Secure Software Development Life Cycle Processes

Nooper Davis

July 2013

ABSTRACT: This article presents overview information about existing processes, standards, life-cycle models, frameworks, and methodologies that support or could support secure software development. The initial report issued in 2006 has been updated to reflect changes.

INTENDED AUDIENCE[1]: The target audience for this document includes program and project managers, developers, and all individuals supporting improved security in developed software. It is also relevant to software engineering process group (SEPG) members who want to integrate security into their standard software development processes.

Scope

Technology and content areas described include existing frameworks and standards such as the Capability Maturity Model Integration[2] (CMMI) framework, Team Software Process (TSP),[3] the FAA-iCMM, the Trusted CMM/Trusted Software Methodology (T-CMM/TSM), and the Systems Security Engineering Capability Maturity Model (SSE-CMM). In addition, efforts specifically aimed at security in the SDLC are included, such as the Microsoft Trustworthy Computing Software Development Lifecycle, the Team Software Process for Secure Software Development (TSPSM-Secure), Correctness by Construction, Agile Methods, and the Common Criteria. Two approaches, Software Assurance Maturity Model (SAMM) and Software Security Framework (SSF), which were just released, have been added to give the reader as much current information as possible.

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-2612

Phone: 412-268-5800
Toll-free: 1-888-201-4479

www.sei.cmu.edu

1 Some of the content of this article is used with permission from the Software Engineering Institute report CMU/SEI-2005-TN-024.

2 CMM, Capability Maturity Model, and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

3 Team Software Process and TSP are service marks of Carnegie Mellon University.

Definitions

These are some terms used in this document for which a common understanding would be useful.

Process – The IEEE defines a process as "a sequence of steps performed for a given purpose" [IEEE 90]. A secure software process can be defined as the set of activities performed to develop, maintain, and deliver a secure software solution. Activities may not necessarily be sequential; they could be concurrent or iterative.

Process model – A process model provides a reference set of best practices that can be used for both process improvement and process assessment. Process models do not define processes; rather, they define the characteristics of processes. Process models usually have an architecture or a structure. Groups of best practices that lead to achieving common goals are grouped into process areas, and similar process areas may further be grouped into categories. Most process models also have a capability or maturity dimension, which can be used for assessment and evaluation purposes.

It is important to understand the processes that an organization is using to build secure software because unless the process is understood, its weaknesses and strengths are difficult to determine. It is also helpful to use common frameworks to guide process improvement, and to evaluate processes against a common model to determine areas for improvement. Process models promote common measures of organizational processes throughout the software development life cycle (SDLC). These models identify many technical and management practices. Although very few of these models were designed from the ground up to address security, there is substantial evidence that these models do address good software engineering practices to manage and build software [Goldenson 03, Herbsleb 94].

Even when organizations conform to a particular process model, there is no guarantee that the software they build is free of unintentional security vulnerabilities or intentional malicious code. However, there is probably a better likelihood of building secure software when an organization follows solid software engineering practices with an emphasis on good design, quality practices such as inspections and reviews, use of thorough testing methods, appropriate use of tools, risk management, project management, and people management.

Standards – Standards are established by some authority, custom, or by general consent as examples of best practices. Standards provide material suitable for the definition of processes.

Assessments, evaluations, appraisals – All three of these terms imply comparison of a process being practiced to a reference process model or standard. Assessments, evaluations, and appraisals are used to understand process capability in order to improve processes. They help determine whether the processes being practiced are adequately specified, designed, integrated, and implemented to support the needs, including the security needs, of the software product. They are also an important mechanisms for selecting suppliers and then monitoring supplier performance.

Software assurance – SwA 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 06]. In the Capability Maturity Model for Software, the purpose of "software assurance" is described as providing appropriate visibility into the process being used by the software projects and into the products being built [Paulk 93].

Security assurance – Although the term "security assurance" is often used, there does not seem to be an agreed upon definition for this term. The Systems and Security Engineering CMM describes "security assurance" as the process that establishes confidence that a product's security needs are being met. In general, the term means the activities, methods, and procedures that provide confidence in the security-related properties and functions of a developed solution.

