a new type of attack is found that affects the library component. If there is no patch forthcoming or useable workaround, then the risk

[3] Note: SPMN's web site has been acquired by ISC ACE: Integrated Computer Engineering, A Division of American Systems Corporation.

of using that component should be changed to high. This new high risk needs to be mitigated somehow, and depending on where the effort is in the life cycle, different mitigations would be employed.

Risk Management is a process area for CMMI. The monitoring includes continuous assessment (re-assessment) of risks as the project evolves, such that risk identification, planning, and mitigations are not over once the project starts.

For the duration of a project, and the operational life of a software system, risk management is the mitigation of risks as they occur. This entails handling of identified risks, and, as a prerequisite, being able to identify a risk situation.

In addition to using measurement techniques to track progress against cost and schedule, it will be necessary to track changes in the threat environment that may affect the software. This may involve tapping someone from the information assurance or network operations security function to provide you with information on threats, since they are probably already tracking threats for their own purposes.

4.2 Metrics and Measurement

Metrics and measurement are the tools a manager uses to be able to track a project against the plans and schedules. Well-known and familiar measures such as percent completion, hours spent, lines of code, defects detected, etc., and all their ratios and permutations (defect density, SLOC/person, $spent/$budgeted, etc.) will of course still be useful. None of these provide a view of the progress of assurance activities, however, so additional measurements must be taken.

DHS Software Assurance Forum Measurement Working Group.
The DHS Software Assurance Forum has a measurement working group. Their goal is to define what metrics should be collected to assess the security of software in development. They have produced a draft guidance document that, "provides a measurement approach for quantifying and assessing the level of integration of software assurance techniques into the software development life cycle (SDLC) and evaluating the results of such integration in terms of increasing trustworthiness of the code. The approach described in the document leverages existing measurement approaches used or already under development in software and system development and in information security industries. It points out the common features among the approaches and is meant to help the measurement professionals integrate a broader set of measures into their practice without abandoning their current approaches. The authors of the guide are hoping to facilitate compatibility of network, system, and software testing, assessment, and monitoring tools output to integrate data sources for measurement by using existing measurement approaches" [DHS MWG 2007].

The measurement approaches leveraged by the working group in this guidance include:

- Capability Maturity Model Integration (CMMi) Measurement and Analysis Process Area
- CMMiGQ(I)M Capability maturity Model Integration Goal Question Indicator Metric
- Practical Software and System Measurement (PSM), also known as ISO/IEC 15939 Software Engineering – Software Measurement Process
- NIST Special Publication 800-55, Security Metrics Guide for Information Technology Systems
- NIST Special Publication 800-80, Guide for Developing Performance Metrics for Information Security
- Committee Draft (CD) International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) 27004, Information Security Management Measurement

A second goal of the working group is to "facilitate compatibility of network, system, and software testing, assessment, and monitoring tools output to integrate data sources for measurement." That is, to work with developers of various software engineering tools so that the tools provide useful (and compatible) measures.

The working group plans to collect and publish measurement resources – articles, papers, standards – along with the lists of goals, questions, and measures that could be applicable to software assurance that they are developing.

The version 2.0 draft guidance identifies "percent of product defects that negatively impact the security posture of the system" as a relevant measure for project management for a generic system or software development project. In order to determine this number, however, you must be able to identify software defects that may be exploited in the future. How to do that is not described.

DHS Build Security In.

The Build Security In (BSI) site has a measurement practice area, which so far contains an excellent overview paper. The focus is on measurement for secure development, unlike the majority of security measurement guidelines which address security measurements of system or network operations. It suggests a process, measures for the process, measures for the product under development, and tools [McCurley 2007].

Some examples of suggested measures include:

- Number of security defects discovered in-house versus in the field
- Number of security defects detected in strategy or design versus in the field (repeat for each phase)
- Predicted versus actual labor costs for fixing defects at each stage of development
- Security defects per thousands of lines of code (KLOC)
- Number (and percent) of security defects considered low/medium/high/critical
- Number (and percent) of security defects fixed
- Average or median time (and cost) to fix each defect

- Quartile rankings for each developer group, based on defects/KLOC and average or median time to fix, ranked by severity
- Number of known vulnerabilities in the system

Practical Software Measurement.
The Practical Software Measurement (PSM) initiative is a rich source of information and guidance on measurement in general [PSM]. PSM is sponsored by the Department of Defense and the US Army, and supported by a number of commercial practitioners, too. Its goal is to provide project managers with objective information needed to successfully meet cost, schedule, and technical objectives. The PSM guidance is based on best measurement practices of DoD, government and industry programs, is a flexible process, defines an information-driven analysis approach, and provides a basis for enterprise level management. Due to its DoD sponsorship, it supports current software and system acquisition and measurement policy, but also is compatible with the ISO/IEC 15939 standard, Software Measurement Process.

The PSM book is the official, definitive guide to PSM written by the leaders of the PSM development initiative [McGarry 2001]. It describes the principles and practices for developing, operating, and continuously improving an organization's measurement program. It uses real-world examples to illustrate practical solutions and specific measurement techniques. The book examines the foundations of a software measurement program in depth, defining and prioritizing information needs, developing a project-specific information model, tailoring a process model to integrate measurement activities, and analyzing and understanding the results. Specific topics include:

- The relationship between project- and organizational-level measurement
- Defining an information-driven, project-specific measurement plan
- Performing measurement activities and collecting data
- The basics of data analysis, including estimation, feasibility analysis, and performance analysis
- Evaluating the effectiveness of the measurement processes and activities
- Sustaining organizational commitment to a measurement program
- Key measurement success factors and best practices

PSM has a working group addressing the need for measures of security. They have developed a White Paper that reports on research on the application of measurement principles to the security properties of software-intensive systems. The work has the objective of integrating security measurement with the general measurement principles as developed by the PSM project and in accordance with the related standard ISO/IEC 15939:2002. The application of measurement principles to security is a relatively new field and presents several challenges, explored in the paper [PSM 2006a]. This same working group has produced a companion white paper on safety [PSM 2006b].

PSM holds an annual users group conference. Presentations on measures for security have been included at the last several of these. They encompass the broader definition of security, including information assurance, and are not strictly measures for software assurance. These are available from the PSM website, as are the Security and Safety working group workshop materials:

- Getting Started with Measuring Your Security [Moss 2006a]
- Software Assurance Measurement Requirements: Information Needs for IA and Cyber Security [Jarzombek 2005]
- IA Metrics - Why and how to Measure Goodness of Information Assurance [Bartol 2005]
- Costing the Development of Secure Systems [Colbert 2004]
- Safety and Security [Caseley 2003]

The PSM site also has presentations from other meetings and working groups in which work on measurement for security is discussed. For example:

- Mature and Secure: Creating a CMMi and ISO/IEC 21827 Compliant Process Improvement Program [Moss 2006b]

4.3 Configuration Management

Configuration management is a familiar set of tasks and processes that have a profound effect on the manager's ability to control a project. When the need for software assurance is added to a project, then CM can be used to help manage and control assurance-related issues.

For example, strict configuration control of software artifacts and supporting data helps ensure the trustworthiness of those artifacts throughout the development life cycle, by eliminating opportunities for malicious developers to sabotage the security of the software. By contrast, inaccurate or incomplete CM may enable malicious developers to make unauthorized or undocumented changes to the software. Lack of proper software change control, for example, could allow rogue developers to insert or substitute malicious code, introduce exploitable vulnerabilities, or remove or modify security controls implemented in the software.

By tracking and controlling all of the artifacts of the software development process, CM helps ensure that changes made to those artifacts cannot compromise the trustworthiness of the software as it evolves through each phase of the process. For example, establishing a configuration baseline has a significant security implication in CM because it represents a set of critical observations and data about each development artifact, information that can then be used to compare known baseline versions with later versions, to help identify any unauthorized substitutions or modifications [SwACBK 11.8].

Secure CM needs include methods that preserve the security of software. These methods include:

- Increasing developer accountability for software development artifacts by increasing the traceability of software development activities

- Ongoing impact analyses and control of changes to software development artifacts
- Minimization of undesirable changes that may affect the security of the software [SwASOAR 5.1.6]

The most effective way to improve/implement CM is to use a sufficient tracking and control tool. Paper-based CM, while theoretically possible, is unworkable in practice because no one will use it. Having a tool that to implements CM allows things such as the following recommendations to actually be accomplished:

- Developers and testers should have to authenticate to the CM/version control system using strong credentials (e.g., PKI certificates, one-time passwords) before being allowed to check out or check in an artifact
- Check-ins should be digitally signed

Over the life a project, CM needs to be applied to more than just the code, because a malicious developer could alter the requirements specification, either inserting spurious requirements or deleting valid requirements. Similar alterations could introduce security defects into the design, and/or modify test plans or results to remove evidence of such sabotages.

Some examples of artifacts that would be included in the CM system include:

- Threat models
- Use/misuse/abuse cases
- Requirements, architecture, and design specifications
- Source code and binary executables
- Test plans/scenarios/reports/oracles, code review findings, vulnerability assessment results
- Installation and configuration guides, scripts, and tools
- Administrator and end user documentation
- IV&V documents (includes, e.g., SSAA, other C&A documents, CC security target description)
- Security patches and other fixes

Criteria for determining which development artifacts should be placed under configuration control as configuration items include:

- Items that are mission critical, security critical, safety critical, or high risk
- Items that, if they failed or malfunctioned, would adversely affect security, human safety, or mission accomplishment, or would have a significant financial impact
- Items for which an exact configuration and status of changes must be known at all times
- The development artifacts of high-consequence software should always be designated as configuration items

Effective countermeasures for insider threat (and sloppiness) that can be implemented through CM include:

- Configuring the CM system to automatically create a digital signature and time stamp for each artifact upon check-in, so that any later unauthorized changes to the artifact can be detected easily
- Requiring that every configuration item be checked into the CM system as a baseline before it is reviewed or tested. In this way, as changes are made based on findings of the review/test, the new configuration item that results can easily be compared against the pre-review/pre-test baseline to determine whether those changes also included unintentional vulnerabilities or malicious elements
- Assigning different, non-contiguous roles with separate access rights in the CM system to development, testing, and production environments, and their corresponding personnel

Other potential enhancements to standard CM practices for software assurance include:

- Storage of a digitally signed copy of the configuration item with its configuration item progress verification report
- Reporting of differences between security aspects of previous and subsequent versions and releases
- Separation of roles/access privileges, and least privilege enforcement, for CM system users
- Flexible but carefully controlled delegation of CM administrator privileges
- No remote access, or remote access only via encrypted, authenticated interfaces
- Audit of all CM system access attempts, check-ins, check-outs, configuration changes, traceability between related components as they evolve, details of other work done

4.3.1 CM for Non-Developmental Software

The key point for non-developmental software is that you are dealing with something that is beyond your control, and that therefore affects your risk assessment. A second key is that non-developmental software follows its own life cycle, so issues and concerns with it will continue beyond your own development cycle.

The following discussion points are taken from the SwASOAR:

- The schedules and frequency of new releases, updates, and security (and non-security) patches, and response times for technical support by acquired or reused software suppliers, are beyond the control of both developers and the configuration manager
- In the case of security patches, developers can never be sure when or even if the supplier of a particular software component will release a needed security patch for a reported vulnerability that might render a selected component otherwise unacceptable for use in the software system, nor can the developer predict whether a particular security patch may invalidate the security assumptions that other components in the component-based system have about the component to be patched
- The software configuration manager should monitor vulnerability reports issued by the US-CERT, U.S. DoD's Information Assurance Vulnerability Alert (IAVA)

program, and entries in the CVE database and should download all necessary patches indicated by the vulnerability reports, then work with the developers to determine the impact of adopting those patches versus the risk of not adopting them
- Suppliers often include other features that have no vulnerability-mitigating purpose in security patches, using the patch as a chance to introduce features that will later appear in their next full release of the software. Unfortunately, these features are seldom documented or even announced when delivered with the patch, making the need for impact analysis of such features impossible to recognize
- For purposes of software configuration management control, COTS and open source software (OSS) components are like every other artifact, and so secure CM should track all fixes, patches, updates, and new released by the suppliers of COTS and OSS software components [SwASOAR 5.1.6.2]

4.3.2 Configuration Management Tools

Configuration Management tools are also known as version control systems. Standard familiar tools are CVS, Subversion, Visual SourceSafe, and SCCS. Bug tracking tools can also be used to support CM. There are several good lists of CM tools available online, including:
- Wikipedia list [Wikipedia]
- Google directory [Google]
- A newsgroup compilation [Eaton]

Some examples of secure software version control systems and repositories include:
- Mortice Kern Systems' MKS Integrity
- Oracle 10g Software Configuration Manager
- Information Systems Security Operation (ISSO) research team at Sparta, Inc. is working to move secure SCM technology forward with its prototype Secure Protected Development Repository (SPDR) [SwASOAR 5.1.6.1]

The Better SCM initiative has done a comparative review of open source CM systems. Two of the aspects compared were directly relevant to the systems' ability to support secure CM:
- Ability to assign access permissions to users, and to restrict access to the repository based on those permission assignments
- Ability to limit read and write accesses (check-ins and check-outs) to a single directory [Better SCM]

For large software/system development projects, outsourced secure CM, such as the Secure CM service offering from Source Manage, may be a desirable alternative to doing CM in-house.

4.4 Quality Assurance

“In a secure software development process, quality assurance practitioners must always have security in mind. They must be very skeptical of the accuracy and thoroughness of any security related requirements in the software’s specification. In short, they must be willing to adapt their requirements-driven mentality to introduce some risk-driven thinking into their verification processes. The quality assurance process, then, will necessarily incorporate some risk management activities focusing on ‘secure in deployment’ objectives” [SwACBK 11.9].