In the Security Assurance section of its Software Assurance Guidebook [NASA], NASA defines a minimum security assurance program as one that ensures the following:

- A security risk evaluation has been performed.
- Security requirements have been established for the software and data being developed and/or maintained.
- Security requirements have been established for the development and/or maintenance process.
- Each software review and/or audit includes evaluation of security requirements.
- The configuration management and corrective action processes provide security for the existing software and the change evaluation processes prevent security violations.
- Physical security for the software and the data is adequate.

Security assurance usually also includes activities for the requirements, design, implementation, testing, release, and maintenance phases of an SDLC.

BACKGROUND

A survey of existing processes, process models, and standards identifies the following four SDLC focus areas for secure software development.

1. **Security Engineering Activities.** Security engineering activities include activities needed to engineer a secure solution. Examples include security requirements elicitation and definition, secure design based on design principles for security, use of static analysis tools, secure reviews and inspections, and secure testing. Engineering activities have been described in other sections of the Build Security In web site.
2. **Security Assurance Activities.** Assurance activities include verification, validation, expert review, artifact review, and evaluations.
3. **Security Organizational and Project Management Activities.** Organizational activities include organizational policies, senior management sponsorship and oversight, establishing organizational roles, and other organizational activities that support security. Project management activities include project planning and tracking resource allocation and usage to ensure that the security engineering, security assurance, and risk identification activities are planned, managed, and tracked.
4. **Security Risk Identification and Management Activities.** There is broad consensus in the community that identifying and managing security risks is one of the most important activities in a secure SDLC and in fact is the driver for subsequent activities. Security risks in turn drive the other security engineering activities, the project management activities, and the security assurance activities. Risk is also covered in other areas of the Build Security In web site.

Other common themes include security metrics and overall defect reduction as attributes of a secure SDLC process. The remainder of this document provides overviews of process models, processes, and methods that support one or more of the four focus areas. The overviews should be read in the following context:

- Organizations need to define organizational processes. To do that, they use process standards, and they also consider industry customs, regulatory requirements, customer demands, and corporate culture.
- Individual projects apply the organizational processes, often with appropriate tailoring. In applying the organizational processes to a particular project, the project selects the appropriate SDLC activities.
- Projects use appropriate security risk identification, security engineering, and security assurance practices as they do their work.
- Organizations need to evaluate the effectiveness and maturity of their processes as used. They also need to perform security evaluations.

CAPABILITY MATURITY MODELS

Capability Maturity Models provide a reference model of mature practices for a specified engineering discipline. An organization can compare its practices to the model to identify potential areas for improvement. The CMMs provide goal-level definitions for and key attributes of specific processes (software engineering, systems engineering, security engineering), but do not generally provide operational guidance for performing the work. In other words, they don't define processes, they define process characteristics; they define the what, but not the how. "CMM-based evaluations are not meant to replace product evaluation or system certification. Rather, organizational evaluations are meant to focus process improvement efforts on weaknesses identified in particular process areas" [Redwine 04].

Historically, CMMs have emphasized process maturity to meet business goals of better schedule management, better quality management, and reduction of the general defect rate in software. Of the four secure SDLC process focus areas mentioned earlier, CMMs generally address organizational and project management processes and assurance processes. They do not specifically address security engineering activities or security risk management. They also focus on overall defect reduction, not specifically on vulnerability reduction. This is important to note, since many defects are not security-related, and some security vulnerabilities are not caused by software defects. An example of a security vulnerability not caused by common software defects is intentionally-added malicious code.

Of the three CMMs currently in fairly widespread use, Capability Maturity Model Integration (CMMI), the Federal Aviation Administration integrated Capability Maturity Model (FAA-iCMM), and the Systems Security Engineering Capability Maturity Model (SSE-CMM), only the SSE-CMM was developed specifically to address security. The Trusted CMM, derived from the Trusted Software Methodology, is also of historical importance.

Capability Maturity Model Integration (CMMI)

Capability Maturity Model Integration (CMMI) helps organizations increase the maturity of their processes to improve long-term business performance. Three different constellations of the CMMI exist: CMMI for Acquisition (CMMI-ACQ), CMMI for Services (CMMI-ACQ), and CMMI for Development (CMMI-DEV). As of December 2005, the Software Engineering Institute (SEI) reports that 1,106 organizations and 4,771 projects have reported results from CMMI-based appraisals. In November 2010, all three CMMI constellations were updated to version 1.3.

CMMI-ACQ provides improvement guidance to acquisition organizations for initiating and managing the acquisition of products and services. CMMI-SVC

provides improvement guidance to service provider organizations for establishing, managing, and delivering services.

CMMI-DEV provides the latest best practices for product and service development, maintenance, and acquisition, including mechanisms to help organizations improve their processes and provides criteria for evaluating process capability and process maturity. Improvement areas covered by this model include systems engineering, software engineering, integrated product and process development, supplier sourcing, and acquisition. CMMI-DEV has been in use for many years, replacing its predecessor, the Capability Maturity Model for Software or Software CMM (SW-CMM), which has been in use since the mid-1980s.

CMMI-DEV addresses four categories for process improvement and evaluation. Each category includes several Process Areas. As can be seen from Figure 1, CMMI-DEV addresses project management, supplier management, organization-level process improvement and training, quality assurance, measurement, and engineering practices. However, it does not specifically address the four areas mentioned earlier (security risk management, security engineering practices, security assurance, and project/organizational processes for security). Although it is not unreasonable to assume that all of these could be addressed as special cases of practices already addressed by CMMI-DEV, additional goals and practices to make assurance explicit are under development through a partnership of Booz Allen Hamilton, Motorola, and Lockheed Martin. Progress of this effort can be found on the Processes and Practices Working Group page on the Software Assurance Community Resources and Information Clearinghouse site. Further information on CMMI is available on the SEI website.