“Configuration management of patches must extend to security patches, to ensure that they are applied in a timely manner (both to the commercial and open source software in the software system itself and to its execution environment), and that interim risk analyses are performed to determine the impact the patch will have and to identify any conflicts that may be caused by applying the patch, particularly those impacts/conflicts with security implications, to mitigate those effects to the extent possible (which, in some cases, may mean not installing the patch because its impact/conflicts may put the software at greater risk than the vulnerability the patch is meant to address).

“All server file systems should be reviewed frequently, and extraneous files are removed, to prevent avoidable future conflicts (resulting from patching or new component releases).

“Security ‘refresh’ testing should be used to verify the software’s continued correct and secure operation any time any component or configuration parameter of its execution environment or infrastructure changes.

“The software’s security configuration should be periodically audited, to ensure that the file permissions, user account privileges, configuration settings, logging and auditing, etc., continue to be correct and to achieve their security objectives, considering any changes in the threat environment.

“Quality assurance practitioners should periodically audit the correctness of the performance of these procedures” [SwACBK 11.9].

The report, *Best Practices on Incorporating Quality Assurance into Your Software Development Life Cycle* identifies a number of security risk management measures to be included among broader QA activities in the different phases of the software development life cycle, as shown in Table 4 [Sadovsky]. The role of the QA team in these activities is to review these processes and their products, rather than to perform them.

Table 4: Security-Relevant QA Activities Throughout the SDLC

Life Cycle Phase	**QA Activities**
Requirements	Identification of acceptable levels of down time and data loss Identification of security requirements
Design	Identification and provision of countermeasures to vulnerabilities Design to reflect needs of forensics and disaster recovery activities and the ability to test for both
Implementation	Automation of nightly code scans, application and database vulnerability scans, network and configuration scans Documentation and use of manual security test procedures
Testing	Addition of penetration testing, black box testing to functional, compatibility, and regression testing
Deployment	Security training of help desk, system administration, and support staff Identification of security policy issues Establishment of schedule and procedures for system and data backups, disaster recovery
Operations/ Maintenance	Repetition of "routine" security reviews and vulnerability scans Secure change control
Decommissioning	Sanitization of media Proper disposal of hardware and software

The Development Management area of the System Development chapter of the Information Security Forum's *Standard of Good Practice* states that, "Quality assurance of key security activities should be performed during the development life cycle…to provide assurance that security requirements are defined adequately, agreed security controls are developed, and security requirements are met." Section SD1.3.2 of the Standard lists key security activities that should be subjected to quality assurance reviews and controls during the SDLC. These activities are:

- Assessing development risks (i.e., those related to running a development project, which would typically include risks associated with business requirements benefits, technology, technical performance, costing and timescale)
- Ensuring that security requirements have been defined adequately
- Ensuring that security controls identified during the risk assessment process (e.g., policies, methods, procedures, devices or programmed mechanisms intended to protect the confidentiality, integrity or availability of information) have been developed
- Determining whether security requirements are being met effectively

Section SD1.3.3 of the standard goes on to specify how QA of key security activities should be performed. This includes making sure QA starts early in the SDLC, is documented, and reviewed at all key stages of the SDLC. In Section SD1.3.4, the standard states that security risk should be minimized through the revision of project plans (including schedule, budget, and staffing) and resources whenever it is discovered that security requirements are not being effectively satisfied, to the extent that development activities should be cancelled if security requirements still cannot be satisfied after such adjustments are made” [ISF].

4.5 Reporting

The Project Management Body of Knowledge (PMBOK) defines reporting within the knowledge area of communications. The communications processes include communications planning, information distribution, performance reporting, and managing stakeholders [PMI 2004]. Project reporting can be used to communicate, but it can also be used to obscure or distort. Therefore, managers need to check the validity and completeness of the inputs received, and ensure that their own reports are accurate.

4.5.1 Status Reports

A project manager communicates in two directions: inward with the development team, and outwardly with higher-level management and customers or other stakeholders.

Differences and additions to reporting activities may include:

- The metrics used, such as those proposed by the DHS Measurement Working Group [DHS MWG 2007]
- The distribution list may include additional stakeholders/interested parties, e.g., Information Assurance group
- The level of detail reported, for example, information included in test reports must meet criteria for the assurance case
- Documentation of security-related decisions, for example, a particular version of an off-the-shelf component is (to be) used because of a known vulnerability in another version
- Reports for activities that are not normally performed, such as a particular security review
- Level of protection needed for project reports and documentation, in terms of who has access to them, who has permission to modify them, and where they are kept
- Degree of formality in status (or other) reports
- Degree of automation employed to generate data for the reports

Other communication is between the project manager and the development staff. To the degree that measurements are not automated, the manager must trust that the progress or other metrics reported by the staff are accurate.

Any project deliverable (software development artifact) also falls under the category of project communications. Software assurance concerns with these artifacts include the same issues with access, modification, etc. as any other project documentation. Perhaps even to a higher degree because of the difference between an ephemeral percent complete status as of a particular date and details of, for instance, the access control implementation.

Examples of what to report can be derived from the measurement/metrics work concerning what to measure (see Section 4.2).

Other lists of reporting requirements will come from standards and regulations by which the project is bound.

4.5.2 Lessons Learned

Compiling lessons learned is always a good practice for software engineering. Process improvement techniques rely on learning from past performance to know what to repeat and what mistakes to (try to) avoid.

As you make changes to your processes, methods and tools to incorporate software assurance, it will be wise to carefully record what does and doesn't work to improve security. Perhaps even a separate software assurance lessons learned repository should be maintained and analyses done.

5 MANAGEMENT IN THE DEVELOPMENT LIFE CYCLE

Project managers must typically perform activities to assure secure software in each stage of the software life cycle. A project developing secure software will be designed to apply various techniques. Mastering these techniques and knowing how to apply them effectively among life cycle stages requires detailed technical knowledge. The project manager cannot be expected to possess all of this technical knowledge. But he or she should know of the existence of activities in which members of the technical staff apply this knowledge. Furthermore, the project manager needs to know enough about these activities to analyze the impacts on cost, schedule, quality, and functionality of implementing them singly or in combination. This section provides a management-level qualitative overview of developing secure software, while recognizing the quantitative experience base does not yet exist for providing definitive costs and benefits of such activities.

5.1 General Concepts

A presentation of certain general concepts seems useful before providing a breakdown by life cycle stage of activities to develop and maintain secure software. This subsection provides a review of software development life cycles and an overview of certain techniques that have impacts on all software life cycle stages. These techniques consist of developing and maintaining an assurance case, formal processes, and formal methods.

5.1.1 Life Cycle Models

Activities to develop and maintain secure software occur in all stages of software life cycles. For purposes of exposition, the following life cycle stages are identified:

- Concept or requirements analysis
- Design, including both software architecture and detailed design
- Development, including coding and various testing phases
- Production
- Utilization and support, also known as operations and maintenance
- Retirement

These stages map easily to life cycle phases in a traditional waterfall software development life cycle model (Figure 3). These stages, however, occur in other life cycle models. For example, in an incremental model (Figure 4), requirements, design, development, and production occur during the development of each increment. Variations in these life cycles might include a parallel system development life cycle preceding and concluding after the software life cycle. Increments might not end up in working functionality for a given increment; for example, the first increment might end up with a prototype. A requirements phase before the first increment might define how functionality, performance, and other requirements are allocated to individual increments. A life cycle model (for example, the Cleanroom development process) might include a Statistical Process Control (SPC) approach for tuning processes for future increments based on measurements of previous increments. Some approaches (for example, Model Based Development) rely heavily on the use of automated tools to generate code from higher-level descriptions of a system. Researchers and practitioners have modified and combined various ideas in software development models not yet named, such as agile development, evolutionary development, the Rational Unified Process (RUP), and the spiral model.

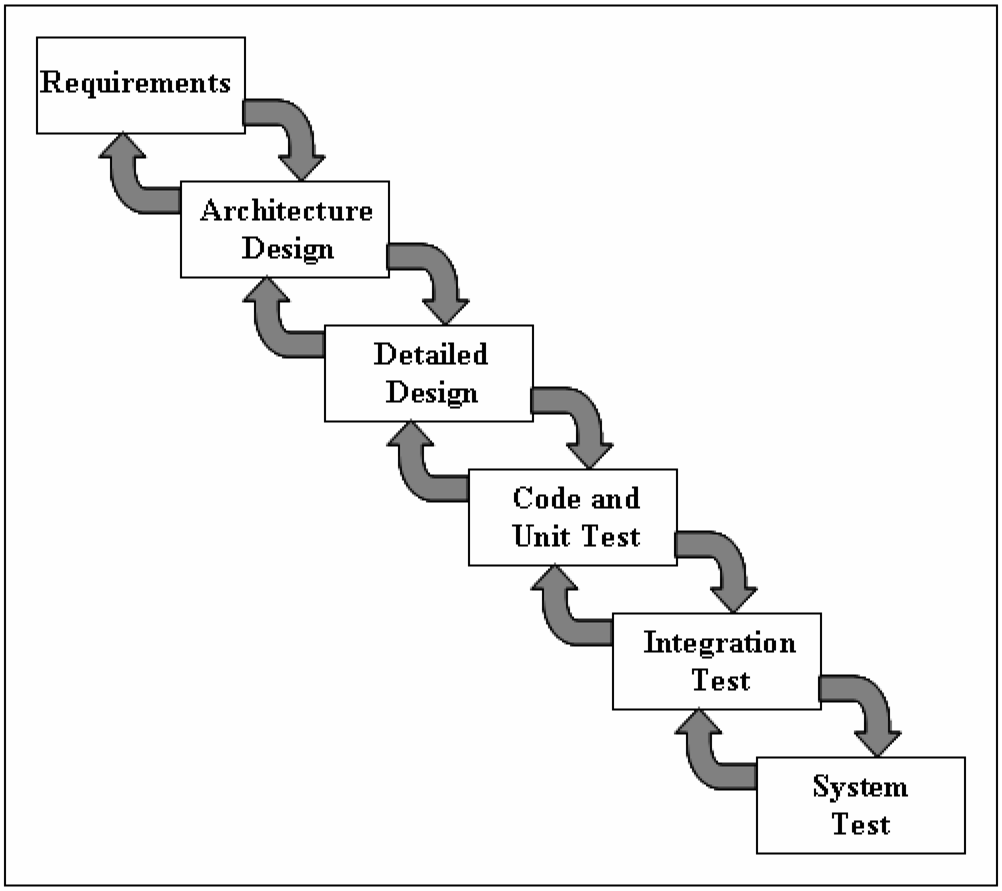


Figure 3: A Waterfall Software Development Life Cycle

The description of activities to develop and maintain secure software applies to both the waterfall life cycle models and more recent software life cycle models, as well. Some have doubted that secure software can be produced in the agile model, an approach recently gaining popularity in general software development. Agile development is closely related to eXtreme Programming (XP) and is without some of the rigidity of other models. A discussion of this controversy over agile methods can be found in [SwASOAR 5.1.8.1 and Appendix F].

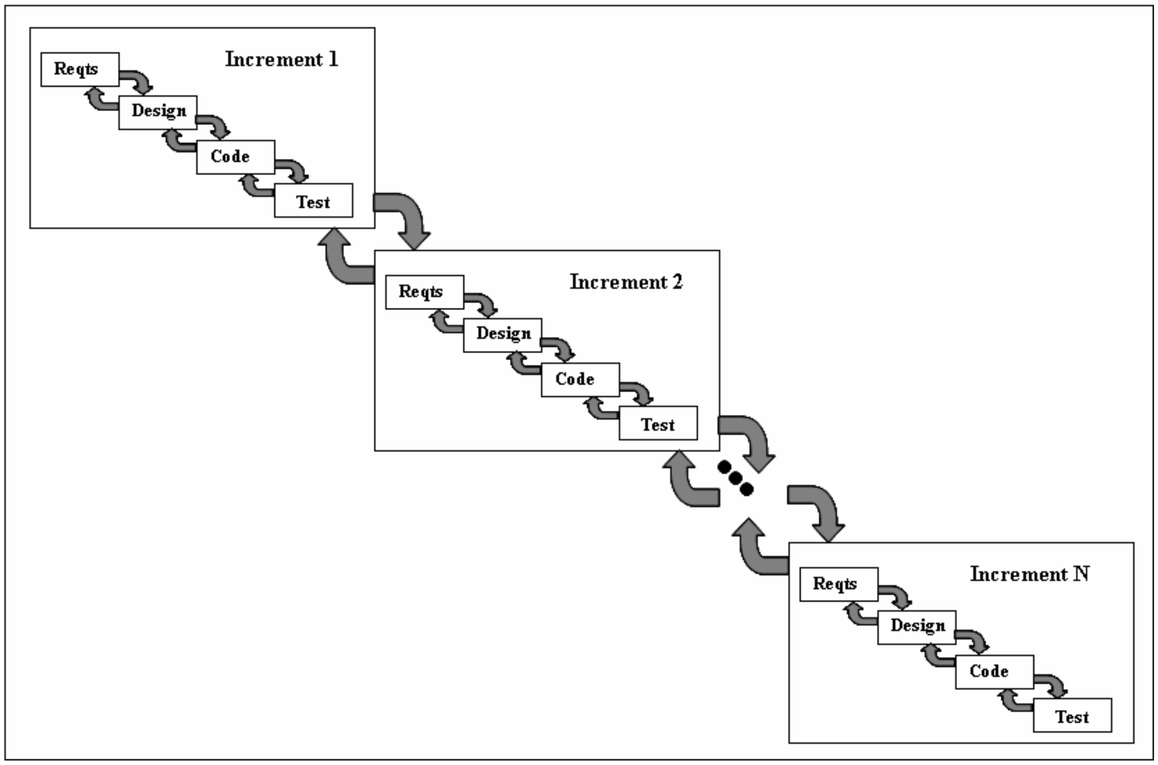


Figure 4: An Iterative and Incremental Development Life Cycle

5.1.2 Assurance Case

An assurance case can be a justification for confidence that a software or software-intensive system is secure. One definition of an assurance case is that it is:

> "a documented body of evidence that provides a convincing and valid argument that a specified set of critical claims regarding a system's properties are adequately justified for a given application in a given environment" [Ankrum 2006].