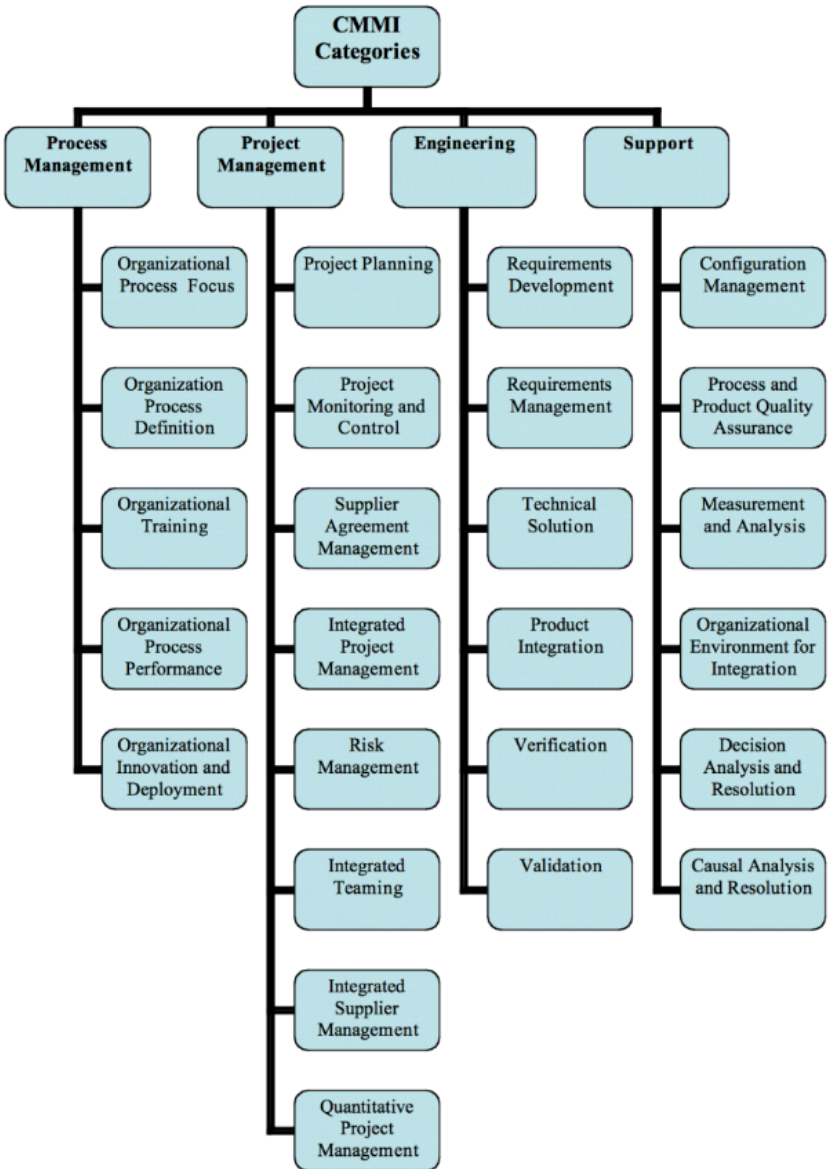

Figure 1. CMMI-DEV Process Areas

The FAA-iCMM is widely used in the Federal Aviation Administration. The FAA-iCMM provides a single model of best practices for enterprise-wide improvement, including outsourcing and supplier management. The latest version includes process areas to address integrated enterprise management, information management, deployment/transition/disposal, and operation/support. The FAA-iCMM integrates the following standards and models: ISO 9001:2000, EIA/IS 731, Malcolm Baldrige National Quality Award and President's Quality Award

criteria, CMMI-SE/SW/IPPD and CMMI-A, ISO/IEC TR 15504, ISO/IEC 12207, and ISO/IEC CD 15288.

The FAA-iCMM has been organized into the three categories and 23 Process Areas shown in Figure 2. The FAA-iCMM addresses project management, risk management, supplier management, information management, configuration management, design, and testing, all of which are integral to a secure SDLC. However, the FAA-iCMM does not address security specifically in any of these areas. Just as with CMMI, the FAA-iCMM includes generic set of best practices that do not specifically address security concerns. A reference document (PDF) with pointers to the details about the model and each process area is available.