It is easier to develop an assurance case at the same time as a software system, rather than after development has been completed.

Structuring assurance cases so they can be understood is a challenge. Some standards have been developed for assurance cases:

- United Kingdom (UK) Ministry of Defense (MOD) sponsor of SafSec, developed by Praxis High Integrity Systems [SafSec]
- ISO/IEC and IEEE 15026, System and Software Assurance [IEEE 15026]
- Application-area specific (e.g., airborne systems, medical devices)

Also of interest, some workshops on software security assurance cases were hosted in 2005 by the CMU SEI and in 2006 by the European Commission's Joint Research Center (JRC).

Some software systems must undergo Certification and Accreditation (C&A) before being permitted to operate in certain operational environments. In particular, C&A requirements have been imposed on systems in the Information Assurance (IA) application domain. Security C&A processes typically mandate the development of an assurance case. Prominent C&A standards include:

- ISO/IEC 15408, *Information Technology – Security Techniques – Evaluation Criteria for IT Security*, commonly known as the Common Criteria. IA products that have been certified against the Common Criteria have Protection Profiles [ISO/IEC 15408]
- National Institute of Standards and Technology (NIST) Cryptographic Module Verification Program (CMVP) certifies products against Federal Information Processing Standards (FIPS) 140 requirements for software for cryptography
- The DoD Information Assurance Certification and Accreditation Process (DIACAP, DODI 8510, replacing the DoD Information Technology Security Certification & Accreditation Process (DITSCAP, DODI 5200.40)), Federal Information Security Management Act (FISMA), and Protecting Compartmented Information within Information Systems (DCID 6/3) define Federal, DoD, and intelligence community certification and accreditation processes

Table 5 lists elements of an assurance case at each Common Criteria level. This example of elements of an assurance case is provided as an example to help you see what goes into one. For more on assurance cases, see the *Acquisition Management Guide for Software Assurance*, [DHS AWG 2006] and [SwASOAR 6.1.9.1].

Table 5: Evaluation Assurance Levels in the Common Criteria

EAL	Elements
1: Functionally Tested	• Analysis of security functions using a functional and interface specification and guidance documentation • Independent testing of security functions of the Target Of Evaluation (TOE)
2: Structurally Tested	• Analysis of security functions using a functional and interface specification, guidance documentation, and high-level design • Independent testing of TOE security functions • Evidence of developer testing based on the functional specification • Selective independent confirmation of the developer test results • Strength of function analysis • Evidence of a developer search for obvious vulnerabilities • A configuration list for the TOE • Evidence of secure delivery procedures
3: Methodically Tested and Checked	• Analysis of security functions using a functional and interface specification, guidance documentation, and high-level design • Independent testing of TOE security functions • Evidence of developer testing based on the functional specification and high-level design • Selective independent confirmation of the developer test results

	• Strength of function analysis • Evidence of a developer search for obvious vulnerabilities • Development environment controls • CM • Evidence of secure delivery procedures
4: Methodically Designed, Tested, and Reviewed	• Analysis of security functions using a functional and complete interface specification, guidance documentation, the high-level and low-level design, and a subset of the implementation • Informal model of the security policy • Independent testing of TOE security functions • Evidence of developer testing based on the functional specification and high-level design • Selective independent confirmation of the developer test results • Strength of function analysis • Evidence of a developer search for vulnerabilities • Independent vulnerability analysis demonstrating resistance to penetration attackers with a low attack potential • Development environment controls • CM, including automation • Evidence of secure delivery procedures
5: Semi-formally Designed and Tested	• Analysis of security functions using a functional and complete interface specification, guidance documentation, the high-level and low-level design, and all of the implementation • Formal model of the security policy • Semiformal presentation of the functional specification and high-level design, and a semiformal demonstration of correspondence between them • A modular design • Independent testing of TOE security functions • Evidence of developer testing based on the functional specification and high-level and low-level designs • Selective independent confirmation of the developer test results • Strength of function analysis • Evidence of a developer search for vulnerabilities • Independent vulnerability analysis demonstrating resistance to penetration attackers with a moderate attack potential • Validation of the developer's covert channel analysis • Structured development process • Development environment controls • Comprehensive CM, including automation • Evidence of secure delivery procedures
6: Semi-formally Verified Design and Tested	• Analysis of security functions using a functional and complete interface specification, guidance documentation, the high-level and low-level design, and a structured presentation of the

	implementation • Formal model of the security policy • Semiformal presentation of the functional specification, high-level design and low-level design, and a semiformal demonstration of correspondence between them • A modular and layered design • Independent testing of TOE security functions • Evidence of developer testing based on the functional specification and high-level and low-level designs • Selective independent confirmation of the developer test results • Strength of function analysis • Evidence of a developer search for vulnerabilities • Independent vulnerability analysis demonstrating resistance to penetration attackers with a high attack potential • Validation of the developer's systematic covert channel analysis • Structured development process • Development environment controls • Comprehensive CM, including complete automation • Evidence of secure delivery procedures
7: Formally Verified Design and Tested	• Analysis of security functions using a functional and complete interface specification, guidance documentation, the high-level and low-level design, and a structured presentation of the implementation • Formal model of the security policy • Formal presentation of the functional specification and high-level design, a semiformal presentation of the low-level design, and formal and semiformal demonstration of correspondence between them, as appropriate • A modular, layered and simple design • Independent testing of TOE security functions • Evidence of developer testing based on the functional specification, high-level and low-level designs, and implementation representation • Complete independent confirmation of the developer test results • Strength of function analysis • Evidence of a developer search for vulnerabilities • Independent vulnerability analysis demonstrating resistance to penetration attackers with a high attack potential • Validation of the developer's systematic covert channel analysis • Structured development process • Development environment controls • Comprehensive CM, including complete automation • Evidence of secure delivery procedures

5.1.3 Formal Processes

Typically, software assurance will require the introduction of more formalism, in two senses, into the software life cycle. In one sense, software development is formal if processes are defined and documented. This sense is commonly used in the Software Process Improvement (SPI) community. For example, certification for Capability Maturity Model (CMM) at the second level requires documented processes for Configuration Management (CM), Quality Assurance (QA), subcontract management, project tracking and oversight, project planning, and requirements management.

Using formal development processes, in this sense, increases organizational capabilities. For example, it can enable your organization to bid on additional contracts. These benefits come at costs, impacting high-level management and company policies, as well as individual projects. For example, your organization will most likely have a Software Engineering Process Group (SEPG), will maintain process definitions, and will provide training material. The management organization for a software development or maintenance project will typically include a reporting chain outside the project for a Quality Assurance group and a Configuration Management control board within the project. Material produced by the project will include a Software Development Plan. Maintaining higher maturity processes has additional implications for an organization.

The introduction of software assurance into formal development and maintenance processes can be considered by an SEPG. The SwASOAR identifies life cycle methodologies for introducing security considerations that have been successfully used on more than one commercial project or academic pilot:

- Microsoft Trustworthy Computing Security Development Life cycle (SDL)
- Oracle Software Security Assurance Process
- Comprehensive, Lightweight Application Security Process (CLASP), as architected by McAfee, Incorporated
- Seven Touchpoints for Software Security, as described by Gary McGraw
- Team Software Process for Secure Software Development (TSP-Secure), from the Software Engineering Institute (SEI) at Carnegie Mellon University

More details about these methodologies and other methodologies currently being researched are available in [SwASOAR 5.1.8.2]. Processes to be adopted are constrained by the assurance case you want to produce and whatever Certification and Accreditation, if any, the product must undergo.

5.1.4 Formal Methods

Formalism, in the second sense, is that as used within formal methods. Formal methods incorporate "mathematically based techniques for the specification, development, and verification of software." That is, formal methods allow the development of mathematical proofs, in principle, that an implementation satisfies a specification. Mathematical proofs, in this sense, describe syntactical manipulations of formulae by well-defined inference rules, as such proofs are studied in mathematical logic.

Formal methods are mandated to be part of an assurance case in the Common Criteria for certain assurance levels.. They can increase the likelihood of functional correctness and help provide structure for reviews. While still expensive, formal methods, including automated tools, are most mature for specifying and verifying functional properties of a system. They are harder to apply to non-functional properties. Safety and security are the most developed non-functional areas for the application of formal methods.

The usage of formal methods in the software development life cycle varies with the level of assurance you want to attain. At a low level, one might apply formal methods only to a requirements specification. The architecture design and lower-level designs and implementations would be documented, at best, semi-formally. Correspondences between the architecture design and the requirements specification would only be semi-formally demonstrated. At a much higher assurance level, one would formally describe requirements specifications, the architecture design, the detailed design, and a presentation of the implementation. Each level would be accompanied by a formal verification that it implements the design or specification at the next higher level of abstraction. Also, you may apply different degrees of formality to different parts of a project, depending on the criticality of some portion, or as a result of prior risk analysis.

Activities available in implementing formal methods and general limitations of formal methods are explained in [SwASOAR 5.1.2].

5.2 Life Cycle Stage: Concept

In the concept or software requirements phase, analysts elicit requirements for secure software. Requirements for secure software, in the sense of this report, are distinguished from requirements for security functionality.

Requirements for security functionality include functions implementing security policy. These security functionality requirements include functions for access control, identification, authentication and authorization, encryption, decryption, and key management. Security functionality requirements are otherwise known as “security service requirements”. They prevent violations of security properties of a system and of information processed by the system. Violations include Denial of Service (DoS) and unauthorized access, modification, and disclosure of information.

Requirements for secure software, on the other hand, begin as non-functional requirements. These requirements ensure that the system will remain dependable even when the system is threatened. They are often directed towards reducing or eliminating vulnerabilities in software. They include requirements on processes, on the Software Development Plan (SDP), and on project management. Microsoft calls these requirements for secure software “Security Objectives”. They are goals and constraints that affect the Confidentiality, Integrity, and Availability (CIA) of the data and application.

Analysts perform some new activities in the requirements stage to specify requirements for secure software. Among these activities are risk assessment and threat analysis. These activities result in the development and maintenance of additional artifacts. Some of these artifacts will be associated with the specific methods, techniques, or tools adopted in analyzing risks and threats. Other artifacts will be associated with the assurance case, which will typically be started during requirements. Activities associated with secure software requirements will also introduce new elements into requirements documents you may already be producing. You may already specify use cases. Secure software may introduce abuse/misuse cases. Secure software requirements may also result in greater emphasis on exception handling in requirements documents.

Requirements for secure software may impact how other requirements are recorded. For example, achieving a high assurance level may require that a formal specification be used for your system, while your prior practice was not to adopt formal methods.

Often requirements methodologies are accompanied by defined processes and criteria for reviews. These processes and criteria can be expanded. For example, checklists can contain additional entries based on requirements for software assurance needs.

For more on specifying and analyzing requirements for secure software, especially for overviews of methods, techniques, and tools, see [SwASOAR 5.2], [SSLC G.2], and [SwACBK 5].

5.3 Life Cycle Stage: Design

Architecture design, also known as preliminary and high-level design, identifies components and allocates requirements to them. This design stage often defines interfaces between components and the data that flows among them. Detailed design defines control, algorithms, and data structures for each component in a system. Detailed design also decomposes the data flowing among components.

During design stages, especially architecture design, attack models developed during requirements are refined. The security properties of the architecture are modeled, and designers compare these properties with requirements and the security policy defined during the concept stage. As with the concept or requirements stage, producing secure software can lead to the greater adoption of formal methods.

Researchers have developed a number of canned solutions for design problems. These canned solutions are known as “design patterns”, and have become increasingly adopted by practitioners. Design patterns have been developed to address both information assurance and software assurance, at both the architecture and lower levels. Security design patterns are probably most mature for Web applications.

Verification is performed at each later stage to demonstrate that later artifacts implement earlier artifacts. A requirements traceability analysis, for example, is common.

Verification conducted during design phases, to assure secure software, demonstrates that the design:

- Fulfills the specified requirements
- Does not include any unintended functions
- Exhibits required properties, such as security properties
- Does not include exploitable weaknesses.

Both designs and assurance cases are analyzed in piecemeal in-process reviews and in total in larger system reviews. The Preliminary Design Review (PDR) and the Critical Design Review (CDR) are examples of system reviews. The System Requirements Review (SRR) and the Test Readiness Review (TRR) are examples of reviews in other stages of the life cycle. Often, checklists are used as guidance in both in-process reviews and major system reviews. Definitions of review processes and checklists can be updated to incorporate secure software concerns. These updates may add personnel with new roles into these reviews.

For more about designing secure software, see [SwASOAR 5.3], [SSLC G.3], and [SwACBK 6].

5.4 Life Cycle Stage: Development

Some well-known vulnerabilities, such as allowing buffer overflows, can be introduced in implementation. Requirements introduced in earlier stage impose constraints on the development process intended to avoid the introduction of faults, including faults that manifest as vulnerabilities, and to detect any faults introduced. A design intended to produce secure software will have introduced components to remove exposure to vulnerabilities. For example, there may be components to prevent exposure of vulnerabilities in COTS, and components for input validation and output encoding to prevent SQL Injection and Cross-Site Scripting attacks. Care and management attention are needed in the execution of later life cycle stages to produce secure software.

5.4.1 Coding

Coding has the goal of developing executable software implementing the design. Security issues in coding are addressed through, for example:

- Choice of languages
- Choice and use of compiler, library, and development and execution libraries
- Coding rules and conventions
- Rules and conventions for comments.
- The documentation of code, constructs, and implementation decisions with security-implications
- How non-developmental software is handled
- Implementations of filters and wrappers

Many organizations with rigorous development processes have adopted some type of code reviews or structured inspections. These inspections can address issues of secure software by introducing such concerns into process definition and by updating checklists typically used in structured inspections.

More on coding and security is in [SwASOAR 5.4], [SSLC G.4-G.5], and [SwACBK 7].

5.4.2 Testing

Security testing assesses software interactions with external entities and the behavior and interactions of the components comprising the software. External entities include human users, software in other systems, and other entities in the system environment. Security testing attempts to verify that the software:

- Exhibits predictable and secure behavior
- Exposes no vulnerabilities or weaknesses (even better would be software that contains no vulnerabilities or weaknesses, exposed or not)
- Maintains a secure state (often through error and exception handling capabilities) when confronted by attack patterns or intentional faults
- Satisfies all specified and implicit non-functional security requirements
- Does not violate any specified security constraints
- Provides executables, such as runtime-interpretable source code and byte code, that have been obscured or obfuscated as consistent with other constraints as possible to prevent reverse engineering

Testing can be categorized in several ways. Generally, security testing can be performed in multiple categories. One categorization is by life cycle testing phases. Coders may conduct informal unit tests. Integration testing occurs when components are incrementally combined and tested. Several types of system testing can be performed, for example:

- To demonstrate that functional requirements are met
- To measure reliability, with random sampling from a specified operational profile
- To measure performance under controlled loads

Various toolsets are available for testing. These include test harnesses, Model-Based Testing tools to generate tests from user models, implementations of software reliability models, and regression testing tools to systematically rerun tests.

How tests are developed and how progress is measured varies with the type of testing. These variations provide other categorizations. In black box testing, tests are based on specifications. In white box testing (also known as glass box testing), tests are also based on source code. Test coverage metrics in the former case measure how many requirements have been tested or how much of the input domain, partitioned on the basis of requirements, has been tested. Test coverage metrics for the latter case measure, for example, the number of lines, conditions, or paths tested. Metrics include the number of successful and failed tests.

Another categorization is provided by who performs testing. The development organization can run tests, and an independent entity can perform Independent Verification and Validation (IV&V). Often an IV&V organization will also carry out static analyses, perhaps based on documentation and code that are available before the system is executable.

Security testing includes static analyses and reviews conducted throughout the life cycle, as well as dynamic analyses in all testing phases. A variety of both white box and black box techniques are available for security testing. Both developer and independent organizations can conduct security testing. Security testing differs from testing that the software's security functions are correctly implemented. It also differs from testing interoperability and usability. Penetration testing, in which "white-hat" hackers or "tiger teams" attempt to exploit vulnerabilities in a system is a particularly colorful form of security testing, but is only one technique of many. Other techniques explore how a system responds to invalid data, how a system functions in anomalous states, and whether security-related properties can be demonstrated for a system design or implementation.

As with other forms of testing, security testing activities begin early in most life cycles. For example, one might include a security test plan in early phases of the software life cycle. Just as models of the user produced during requirements relate to system testing, so models of the threat environment also begun during requirements relate to later security testing.

For more on software security testing see [SwASOAR 5.5], [SSLC G.6], and [SwACBK 8]. One can also outsource security testing. Some companies offer such testing as a commercial product.

5.5 Life Cycle Stage: Production

In production, the software is distributed or deployed. Secure software will have removed characteristics of development software that represent potential vulnerabilities, and a secure installation configuration will have been defined. As noted in Section 5.1.2, an assurance case may require evidence that the software is examined for vulnerabilities, that development environment controls are in place, that Configuration Management (CM) is adopted, and that secure delivery practices are used.

5.5.1 Preparing for Distribution/Deployment

Often, the system during development and test will differ from the system one wants to distribute. Instrumentation may have been added. A debugging mode might exist. Backdoors might ease developer access. Comments might describe confidential information. Unused functions might reflect development history. Test data may include default accounts and roles. Code might depend on characteristics of the development library. All of these possible characteristics of a development system reflect potential vulnerabilities. Preparation for distribution or deployment of a secure system would review the software and remove unnecessary code, comments, and data.

Preparation for distribution and deployment might also include instructions on how to install a system as a “secure configuration”. For example, guidance is available for secure configurations of COTS and popular open source products in common environments.

Another aspect of distribution and deployment is setting up a help desk or other capability for user support.

More information on preparing for distribution and deployment can be found in [SwASOAR 5.6] and [SSLC G.7].

5.5.2 Trusted Distribution

“Trusted distribution, in conjunction with configuration management, provides assurance that the [system] software, firmware, and hardware, both original and updates, are received by a customer site exactly as specified by the vendor’s master copy. Trusted distribution also ensures that [system] copies sent from other than legitimate parties are detected” (from the Dark Lavender book) [NCSC TG-008].

Whether or not you are mandated to use processes conforming to one or more of the books in the rainbow series, you may find their approaches and concepts helpful.

If software is to undergo trusted distribution, strong authentication must be used for installation and configuration. A default password should not exist in the software, including its installation routine. A separate password should be used for each installation, and that password should be distributed by a separate distribution path than is used for distributing the software. Configuration interfaces should be clear and secure. Configuration interfaces should make the results of administration actions clear, and defaults should allow access to the configuration interface to no role other than the administrator. Default privileges for non-administrator roles should be execute-only for the software’s executable files.

For more information on trusted distribution, see *A Guide to Understanding Trusted Distribution in Trusted Systems* [NCSC TG-008] and [SSLC G.7.10].

5.6 Life Cycle Stage: Utilization and Support

This section provides a high-level discussion of post-deployment software security concerns. Life cycle phases considered consist of Operations and Maintenance and Retirement.

In many organizations, the development manager’s responsibility does not extend into defining and assuring secure processes and policies for security in these phases.

Documentation of the product can include definition of secure configurations and operating and users manuals. Many of the issues discussed in this section would be addressed by an operations manager, perhaps in concert with a Designated Approving Authority (DAA), if the system must be Certified and Accredited.

5.6.1 Operations and Maintenance

Operations provide many areas in which policies can be developed. The length, content, and duration of passwords might be defined in such a policy. Some systems might require other forms of authentication. What training must users undergo, and how often? Must users in certain roles undergo background checks?

Tracking vulnerabilities and installing patches is another area in which a policy needs to be defined for operations. Such a policy might include decisions on who would perform such tracking. Such tracking might incorporate the tracking of vulnerabilities and patches for COTS integrated in your system. Decisions need to be made on how and when patches would be installed. Some patches will address non-security concerns (for example, functionality). Analysis of the risks, costs, and benefits associated with installing patches and releases can take into account the potential of introducing new security vulnerabilities. Should you perform a periodic analysis of the desirability of replacing or redeveloping selected system components on grounds of a security vulnerability?

The maintenance of security-related aspects of the development process in operations is another concern. For example, environmental and personnel controls need to be maintained. A CM process supports a disciplined process for addressing faults and vulnerabilities. Perhaps, the assurance case developed along with the system should be maintained and updated during operations. Decisions on some of these aspects will be driven by whether or not you have C&A requirements. Typically, a system's C&A expires after some defined period and must be renewed.

A security policy can define time between periodic tests of various types. For example, one type might be security testing by a tiger team. Another type might include a test of backup and restoration procedures.

For further information on software security considerations in system maintenance, see [SSLC G.8], [SwACBK 12.5], and the NIST *Security Metrics Guide for Information Technology Systems* [NIST 2003].

5.7 Life Cycle Stage: Retirement

Typically, a system is not retired without the organization that operates the system migrating its capabilities into some new system. Risks and issues in managing security can arise in both migration and retirement. These risks and issues can be documented in the transition strategy. They can include assurance that system capabilities are provided securely during the transition.

Whether you are mandated to manage security risks and issues might depend on whether either the old system, the new system, or both are accredited under some Certification and Accreditation process. If they are, at least one of the systems has a Designated Approving Authority to oversee the transition process.

The Software Assurance Common Body of Knowledge suggests that one should define processes for a secure transition and to ensure security of data and the new system after transition. These processes can include tests, inspections, and verification and validation procedure of both the transition process and of the new software system after the transition has been made. These processes can also include documentation of the results of the transition, including tests, inspections, and verification and validation.

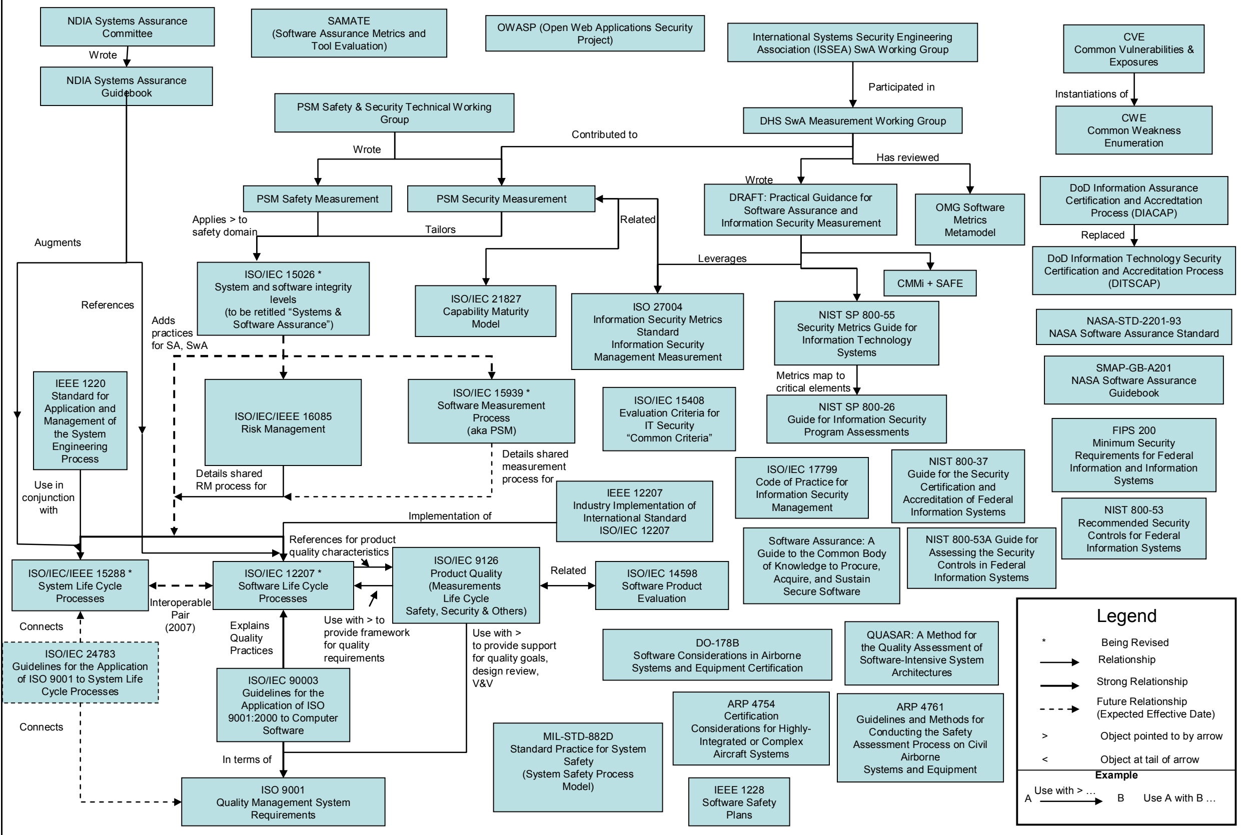
For more about security considerations in retiring a software-intensive system, see [SwACBK 12.5.9].

6 STANDARDS FOR SECURE SOFTWARE ENGINEERING

There are a number of standards organizations that have developed or are developing standards in security-related topics, these include.

- IEEE-CS Software & Systems Engineering Standards Committee (S2ESC)
- IEC/ISO Joint Technical Committee (JTC) 1 Subcommittees (SC) 7, 27 and 22
- Object Management Group (OMG) Software Assurance(SwA) Special Interest Group (SIG)
- Committee on National Security Systems (CNSS)
- National Institute of Standards and Technology (NIST)
- Open Web Application Security Project (OWASP)

A sense of the complexity of the standards world and some of relationships among standards and standards organizations can be gleaned from Figure 5 [DHS].



NDIA Systems Assurance Committee
Wrote
NDIA Systems Assurance Guidebook
SAMATE (Software Assurance Metrics and Tool Evaluation)
OWASP (Open Web Applications Security Project)
International Systems Security Engineering Association (ISSEA) SwA Working Group
CVE Common Vulnerabilities & Exposures
Participated in
Instantiations of
PSM Safety & Security Technical Working Group
DHS SwA Measurement Working Group
CWE Common Weakness Enumeration
Contributed to
Wrote
Has reviewed
Wrote
PSM Safety Measurement
PSM Security Measurement
DRAFT: Practical Guidance for Software Assurance and Information Security Measurement
OMG Software Metrics Metamodel
DoD Information Assurance Certification and Accredtation Process (DIACAP)
Replaced
Applies > to safety domain
Tailors
Related
Augments
Leverages
DoD Information Technology Security Certification and Accreditation Process (DITSCAP)
CMMi + SAFE
ISO/IEC 15026 * System and software integrity levels (to be retitled "Systems & Software Assurance")
ISO/IEC 21827 Capability Maturity Model
ISO 27004 Information Security Metrics Standard Information Security Management Measurement
NIST SP 800-55 Security Metrics Guide for Information Technology Systems
NASA-STD-2201-93 NASA Software Assurance Standard
References
Adds practices for SA, SwA
SMAP-GB-A201 NASA Software Assurance Guidebook
IEEE 1220 Standard for Application and Management of the System Engineering Process
ISO/IEC/IEEE 16085 Risk Management
ISO/IEC 15939 * Software Measurement Process (aka PSM)
ISO/IEC 15408 Evaluation Criteria for IT Security "Common Criteria"
Metrics map to critical elements
NIST SP 800-26 Guide for Information Security Program Assessments
FIPS 200 Minimum Security Requirements for Federal Information and Information Systems
Details shared measurement process for
Details shared RM process for
ISO/IEC 17799 Code of Practice for Information Security Management
NIST 800-37 Guide for the Security Certification and Accreditation of Federal Information Systems
Use in conjunction with
IEEE 12207 Industry Implementation of International Standard ISO/IEC 12207
NIST 800-53 Recommended Security Controls for Federal Information Systems
Implementation of
References for product quality characteristics
Software Assurance: A Guide to the Common Body of Knowledge to Procure, Acquire, and Sustain Secure Software
NIST 800-53A Guide for Assessing the Security Controls in Federal Information Systems
ISO/IEC/IEEE 15288 * System Life Cycle Processes
ISO/IEC 12207 * Software Life Cycle Processes
ISO/IEC 9126 Product Quality (Measurements Life Cycle Safety, Security & Others)
Related
ISO/IEC 14598 Software Product Evaluation
Interoperable Pair (2007)
Connects
Explains Quality Practices
Use with > to provide framework for quality requirements
Use with > to provide support for quality goals, design review, V&V
ISO/IEC 24783 Guidelines for the Application of ISO 9001 to System Life Cycle Processes
DO-178B Software Considerations in Airborne Systems and Equipment Certification
QUASAR: A Method for the Quality Assessment of Software-Intensive System Architectures
ISO/IEC 90003 Guidelines for the Application of ISO 9001:2000 to Computer Software
ARP 4754 Certification Considerations for Highly-Integrated or Complex Aircraft Systems
ARP 4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment
Connects
MIL-STD-882D Standard Practice for System Safety (System Safety Process Model)
In terms of
ISO 9001 Quality Management System Requirements
IEEE 1228 Software Safety Plans
Legend
* Being Revised
Relationship
Strong Relationship
Future Relationship (Expected Effective Date)
> Object pointed to by arrow
< Object at tail of arrow
Example
A Use with > ... B Use A with B ...

Figure 5: Standards Activities Map

The DHS SwA Forum, although it does not have a separate standards working group, has set collaboration with federal agencies, standards bodies, industry and academia as a goal within its process focus. The SwA Forum plans include providing draft guidance for specifying assurance arguments from which to base claims about the safety, security and dependability of software, and providing recommended changes to national and international standards on software testing and software assurance via ongoing work and liaison with IEEE CS S2ESC, ISO/IEC JTC1 SC7/SC27/SC22, OMG, CNSS, and the NIST Working Group on Standards.

A joint effort between the DoD and the FAA to define safety and security extensions to the CMMI/iCMM has resulted in a draft report that defines sixteen Safety and Security Application Area practices. Although not a standard, per se, it provides the same type of guidance for organizations that follow CMM development processes [Ibrahim 2004].

Current work in standards for software assurance is discussed in Section 6.3 of the SwASOAR [SwASOAR 6.3].

Specific standards.
The following is a sampling of standards that have some applicability to secure software development.

IEEE Standard 1074-2006

> The revision of the IEEE Std. 1074-1997, *Developing Software Project Life Cycle Processes*, supports appropriate prioritization of security and building of appropriate levels of security controls into software and systems. The new standard accomplishes this by adding a small number of security activities to the SDLC defined in the earlier version of the standard.

ISO/IEC 15026, System and Software Engineering – System and Software Assurance

> Describes additional techniques needed for high-integrity systems. Currently, not process-oriented, but is being repositioned. Being revised to add three core processes for planning, establishing, and validating assurance cases for software-based systems, in order to establish the necessary level of assurance of software developed via the systems and software engineering life cycle processes defined in ISO/IEC 12207 and ISO/IEC 15288.

ISO/IEC 12207:1995, Software Life Cycle Processes

> 17 processes spanning the life cycle of a software product or service. The standard is somewhat prescriptive in defining a minimum level of responsible practice. Describes processes meeting the needs of organizational process definition.

ISO/IEC 12207:Amendment 1

> Describes processes to meet the needs of process assessment and improvement.

ISO/IEC 15288, System Life Cycle Processes

> Establishes a common framework for describing the life cycle of systems. 25 processes spanning the life cycle of a system. This standard is primarily descriptive.

ISO/IEC 16085, Risk Management Process

ISO/IEC 15939, Measurement Process

NASA-STD-8739.8 SOFTWARE ASSURANCE STANDARD, July 28, 2004.

IEEE 1220:
Provides a standard for managing systems engineering

ISO/IEC 21827, Systems Security Engineering Capability Maturity Model
A process reference model for improving and assessing the maturity of the security engineering processes used to produce information security products, trusted systems, and security capabilities in information systems

7 RESOURCES

The SwASOAR includes an extensive catalogue of resources for software assurance [SwASOAR 7], as well as an appendix listing software assurance-related programs and research at colleges and universities around the world [SwASOAR H]. Rather than repeating that work, this section highlights those resources that are most relevant to management from both Section 7 and Appendix H.

The most applicable on-line source for information is the Build Security In portal [BSI], as its charter and focus are exactly in line with the subject of this SOAR (and even more so, with the SwASOAR). The BSI site is hosted by the US-CERT at https://buildsecurityin.us-cert.gov. Project Management is one of the content areas under the *Best Practices* section. Many of the other content areas have articles and information relevant to managing for secure software, including:

- Acquisition
- Assurance Cases
- Business Case Models
- Deployment and Operations
- Governance & Management
- SDLC Process
- Measurement
- Requirements Engineering
- Risk Management
- System Strategies
- Training and Awareness

The DHS SwA Forum is another program which is closely aligned with the subject of this report. The DHS SwA Forum program and working groups are described in Section 6.1.9.1 of the SwASOAR. Although access to in-process materials from the working groups is restricted to participants, results and publications from the working groups are available from the BSI portal, in the *Additional Resources* area under *DHS-Related*: https://buildsecurityin.us-cert.gov/daisy/bsi/resources/dhs.html. The meetings are announced at https://buildsecurityin.us-cert.gov/daisy/bsi/events.html.

Other organizations that do work or research in software engineering and software technology have begun including security-related topics and issues in their work. For example, both the STSC in the monthly journal *CrossTalk*, and the DACS in the quarterly newsletter *Software Tech News* have published theme issues on software assurance. These organizations include:

- Software Technology Support Center (STSC), http://www.stsc.hill.af.mil/
- Data & Analysis Center for Software (DACS), http://iac.dtic.mil/dacs/
- Software Engineering Institute (SEI), http:// www.sei.cmu.edu
- NASA Software Engineering Lab, http://sel.gsfc.nasa.gov/

From the section listing on-line resources in the SwASOAR, others that are likely to contain information of use to project managers include:

- Open Web Application Security Project (OWASP) portal, http://www.owasp.org
- Cigital Inc. Resources, http://www.cigital.com/resources
- SysAdmin, Audit, Networking, and Security (SANS) Reading Room, http://www.sans.org/reading_room [SwASOAR 7.1.1.1]

Among the books listed, particularly appropriate ones include:

- Haralambos Mouratidis and Paolo Giorgini, eds., *Integrating Security and Software Engineering: Advances and Future Visions*, Idea Group Publishing, 2007
- Gary McGraw, *Software Security: Building Security In*, Addison-Wesley, 2006
- Ross J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, John Wiley & Sons, 2001
- Gary McGraw and John Viega, *Building Secure Software: How to Avoid Security Problems the Right Way*, Addison-Wesley Professional, 2001 [SwASOAR 7.1.2]

Relevant journals and magazines are:

- Secure Software Engineering Journal, http://www.secure-software-engineering.com
- CrossTalk: The Journal of Defense Software Engineering, http://www.stsc.hill.af.mil/crosstalk
- IEEE Security and Privacy, http://www.computer.org/portal/site/security [SwASOAR 7.1.3]

Recurring conferences and workshops include:

- OWASP Application Security Conference, http://www.owasp.org/index.php/Category:OWASP_AppSec_Conference
- Software Security Summit, http://www.s-3con.com
- International Workshop on Software Engineering for Secure Systems, http://homes.dico.unimi.it/~monga/sess07.html
- International Workshop on Secure Software Engineering, http://www.ares-conference.eu/conf/index.php?option=com_content&task=view&id=26&Itemid=3
- IEEE International Workshop on Security in Software Engineering, http://conferences.computer.org/compsac/2007/workshops/IWSSE.html
- Annual Computer Security Applications Conference, http://www.acsac.org [SwASOAR 7.1.4]

Appendix H of the SwASOAR lists software security research in academic institutions. Table 6 provides a subset of that list, showing programs or projects relevant to management topics, including web addresses, where available.

Table 6: Resources in Academia

Athens University of Economics and Business	SENSE - Software Engineering and Security	http://istlab.dmst.aueb.gr/content/groups/g_sensedetails.html
Auburn State University	Software Process for Secure Software Development	http://www.eng.auburn.edu/users/hamilton/security/Information_Assurance_Laboratory_Research_Areas_Dec_2003.html
Ball State University	Measuring the Effect of Software Design on Software Security	http://www.serc.net/web/research/index.asp
Carnegie Mellon University	Team Software Process (TSP) Secure and Secure Quality Requirements Engineering (SQUARE)	http://www.sei.cmu.edu/tsp/tsp-security.html, http://www.cert.org/sse/square.html
Catholic University of Leuven	Measuring framework for software security properties	
City University (London)	International Working Group on Assurance Cases (including software security assurance cases)	http://www.csr.city.ac.uk/AssuranceCases
Fraunhofer Institute for Experimental Software Engineering (Kaiserslautern)	Development Coaching: during all process phases in the development of safety- or security-critical systems	
German Research Center for Artificial Intelligence (Transfer Center)	Secure Software group	http://www.dfki.de/siso [in German]
Iowa State University	Secure configuration management	
North Dakota State University	Secure Software Engineering: A Threat-Driven Approach	http://cs.ndsu.edu/~dxu/research/security.html
Purdue University	Secure Programming Educational Material	http://projects.cerias.purdue.edu/secprog
Queen's University (Kingston, ON)	Unifying Software Engineering and Security Engineering	

Technical University of Munich	Software Security	http://www.model.informatik.tu-muenchen.de/research/projects/detail/index.php?id=projects.detail&arg=18
Technical University of Munich	Working Group on Security and Safety in Software Engineering	http://www4.in.tum.de/~secse/group.html
University of California at Berkeley	Software Security Project	http://www.cs.berkeley.edu/~daw/research/ss
University of Cambridge	Economics of information and software security	http://www.cl.cam.ac.uk/~rja14/econsec.html
University of Duisburg Essen	Development of secure software	
University of Lapland/Institute for Legal Informatics	Regulating Secure Software Development	http://www.ulapland.fi/?newsid=6440&deptid=11589&showmodul=47&languageid=4&news=1
University of Mannheim	Hardware-Software interactions and their Security issues: Dependability (including security) Metrics	http://pi1.informatik.uni-mannheim.de/index.php?pagecontent=site/Research.menu/Projects.page/Dependability%20Metrics.page&show=true
University of Southern California	Costing Software Security	http://csse.usc.edu/cse/pub/research/software_security, http://sunset.usc.edu/csse/TECHRPTS/2006/usccse2006-600/usccse2006-600.pdf
University of Stuttgart	Software Reliability and Security Group	http://www.fmi.uni-stuttgart.de/szs/index.en.shtml
University of Texas at Dallas	Secure component-based software, Software assurance	

8 TERMINOLOGY

Software	The programs, 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 Intensive	A software-intensive system is any system where software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole.

Security phrases

Countermeasure	An action, device, procedure, technique, or other measure that reduces the vulnerability or weakness of a component or system.
Defense-in-Depth	Strategy in which the human, technological, and operational capabilities of a system are integrated to establish variable protective barriers across multiple layers and dimensions of that system.
Mitigation	Countermeasure; the term is typically used in the context of risk.
Safety	Sustaining predictable, dependable execution in the face of unpredictable but unintentional faults.
Security	Sustaining predictable, dependable execution in the face of intentional attacks
Security Properties	The characteristics of a system that protect its information, typically Confidentiality, Integrity, and Availability; sometimes also includes Accountability and Non-Repudiation.
Software Assurance	(1) The justifiable trustworthiness in meeting established business and security objectives [OMG], (2) The level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software.
Trusted Software	Software that has been certified and/or accredited according to specific criteria.

Sections 2.3 and 2.4 of the SwASOAR discuss the differences among similar-seeming security related terms, including:

- Software security
- Application security
- Information security

- Network security
- System security
- Software assurance
- Information assurance
- Security requirement
- Secure software requirement
- Software protection [SwASOAR 2.3, 2.4]

Bad things

Attack	Attempt to gain unauthorized access to a system's services or to compromise one of its required properties.
Compromise	A violation of the security policy of a system, or an incident in which any of the security properties of the system are violated.
Error	Discrepancy between a computed, observed, or measured value or condition and the true, specified or theoretically correct value or condition.
Failure	(1) Non-performance by a system or component of an intended function or service. (2) Deviation of the system's performance from its specified, expected parameters.
Fault	A manifestation of an error in software. If encountered, may cause a failure. Synonyms are Bug and Defect.
Hazard	Unsponsored or unplanned event, accident, or mishap.
Lapse	An error committed by a person as the result of a bad or incorrect decision or judgment by that person. Also, Mistake.
Malware/ malicious logic	Undocumented software or firmware intended to perform an unauthorized or unanticipated process that will have adverse impact on the dependability of a component or system.
Risk	The likelihood that a particular threat will adversely affect a system by exploiting a particular vulnerability.
Threat	Anything with the potential to harm the software system or component through its unauthorized access, destruction, modification, and/or denial of service.
Validation	The technique of evaluating a product to ensure it complies with specified requirements.
Verification	The technique of evaluating a product to confirm it satisfies the conditions imposed.

Vulnerability	A development fault or weakness in deployed software that can be exploited with malicious intent by a threat.
Weakness	A flaw, defect, or anomaly in software that has the potential of being exploited as a vulnerability when the software is operational.

-ilities

Attack-tolerance	The ability of a system to provide continued correct execution when attacked.
Availability	The degree to which the services of a system or component are operational and accessible when needed by their intended users.
Correctness	The degree to which software is free of faults, and consistent with its specification.
Dependability	A collective term subsuming the notions of reliability, safety, availability, integrity, and security.
Fault-tolerance	The built-in capability of a system to provide continued correct execution in the presence of a limited number of faults.
Predictability	The degree to which it is possible to know beforehand how a system will respond to an event.
Quality	The degree to which a set of inherent characteristics fulfills requirements.
Reliability	The probability of a product performing its intended function under specific conditions for a given period of time.
Robustness	The extent to which software can continue to operate correctly despite the violation of assumptions in its specification, such as invalid input.
Safety	Dependability in the face of realized hazards.
Usability	The degree to which the software or system enables or its users to achieve their goals.

8.1 Acronyms

ACSAC	Annual Computer Security Applications Conference
AEGIS	Appropriate and Effective Guidance in Information Security
API	Application Program Interface
ASP	Active Server Pages programming language
ASQ	Association for Software Quality
BSA	Business Software Alliance
BSI	Build Security In
C&A	Certification and Accreditation
CAPEC	Common Attack Pattern Enumeration and Classification
CASP	Certified Application Security Professional
CC	Common Criteria
CD	Committee Draft
CDR	Critical Design Review
CIA	Confidentiality, Integrity, Availability
CIO	Chief Information Officer
CLASP	Comprehensive, Lightweight Application Security Process
CM	Configuration Management
CMBOK	Configuration Management Body of Knowledge
CME	Common Malware Enumeration
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integrated
CMMi	Capability Maturity Model Integration
CMMiGQ(I)M	Capability maturity Model Integration Goal Question Indicator Metric
CMU SEI	Carnegie Mellon University Software Engineering Institute
CMVP	Cryptographic Module Verification Program
CNSS	Committee on National Security Systems
COCOMO	Constructive Cost Model
CONIPMO	Constructive Network Infrastructure Protection Model
COSECMO	Security Extension to COCOMO II
COTS	Commercial Off the Shelf
CSAD	Certified Secure Application Developer
CSO	Chief Security Officer
CTO	Chief Technology Officer
CVE	Common Vulnerability Enumeration
CVS	Version Control for Source Code
CWE	Common Weakness Enumeration
DAA	Designated Approving Authority
DACS	Data & Analysis Center for Software
DCID 6/3	Protecting Compartmented Information within Information Systems
DHS	Department of Homeland Security
DHS-CERT	DHS Computer Emergency Response Team
DIACAP	DoD Information Assurance Certification and Accreditation Process
DIMACS	Center for Discrete Mathematics and Theoretical Computer Science
DITSCAP	DoD Information Technology Security Certification & Accreditation

	Process
DoD	Department of Defense
DODI	Department of Defense Instruction
DoS	Denial of Service
EABOK	Enterprise Architecture Body of Knowledge
EAL	Evaluation Assurance Level
E-Council	International Council of Electronic Commerce Consultants
ECSP	EC-Council Certified Secure Programmer
FAA	Federal Aviation Administration
FBI	Federal Bureau of Investigation
FIPS	Federal Information Processing Standards
FISMA	Federal Information Security Management Act
ftp	File Transfer Protocol
GRC	NASA Glenn Research Center
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
IA	Information Assurance
IAC	Information Analysis Center
IATAC	Information Assurance Technology Analysis Center
IAVA	Information Assurance Vulnerability Alert
iCMM	FAA's version of CMMI
ICSE	International Conference on Software Engineering
ICSM'	International Conference on Software Maintenance
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IEEE-CS	IEEE Computer Society
IS	Information Security
ISC ACE	Integrated Computer Engineering, A Division of American Systems Corporation
ISO/IEC	International Organization for Standardization/International Electrotechnical Commission
ISPA	International Society of Parametric Analysts
ISSA	Information Systems Security Association
ISSO	Information Systems Security Operation
IT	Information Technology
ITAA	Information Technology Association of America
IV&V	Independent Verification and Validation
JRC	Joint Research Center
JSP	Java Server Pages programming language
JTC	Joint Technical Committee
KLOC	Thousand Lines of Code
KSLOCS	Thousand Source Lines of Code
MBASE	Model-Based Architecting and Software Engineering
MCAD	Microsoft Certified Application Developer
MCSD	Microsoft Certified Solution Developer
MOD	Ministry of Defense (UK)

MS	Microsoft
NASA	National Aeronautics and Space Administration
NCSP	National Cyber Security Partnership
NIST	National Institute of Standards and Technology
OMG	Object Management Group
OS	Operating System
OSS	Open Source Software
OTS	Off the Shelf Software
OVAL	Open Vulnerability and Assessment Language
OWASP	Open Web Application Security Project
PC	Personal Computer
PDR	Preliminary Design Review
PERT	Program (or Project) Evaluation and Review Technique
PHP	programming language
PKI	Public Key Infrastructure
PM	Project Management
PMBOK	PM Body of Knowledge
PMI	Project Management Institute
PRINCE2	PRojects IN Controlled Environments
PSM	Practical Software Measurement
PSM	Practical Software and System Measurement
PSM	Practical Software Measurement
QA	Quality Assurance
R&D	Research & Development
ROI	Return on Investment
RUP	Rational Unified Process
RUPSec	Rational Unified Process Secure
S2ESC	Software & Systems Engineering Standards Committee
SafSec	Safety & Security
SAMATE	Software Assurance Metrics And Tool Evaluation
SANS	SysAdmin, Audit, Network, Security
SC	Subcommittee
SCCS	Source Code Control System
SCM	Software Configuration Management
SDL	Microsoft Trustworthy Computing Security Development Life Cycle
SDLC	Software Development Life Cycle
SDP	Software Development Plan
SECU	Security Driver Rating Factor
SEHAS	Software Engineering for High Assurance Systems
SEI	Software Engineering Institute
SEPG	Software Engineering Process Group
SEW	Software Engineering Workshop
SIG	Special Interest Group
SITE	Site Driver Rating Factor
SLOC	Source Lines of Code
SOAR	State of the Art Report
SOX	Sarbanes-Oxley

SPC	Statistical Process Control
SPDR	Secure Protected Development Repository
SPI	Software Process Improvement
SPMN	Software Project Managers Network
SPSA	Secure Programming Skills Assessment (exam for the SANS CASP certification)
SQL	Structured (or standard) Query Language
SRR	System Requirements Review
SSAA	System Security Authorization Agreement
SSAI	Software Security Assessment Instrument
SSDM	Secure Software Development Model
SSEC	Software Security Engineering Certification
SSLC	DHS Security in the Software Life Cycle
Std	Standard
STN	Software Tech News
STSC	Software Technology Support Center
SW	Software
SwA	Software Assurance
SwACBK	DHS Software Assurance Common Body of Knowledge
SwASOAR	*DACS and IATAC, Software Security Assurance: A State-of-the-Art-Report*
SWEBOK	Software Engineering Body of Knowledge
TOE	Target Of Evaluation
TRR	Test Readiness Review
TSP-Secure	Team Software Process for Secure Software Development
TWG	Technical Working Group
UG	Users' Group
UK	United Kingdom
US DoD	United States Department of Defense
US NIAP	United States National Information Assurance Partnership
USC	University of Southern California
WASC	Web Application Security Consortium
WETICE	Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises
WG	Working Group
XP	eXtreme Programming

9 REFERENCES

[Abran 2004]	Alain Abran, James W. Mooere (Executive editors); Pierre Bourque, Robert Dupuis, Leonard Tripp (Editors), *Guide to the Software Engineering Body of Knowledge*, 2004 Version, IEEE Computer Society, 2004
[Ankrum 2006]	T. Scott Ankrum and Alfred H. Kromholz, *Structured Assurance Cases: Three Common Standards*, presented at Association for Software Quality (ASQ) Section 509 Meeting, The MITRE Corporation, 25 January 2006, http://www.asq509.org/ht/action/GetDocumentAction/id/2132
[Bartol 2005]	Nadya Bartol, *IA Metrics - Why and how to Measure Goodness of Information Assurance*, Ninth Annual PSM Users' Group Conference: Measurement in Support of System and Process Integrity, 18-22 July 2005, http://www.psmsc.com/UG2005/Presentations/15_Bartol_IA_Metrics.pdf
[Baskerville 2003]	Baskerville, R., and Portougal, V., *A Possibility Theory Framework for Security Evaluation in National Infrastructure Protection,* Journal of Database Management, 14(2), 1-13, 2003
[Better SCM]	http://better-scm.berlios.de/comparison/comparison.html
[BSI]	https://buildsecurityin.us-cert.gov/daisy/bsi/home.html
[BSIa]	Business Case Models https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/business.html
[CAPEC]	http://capec.mitre.org
[Caseley 2003]	Paul Caseley, *Safety and Security*, Seventh Annual PSM Users' Group Conference: Making Measurement Work throughout the Enterprise, 14-18 July 2003 http://www.psmsc.com/UG2003/Presentations/16CaseleyFinal.pdf
[CME]	http://cme.mitre.org
[CNSS 2006]	Committee on National Security Systems (CNSS), *National Information Assurance (IA) Glossary, Instruction No. 4009*, 2006
[CNSS]	Committee on National Security Systems, Instructions, http://www.cnss.gov/instructions.html

[Colbert 2002]	Ed Colbert and Murali Gangadharan, *Security Extensions to COCOMO II*, 17th International Forum on COCOMO and Software Cost Modeling, USC Center for Software Engineering, October 22-25, 2002, http://www.sunset.usc.edu/events/2002/cocomo17/COCOMO%20Forum%20%20Security%20Extensions.pdf
[Colbert 2004]	Ed Colbert, Barry Boehm, *Costing the Development of Secure Systems,* 8th Annual Practical Software & Systems Measurement Users' Group Conference: Measurement for Enterprise Excellence, 26-30 July 2004, http://www.psmsc.com/UG2004/Presentations/BoehmBarryColbertEd_CostingDevelopmentOfSecureSystems.pdf
[CVE]	http://cve.mitre.org
[CWE]	http://cwe.mitre.org
[Cylab]	http://www.cylab.cmu.edu/default.aspx?id=2015
[DHS AWG 2006]	Mary Linda Polydys, Editor, *Acquisition Management Guide for Software Assurance,* Draft, Version 0.7, July 20, 2006, http://standards.computer.org/sesc/s2esc_excom/minutes/2006-08/Draft%20SwA%20Mgrs%20Desktop%20Guide%20-%20July%2020.doc
[DHS AWG]	DHS Software Assurance Forum Software Acquisition Working Group
[DHS BCWG]	DHS Software Assurance Forum Business Case Working Group
[DHS MWG 2007]	DHS Software Assurance Forum Measurement Working Group, *Practical Measurement Guidance for Software Assurance and Information Security*, DRAFT, Version 2.0, Mar 3, 2007
[Eaton]	newsgroup compilation: http://www.daveeaton.com/scm/CMTools.html
[E-Council]	EC-Council Certified Secure Programmer and Certified Secure Application Developer, http://www.eccouncil.org/ECSP.htm
[Ellison]	Robert J. Ellison, *Security and Project Management*, https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/project/38.html?branch=1&language=1
[Flechis 2004]	Ivan Flechis, M. Angela Sasse, Stephen M. V. Hailes, *Bringing Security Home: A process for developing secure and usable systems*, ACM New Security Paradigms Workshop 2003, Ascona Switzerland, c2004, quoting @stake, *The Security of Applications: Not All Are Created Equal*, 2002, http://www.atstake.com (no longer online)

[Gilliam 2003]	David P. Gilliam, Thomas L. Wolfe, Josef S. Sherif, Matt Bishop, *Software Security Checklist for the Software Life Cycle,* Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03) pp.1080-1383
[Google]	Google CM tools directory: http://www.google.com/Top/Computers/Software/Configuration_Management/Tools/
[Ibrahim 2004]	Linda Ibrahim, et al, *Safety and Security Extensions for Integrated Capability Maturity Models*, Washington D.C.: United States Federal Aviation Administration, Sept. 2004, http://www.faa.gov/ipg/pif/evol/index.cfm
[IEEE 15026]	P15026 - Standard for Systems and Software Engineering - Systems and Software Assurance, http://standards.ieee.org/board/nes/projects/15026.pdf
[ISF]	Information Security Forum, *Standard of Good Practice,* http://www.isfsecuritystandard.com/index_ie.htm
[ISO/IEC 15408]	Information technology -- Security techniques -- Evaluation criteria for IT security – Part 3 Security assurance requirements, http://standards.iso.org/ittf/PubliclyAvailableStandards/c040614_ISO_IEC_15408-3_2005(E).zip
[ISSA]	Information Systems Security Association, Certifications, http://www.issa.org/Resources/Industry-Certifications.html
[Jarzombek 2005]	Joe Jarzombek, *Software Assurance Measurement Requirements: Information Needs for IA and Cyber Security*, Ninth Annual PSM Users' Group Conference: Measurement in Support of System and Process Integrity, 18-22 July 2005, http://www.psmsc.com/UG2005/Presentations/14_Jarzombek_SwAssurance_NeedMeasurement.pdf
[Kurtz 2001]	Tim Kurtz, *Project Planning and Estimation with Ask Pete*, Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop, 2001, p. 55
[McCurley 2007]	James McCurley, Dave Zubrow, and Carol Dekkers, *Measures and Measurement for Secure Software Development*, 2007-02-05, https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/measurement/227.html?branch=1&language=1
[McGarry 2001]	John McGarry, David Card, Cheryl Jones, Beth Layman, Elizabeth Clark, Joseph Dean, Fred Hall, Practical Software Measurement: Objective Information for Decision Makers, Addison-Wesley Professional; 1st edition, October 15, 2001

[McGlone 2007]	Tim McGlone, *Ex-contractor sentenced for sabotaging Navy subs*, The Virginian-Pilot, April 5, 2007, http://content.hamptonroads.com/story.cfm?story=122352&ran=199274
[McGraw 2006]	Gary McGraw, *Software security : Building Security In*, Addison-Wesley, 2006
[Minkiewicz 2005]	Arlene F. Minkiewicz, *Costs of Security in a COTS-Based Software System*, The International Society of Parametric Analysts ISPA 27th Annual Conference, June 14th, 2005, http://www.ispa-cost.org/conferences.htm
[Moss 2006a]	Michele Moss, *Getting Started with Measuring Your Security*, Tenth Annual PSM Users' Group Conference: Performance and Decision Analysis, 24-28 July 2006, http://www.psmsc.com/UG2006/Presentations/13_GettingStartedwithSecurityMeasurement_Moss.pdf
[Moss 2006b]	Michele Moss, *Mature and Secure: Creating a CMMi and ISO/IEC 21827 Compliant Process Improvement Program,* PSM Technical Working Group (TWG) Meeting, Herndon, VA, March 2006, http://www.psmsc.com/Downloads/TWGMarch06/06_Creating_CMMi_ISO_IEC%2021827CompliantProcImpProgram_Moss.pdf
[MSDN 2006]	talhahm, *Invest in security? Show me the ROI*, Microsoft Application Threat Modeling Blog, August 11, 2006, http://blogs.msdn.com/threatmodeling/archive/2006/08/11/695044.aspx
[NCSC TG-008]	National Computer Security Center, *A Guide to Understanding Trusted Distribution in Trusted Systems* (the “Dark Lavender” book), NCSC-TG-008, 1988, http://csrc.nist.gov/secpubs/rainbow/tg008.txt
[NIST 2003]	NIST, *Security Metrics Guide for Information Technology Systems*, NIST Special Publication 800-55, July 2003, http://csrc.nist.gov/publications/nistpubs/800-55/sp800-55.pdf
[OMG]	Object Management Group, *A White Paper on Software Assurance,* Nov. 28, 2005, http://swa.omg.org/docs/softwareassurance.v3.pdf
[OVAL]	http://oval.mitre.org
[OWASP]	http://www.owasp.org
[Paller 2007]	Allan Paller, SANS Announces Secure Software Programmer Exams for Application Security Certification, July 13, 2007, http://www.sans.org/press/gssp_exams_pr.pdf
[Paquet 2005]	Paquet, C. and W. Saxe, *The Business Case for Network Security: advocacy, governance and ROI*, Indianapolis: Cisco Press, 2005

[Pfleeger 1998]	Shari L. Pfleeger, *Assessing Project Risk*, DoD Software Tech News, Vol 2, No 2, 1998
[PMI 2004]	Project Management Institute, *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) - Third Edition*, 2004
[Pressman]	R.S. Pressman & Associates, Inc., *Adaptable Process Model Checklists: Risk Assessment*, http://www.rspa.com/checklists/risk.html
[PSM 2006a]	Security Measurement White Paper V3.0,13 January 2006, http://www.psmsc.com/Downloads/TechnologyPapers/SecurityWhitePaper_v3.0.pdf
[PSM 2006b]	Safety Measurement White Paper V3.0, 23 January 2006
[PSM]	http://www.psmsc.com/Default.asp
[Redwine 2007]	Samuel T. Redwine, Jr., *Towards an Organization for Software System Security Principles and Guidelines*, Version 26, February 23, 2007
[Reifer 2007]	Donald J. Reifer, *A Model for Estimating the Cost of Securing the Network Infrastructure*, 03/07, http://www.reifer.com/documents/CONIPMO-Overview.doc
[Sadovsky]	Katya Sadovsky, et al., *Best Practices on Incorporating Quality Assurance into Your Software Development Life Cycle,* http://www.educause.edu/content.asp?page_id=666&ID=EDU06277&bhcp=1
[SafSec]	Praxis High Integrity Systems Ltd, *SafSec*, http://www.praxis-his.com/safsec/index.asp
[SAMATE]	http://samate.nist.gov
[SANS 2004]	SANS NewsBites - Volume: VI, Issue: 3, http://www.sans.org/newsletters/newsbites/newsbites.php?vol=6&issue=3
[SANS 2005]	SANS NewsBites - Volume: VII, Issue: 53, http://www.sans.org/newsletters/newsbites/newsbites.php?vol=7&issue=53
[SANS 2006]	SANS NewsBites - Volume: VIII, Issue: 7, http://www.sans.org/newsletters/newsbites/newsbites.php?vol=8&issue=7
[SANS 2007a]	SANS NewsBites - Volume: IX, Issue: 34, http://www.sans.org/newsletters/newsbites/newsbites.php?vol=9&issue=34
[SANS 2007b]	SANS NewsBites - Volume: IX, Issue: 32, http://www.sans.org/newsletters/newsbites/newsbites.php?vol=9&issue=32

[SANS 2007c]	SANS NewsBites Volume: IX, Issue: 30, April 13, 2007, http://www.sans.org/newsletters/newsbites/newsbites.php?vol=9&issue=30
[SANS 519]	Security 519, Web Application Security Workshop, http://www.sans.org/training/description.php?tid=455
[Security University]	http://www.securityuniversity.net/certification.php
[Soo Hoo 2001]	Soo Hoo, K.; Sudbury, A. W.; & Jaquith, A. R., *Tangible ROI through Secure Software Engineering*, Secure Business Quarterly 1, 2 (Fourth Quarter 2001)
[SPMN]	Software Project Managers Network / Airlie Software Council http://www.spmn.com
[SSLC]	Department of Homeland Security (DHS), *Security in the Software Life Cycle: Making Software Development Processes – and Software Produced by Them – More Secure*
[SwACBK]	Department of Homeland Security (DHS), *Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software*
[SwAL]	DHS Software Assurance Landscape Document
[SwASOAR]	DACS and IATAC, *Software Security Assurance: A State-of-the-Art-Report,* July 31, 2007
[Symantec 2006]	*Symantec Internet Security Threat Report: Trends for January 06–June 06*, *Volume X*, September 2006
[WASC]	http://www.webappsec.org/projects/threat
[Wikipedia]	Wikipedia CM tools: http://en.wikipedia.org/wiki/List_of_revision_control_software

A APPENDIX: WORK BREAKDOWN STRUCTURE FOR SOFTWARE ASSURANCE

This appendix attempts to consolidate, from a management perspective, the information contained in this *Software Project Management for Software Assurance State of the Art Report* with that of the *DACS' and IATAC's Software Security Assurance: A State-of-the-Art-Report* by creating a work breakdown structure (WBS) for system engineering and software engineering activities for a security enhanced software intensive system. The intent of this WBS is to identify and focus only on the unique activities associated with software and systems assurance, and not to describe every possible activity associated with the entire life cycle of a system.

This WBS builds on the Federal Aviation Administration's Standard Work Breakdown Structure Version 4.1 (see http://fasteditapp.faa.gov/ams/do_action?do_action=ListTOC&contentUID=5). The FAA's Standard WBS defines the "complete set of activities that may be accomplished to provide a solution" that satisfies a mission. It covers birth-to-death activities of that solution. The beauty of this WBS is that it is built from a program office and system development solution perspective.

Table 7 takes this FAA WBS and adds subtasks (highlighted subtasks), where appropriate, for software and systems assurance activities. Definitions of the non-highlighted FAA WBS elements definitions can also be found at the FAA website mentioned above It can be used as an aid in estimating life cycle cost estimates for systems and software assurance.

Some assumptions of Table 7:

- Activities not associated with software or systems engineering activities are not expanded in the WBS.
- Activities unique to the FAA (e.g., *4.7.1 NAS Charting and Aeronautical Information Management* and *5.1.3 FAA Academy Maintenance*) are not expanded in the WBS. The intent of the WBS is to address a broad audience of users.

Table 7: Work Breakdown Structure for Software Assurance

WBS Number and Title
1 MISSION ANALYSIS
1.1 Identify Projected Demand for Services
1.2 Identify Technological Opportunities
1.2.1 Identify & Review COTS Technology for Secure Software Development
1.3 Identify Projected Supply of Services
1.4 Mission Needs Analysis and Assessment
1.4.1 Maintain Awareness and Review Software Assurance Threat Environment
1.5 Initial Requirements Definition
2 INVESTMENT ANALYSIS

2.1 Initial Investment Decision
2.1.1 Planning
2.1.1.1 SWA Investment Analysis Planning
2.1.1.1.1 Identify Knowledgeable SWA Investment Analysis Team Members
2.1.1.1.2 Develop SWA Investment Analysis Plan
2.1.2 Analysis
2.1.2.1 Software Assurance Analysis
2.1.2.1.1 Identify Alternatives to Achieve Software Assurance
2.1.2.1.2 Collect Data to Perform ROI Analysis for Each Alternative
2.1.2.1.3 Perform ROI Analysis for Each Alternative
2.1.2.1.4 Develop Business Case for Each Alternative
2.1.2.1.5 Perform Trade-Off Analysis and Select Alternatives
2.1.3 Documentation
2.1.3.1 Software Assurance Investment Analysis Documentation
2.1.3.1.1 Document ROI Results for Each Alternative
2.1.3.1.2 Document Business Case for Each Alternative
2.1.3.1.3 Document Basis of Decisions
2.2 Final Investment Decision
2.2.1 Planning
2.2.2 Analysis
2.2.3 Documentation
2.3 Rebaseline Decision
3 SOLUTION DEVELOPMENT
3.1 Program Management
3.1.1 Program Planning, Authorization, Management and Control
3.1.1.1 Establish Software Assurance Plans
3.1.1.2 Establish Software Security Metrics
3.1.1.3 Review/Monitor Software Assurance Performance/Metrics
3.1.1.4 Estimate Size, Cost, Schedule of Secure Software Activities
3.1.1.4.1 Establish Security/EAL Level Desired
3.1.1.4.2 Identify/Estimate Tasks to Achieve Level of Formality in Requirements Specification to Achieve EAL
3.1.1.4.3 Identify/Estimate Tasks to Achieve Level of Formality in Design Specification to Achieve EAL
3.1.1.4.4 Identify/Estimate Implementation Tasks to Achieve EAL
3.1.1.4.5 Identify/Estimate Auditing, QA, Testing, and IV&V Tasks to Achieve Assurance Confidence
3.1.1.4.6 Identify/Estimate Management Assurance Tasks to Achieve EAL (e.g., additional CM, automation)
3.1.1.4.7 Identify/Estimate Documentation Tasks to Achieve EAL
3.1.1.5 Take Corrective Actions to Resolve Software Assurance Issues
3.1.1.6 Maintain Lessons Learned on Software Assurance Activities
3.1.2 Contract and Grant Management
3.1.2.1 Perform Security/Risk-Aware Trade-off Analysis to Acquire Software OTS (includes open source/free approaches), Outsource Software Development, or Develop

Software In-House
3.1.2.2 Develop Risk Mitigation Strategies for Candidate Approaches
3.1.2.3 Develop Software Assurance Case for Reusable Software Candidate Approaches
3.1.2.4 Establish RFP/Source Selection Plan/SOW/Contract/Subcontract Language for Secure Software
3.1.2.4.1 Develop Software Assurance Terms and Conditions
3.1.2.5 Develop Measures of Contract Performance
3.1.2.6 Develop Plans to Ensure Quality, Security/Safety, Use of Best Practices
3.1.2.7 Develop Software Assurance Evaluation Criteria
3.1.2.8 Negotiate Security Aware Contract Terms and Conditions
3.1.2.9 Oversee Supplier's Delivery of Software Assurance
3.2 System Engineering
3.2.1 System Engineering Management
3.2.1.1 Select Risk-Averse/System Security Life cycle
3.2.1.2 Select and Implement Security Aware Processes, Procedures, etc.
3.2.1.3 Risk Assessment for Secure Software
3.2.1.3.1 Identify Software Assurance Risks
3.2.1.3.2 Analyze and Determine Software Assurance Risk Exposure
3.2.1.3.3 Prioritize Software Assurance Risks
3.2.1.3.4 Identify Software Assurance Risk Mitigation Techniques
3.2.2 System Requirements and Definition
3.2.2.1 Perform Attack/Fault/Threat-Tree Analysis
3.2.2.2 Identify Techniques to Break System's Security
3.2.2.3 Perform FMEA Analysis
3.2.2.4 Develop and Analyze Threat Models
3.2.2.5 Develop and Analyze Security Policy Models
3.2.2.6 Develop and Analyze Failure Models
3.2.2.7 Maintain Trace Matrix for Security requirements
3.2.2.8 Document/Maintain Security Requirements List
3.2.3 Analysis, Design, and Integration
3.2.3.1 Design & Verify Security Protocols
3.2.3.2 Design & Verify Password Protection Schemes
3.2.3.3 Design Access Control Mechanisms:
3.2.3.3.1 Application Mechanisms
3.2.3.3.2 Middleware Mechanisms
3.2.3.3.3 Operating System Mechanisms
3.2.3.3.4 Hardware Mechanisms
3.2.3.4 Design & Verify Fault Tolerance and Failure Recovery Mechanisms
3.2.3.5 Allocate System Security Requirements to Hardware/Software
3.2.4 Value Engineering
3.2.4.1 Analyze Alternative Software Security Designs
3.2.4.1.1 Perform Software Security Trade Studies
3.2.4.1.2 Recommend Software Security Solutions
3.2.5 Supportability, Maintainability, and Reliability Engineering
3.2.6 Quality Assurance Program

3.2.6.1 Establish Software Assurance QA Plans
3.2.6.2 Perform security reviews, audits of software
3.2.6.3 Monitor Performance Against Security Enhanced Processes
3.2.6.4 Review Adequacy of Security Requirements
3.2.6.5 Review Security Controls Against Requirements
3.2.7 Configuration Management
3.2.7.1 Enhance Standard CM Processes & Practices for Software Assurance
3.2.7.1.1 Place All Software Assurance Artifacts (e.g., threat models, abuse cases, patches) Under Configuration Control and Track Changes
3.2.7.1.2 Authenticate to Version Control System Using Strong Credentials
3.2.7.1.3 Digitally Sign All Check-Ins
3.2.7.1.4 Utilize Security Enhanced CM Tools
3.2.7.2 Maintain Awareness of Software Assurance Environment
3.2.7.2.1 Monitor Vulnerability Reports (e.g., from US-CERT), Analyze Impact, Suggest Corrective Action
3.2.7.2.2 Review All Patches to COTS Products Against System Security Assumptions
3.2.8 Human Factors
3.2.8.1 Training
3.2.8.1.1 Software Assurance Awareness Training
3.2.8.1.2 Security Training
3.2.8.2 Certification
3.2.8.2.1 Security Certification (e.g., Secure Programming Certification)
3.2.8.3 Work Environment
3.2.8.3.1 Protect Software Information
3.2.8.3.2 Secure the Workplace
3.2.9 Security
3.2.9.1 Develop/Maintain Software Security Standards
3.2.9.2 Develop/Maintain Information Security Standards
3.2.10 System Safety Engineering and Management
3.2.11 Other System Engineering Specialties
3.3 HW/SW Design, Development, Procurement, and Production
3.3.1 Hardware Design and Development
3.3.2 Software Design and Development
3.3.2.1 Software Requirements
3.3.2.1.1 Software Risk Analysis
3.3.2.1.2 Software Threat Analysis
3.3.2.1.3 Develop Abuse/Misuse Cases
3.3.2.1.4 Security Requirements Review
3.3.2.1.5 Document & Maintain Software Security Requirements List
3.3.2.1.6 Develop/Maintain Security Requirements Traceability
3.3.2.2 Software Design
3.3.2.2.1 Develop Architecture to Address Security Requirements
3.3.2.2.2 Refine Attack Models
3.3.2.2.3 Perform Security Architecture Review

3.3.2.2.4 Perform Security Detailed Design Review
3.3.2.2.5 Perform Security Interface Review
3.3.2.3 Software Implementation
3.3.2.3.1 Implement with Security Aware Best Practices
3.3.2.3.2 Perform Security Code Review/Resolve Issues
3.3.2.4 Software Unit Test for Security
3.3.2.4.1 Develop Test Cases for Software Security Requirements
3.3.2.4.2 Create Test Environment for Security Testing
3.3.2.4.3 Exercise Security Test Cases
3.3.2.4.4 Document Results of Security Test
3.3.2.4.5 Resolve Software Security Discrepancies
3.3.3 HW/SW Integration, Assembly, Test and Checkout
3.3.3.1 Construct Security Designs Test Cases
3.3.3.2 Analyze Security Models for Test Cases
3.3.3.3 Perform System Tests from Security Perspective
3.3.3.3.1 Black Box Testing
3.3.3.3.2 Penetration Testing
3.3.3.3.3 Abuse Case Testing
3.3.3.4 Security Test Readiness Review
3.3.3.5 Security Integration Review
3.3.3.6 Security Release Review
3.3.3.7 Assurance Case Review
3.3.4 Production Engineering
3.3.4.1 Perform Trusted Distribution of Software
3.3.5 Procurement/Production
3.3.5.1 Perform Security/Risk-Aware Trade-off Analysis to Acquire Software OTS (includes open source/free approaches), Outsource Software Development, or Develop Software In-House
3.3.5.2 Develop Risk Mitigation Strategies for Candidate Approaches
3.3.5.3 Develop Software Assurance Case for Reusable Software Candidate Approaches
3.3.5.4 Establish RFP/Source Selection Plan/SOW/Contract/Subcontract Language for Secure Software
3.3.5.4.1 Develop Software Assurance Terms and Conditions
3.3.5.5 Develop Measures of Contract Performance
3.3.5.6 Develop Plans to Ensure Quality, Security/Safety, Use of Best Practices
3.3.5.7 Develop Software Assurance Evaluation Criteria
3.3.5.8 Negotiate Security Aware Contract Terms and Conditions
3.3.5.9 Oversee Supplier's Delivery of Software Assurance
3.4 Physical and Airspace Infrastructure Design and Development
3.4.1 Facility Planning and Design
3.4.2 Real Estate
3.4.3 Physical Infrastructure
3.4.4 Airspace Redesign
3.5 Test and Evaluation
3.5.1 System Development Test and Evaluation

3.5.1.1 Security Test Readiness Review
3.5.1.2 Security Integration Review
3.5.1.3 Security Release Review
3.5.2 System Operational Test and Evaluation
3.5.2.1 Security Test Readiness Review
3.5.2.2 Security Integration Review
3.5.2.3 Security Release Review
3.5.3 System Independent Software Verification and Validation
3.5.3.1 Construct Security Designs Test Cases
3.5.3.2 Analyze Security Models for Test Cases
3.5.3.3 Perform System Tests from Security Perspective
3.5.3.3.1 Black Box Testing
3.5.3.3.2 Penetration Testing
3.5.3.3.3 Abuse Case Testing
3.5.3.4 Security Test Readiness Review
3.5.3.5 Security Integration Review
3.5.3.6 Security Release Review
3.5.3.7 Assurance Case Review
3.5.4 Independent Operational Test and Evaluation
3.5.4.1 Security Release Review
3.5.4.2 Assurance Case Review
3.6 Data and Documentation
3.7 Logistics Support
3.7.1 Logistics Support Planning
3.7.2 Test and Measurement Equipment Acquisition
3.7.3 Support and Handling Equipment Acquisition
3.7.4 Support Facilities Construction/Conversion/Expansion
3.7.5 Support Equipment Acquisition/Modification
3.7.6 Support Facilities and Equipment Maintenance
3.7.7 Initial Spares and Repair Parts Acquisition
3.7.8 Initial Training
4 IMPLEMENTATION
4.1 Program Management
4.1.1 Program Planning, Authorization, Management and Control
4.1.1.1 Establish Software Assurance Plans
4.1.1.2 Establish Software Security Metrics
4.1.1.3 Review/Monitor Software Assurance Performance/Metrics
4.1.1.4 Estimate Size, Cost, Schedule of Secure Software Activities
4.1.1.4.1 Establish Security/EAL Level Desired
4.1.1.4.2 Identify/Estimate Tasks to Achieve Level of Formality in Requirements Specification to Achieve EAL
4.1.1.4.3 Identify/Estimate Tasks to Achieve Level of Formality in Design Specification to Achieve EAL
4.1.1.4.4 Identify/Estimate Implementation Tasks to Achieve EAL
4.1.1.4.5 Identify/Estimate Auditing, QA, Testing, and IV&V Tasks to Achieve

Assurance Confidence
4.1.1.4.6 Identify/Estimate Management Assurance Tasks to Achieve EAL (e.g., additional CM, automation)
4.1.1.4.7 Identify/Estimate Documentation Tasks to Achieve EAL
4.1.1.5 Take Corrective Actions to Resolve Software Assurance Issues
4.1.1.6 Maintain Lessons Learned on Software Assurance Activities
4.1.2 Contract Management
4.1.2.1 Perform Security/Risk-Aware Trade-off Analysis to Acquire Software OTS (includes open source/free approaches), Outsource Software Development, or Develop Software In-House
4.1.2.2 Develop Risk Mitigation Strategies for Candidate Approaches
4.1.2.3 Develop Software Assurance Case for Reusable Software Candidate Approaches
4.1.2.4 Establish RFP/Source Selection Plan/SOW/Contract/Subcontract Language for Secure Software
4.1.2.4.1 Develop Software Assurance Terms and Conditions
4.1.2.5 Develop Measures of Contract Performance
4.1.2.6 Develop Plans to Ensure Quality, Security/Safety, Use of Best Practices
4.1.2.7 Develop Software Assurance Evaluation Criteria
4.1.2.8 Negotiate Security Aware Contract Terms and Conditions
4.1.2.9 Oversee Supplier's Delivery of Software Assurance
4.1.3 Human Resources Planning and Staffing
4.2 Engineering
4.3 Environmental and Occupational Safety and Health Compliance
4.4 Site Selection and Acquisition
4.5 Construction
4.6 Site Preparation, Installation, Test, and Checkout
4.6.1 Perform Certification & Accreditation
4.7 Joint Acceptance Inspection/Commissioning/Closeout
4.7.1 NAS Charting and Aeronautical Information Management Added 4/2005
4.8 Telecommunications
4.9 Implementation Training
4.9.1 Software Assurance Training
5 IN-SERVICE MANAGEMENT
5.1 Preventive Maintenance/Certification
5.1.1 Preventive Maintenance/Certification
5.1.2 System Management Office (SMO) Overhead
5.1.2.1 Maintain Environmental/Personnel Controls
5.1.2.2 Recertify/Reaccredit
5.1.3 FAA Academy Maintenance
5.2 Corrective Maintenance
5.2.1 Corrective Maintenance
5.2.1.1 Patch Management
5.2.1.2 Track Vulnerabilities
5.2.1.3 Identify Security Faults
5.2.1.3 Follow Systems Development Processes

5.2.2 System Management Office (SMO) Overhead
5.2.3 FAA Academy Maintenance
5.3 Modifications
5.4 Maintenance Control
5.5 Technical Teaming
5.5.1 Airway Transportation System Specialists Technical Teaming
5.5.2 Air Traffic Control Specialists Technical Teaming
5.5.3 Other Staff Technical Teaming
5.6 Watch Standing Coverage
5.7 Program Support
5.7.1 Program Planning, Authorization, Management and Control
5.7.1.1 Establish Software Assurance Plans
5.7.1.2 Establish Software Security Metrics
5.7.1.3 Review/Monitor Software Assurance Performance/Metrics
5.7.1.4 Estimate Size, Cost, Schedule of Secure Software Activities
5.7.1.4.1 Establish Security/EAL Level Desired
5.7.1.4.2 Identify/Estimate Tasks to Achieve Level of Formality in Requirements Specification to Achieve EAL
5.7.1.4.3 Identify/Estimate Tasks to Achieve Level of Formality in Design Specification to Achieve EAL
5.7.1.4.4 Identify/Estimate Implementation Tasks to Achieve EAL
5.7.1.4.5 Identify/Estimate Auditing, QA, Testing, and IV&V Tasks to Achieve Assurance Confidence
5.7.1.4.6 Identify/Estimate Management Assurance Tasks to Achieve EAL (e.g., additional CM, automation)
5.7.1.4.7 Identify/Estimate Documentation Tasks to Achieve EAL
5.7.1.5 Take Corrective Actions to Resolve Software Assurance Issues
5.7.1.6 Maintain Lessons Learned on Software Assurance Activities
5.7.2 Contract Management
5.7.2.1 Perform Security/Risk-Aware Trade-off Analysis to Acquire Software OTS (includes open source/free approaches), Outsource Software Development, or Develop Software In-House
5.7.2.2 Develop Risk Mitigation Strategies for Candidate Approaches
5.7.2.3 Develop Software Assurance Case for Reusable Software Candidate Approaches
5.7.2.4 Establish RFP/Source Selection Plan/SOW/Contract/Subcontract Language for Secure Software
5.7.2.4.1 Develop Software Assurance Terms and Conditions
5.7.2.5 Develop Measures of Contract Performance
5.7.2.6 Develop Plans to Ensure Quality, Security/Safety, Use of Best Practices
5.7.2.7 Develop Software Assurance Evaluation Criteria
5.7.2.8 Negotiate Security Aware Contract Terms and Conditions
5.7.2.9 Oversee Supplier's Delivery of Software Assurance
5.8 Logistics
5.8.1 Supply Support
5.8.2 Replenishment Spares

5.8.3 Repair
5.8.4 Logistics Support Services
5.8.5 Support Equipment Maintenance
5.8.6 Technical Data
5.8.7 Maintenance Support Facilities
5.8.8 Commercial Depot Logistics Service (CDLS) Contracts
5.9 In-Service Training
5.9.1 Airway Transportation System Specialists In-Service Training
5.9.2 Air Traffic Control Specialists In-Service Training
5.10 Second Level Engineering
5.10.1 Program Management and Infrastructure Support
5.10.2 National Airspace System (NAS) Field Support and Restoration
5.10.3 Hardware and Software Engineering Support
5.10.4 Configuration Management
5.10.5 Process Improvement
5.10.6 Quality Assurance
5.10.7 Information System Security
5.10.8 Recurring NAS System Costs
5.10.9 Software Licenses
5.11 Infrastructure Support
5.11.1 Hazardous Materials Handling
5.11.2 Utilities, Building and Grounds Upkeep and Maintenance
5.11.3 Telecommunications
5.11.4 Building and Infrastructure Modernization and Improvements
5.11.5 Real Estate Management
5.11.6 Physical Security
5.12 NAS Charting and Aeronautical Information Management   Revised 4/2005
5.13 System Performance Assessment
5.14 System Operations
5.15 Travel To And From Sites
6 DISPOSITION
6.1 Program Management
6.1.1 Establish Software Assurance Plans
6.1.2 Establish Software Security Metrics
6.1.3 Review/Monitor Software Assurance Performance/Metrics
6.1.4 Estimate Size, Cost, Schedule of Secure Software Activities
6.1.4.1 Establish Security/EAL Level Desired
6.1.4.2 Identify/Estimate Tasks to Achieve Level of Formality in Requirements Specification to Achieve EAL
6.1.4.3 Identify/Estimate Tasks to Achieve Level of Formality in Design Specification to Achieve EAL
6.1.4.4 Identify/Estimate Implementation Tasks to Achieve EAL
6.1.4.5 Identify/Estimate Auditing, QA, Testing, and IV&V Tasks to Achieve Assurance Confidence
6.1.4.6 Identify/Estimate Management Assurance Tasks to Achieve EAL (e.g.,

additional CM, automation)
6.1.4.7 Identify/Estimate Documentation Tasks to Achieve EAL
6.1.5 Take Corrective Actions to Resolve Software Assurance Issues
6.1.6 Maintain Lessons Learned on Software Assurance Activities
6.1.7 Define Processes for Secure Transition
6.2 Decommissioning
6.3 Engineering
6.3.1 Test, Inspect, V&V Security Aspects of Software System After Transition
6.4 Environmental Activities
6.5 Dismantle/Removal
6.6 Site Restoration/Closeout