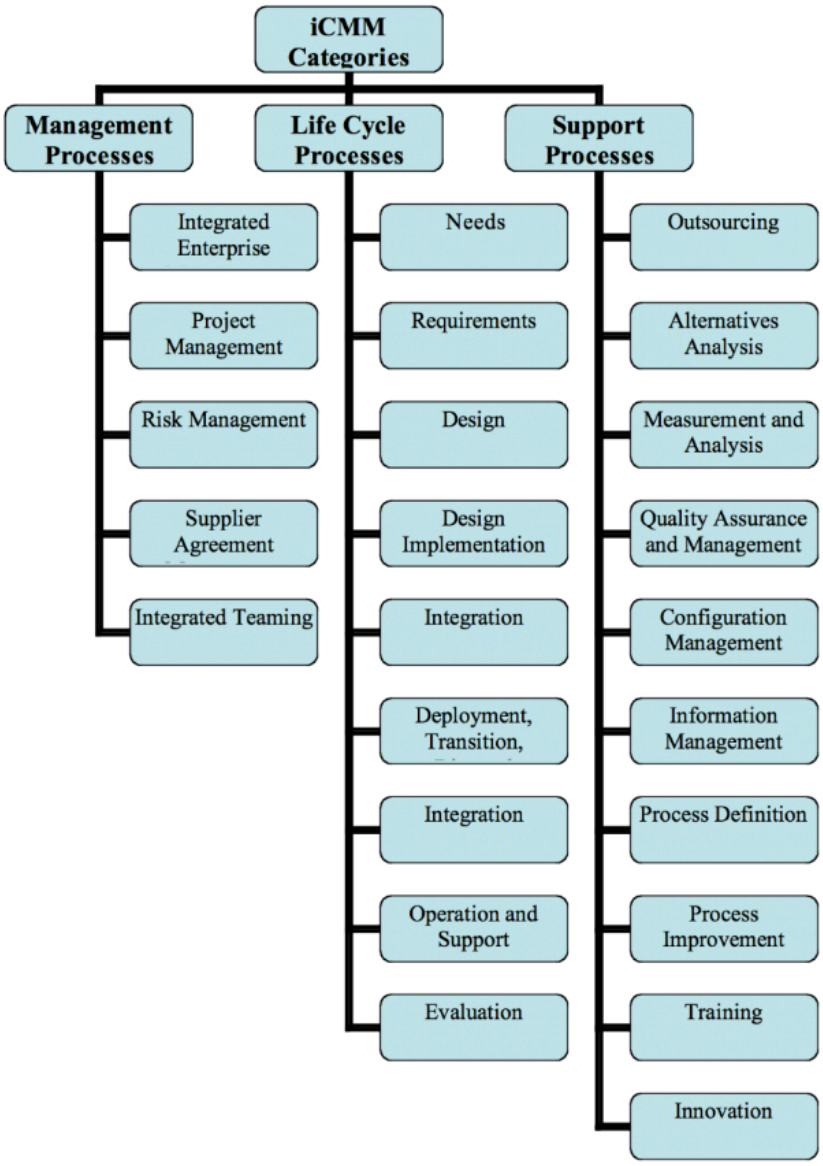

Figure 2. FAA-iCMM Process Areas

To address gaps in the coverage of safety and security, some organizations within the FAA and the Department of Defense (DoD) sponsored a joint effort to identify best safety and security practices for use in combination with the FAA-iCMM. The proposed Safety and Security extension to the FAA-iCMM identifies standards-based practices expected to be used as criteria in guiding process improvement and in appraising an organization's capabilities for providing safe and secure products and services.

The proposed Safety and Security additions include the following four goals and sixteen practices:

Goal 1 – An infrastructure for safety and security is established and maintained.

1. Ensure safety and security awareness, guidance, and competency.
2. Establish and maintain a qualified work environment that meets safety and security needs.
3. Ensure integrity of information by providing for its storage and protection and controlling access and distribution of information.
4. Monitor, report, and analyze safety and security incidents and identify potential corrective actions.
5. Plan and provide for continuity of activities with contingencies for threats and hazards to operations and the infrastructure.

Goal 2 – Safety and security risks are identified and managed.

1. Identify risks and sources of risks attributable to vulnerabilities, security threats, and safety hazards.
2. For each risk associated with safety or security, determine the causal factors, estimate the consequence and likelihood of an occurrence, and determine relative priority.
3. For each risk associated with safety or security, determine, implement, and monitor the risk mitigation plan to achieve an acceptable level of risk.

Goal 3 – Safety and security requirements are satisfied.

1. Identify and document applicable regulatory requirements, laws, standards, policies, and acceptable levels of safety and security.
2. Establish and maintain safety and security requirements, including integrity levels, and design the product or service to meet them.
3. Objectively verify and validate work products and delivered products and services to assure safety and security requirements have been achieved and fulfill intended use.
4. Establish and maintain safety and security assurance arguments and supporting evidence throughout the life cycle.

Goal 4 – Activities and products are managed to achieve safety and security requirements and objectives.

1. Establish and maintain independent reporting of safety and security status and issues.

2. Establish and maintain a plan to achieve safety and security requirements and objectives.
3. Select and manage products and suppliers using safety and security criteria.
4. Measure, monitor, and review safety and security activities against plans, control products, take corrective action, and improve processes.

Further information about safety and security extensions developed for this model is available in [Ibrahim 04].

Trusted CMM/Trusted Software Methodology (T-CMM, TSM)

In the early 1990s, the then-Strategic Defense Initiative (SDI) developed a process called the "Trusted Software Development Methodology," later renamed to the "Trusted Software Methodology (TSM)." This model defined levels of trust, with lower trust levels emphasizing resistance to unintentional vulnerabilities and higher trust levels adding processes to counter malicious developers. SDI ran experiments with the TSM to determine whether such processes could be implemented practically and what the impact of those processes would be (especially on cost and schedule). The TSM was later harmonized with the CMM, producing the Trusted CMM (T-CMM) [Kitson 95]. While the TCMM/TSM is not widely used today, it nevertheless remains a source of information on processes for developing secure software.

Systems Security Engineering Capability Maturity Model (SSE-CMM)

The SSE-CMM® is a process model that can be used to improve and assess the security engineering capability of an organization. The SSE-CMM provides a comprehensive framework for evaluating security engineering practices against the generally accepted security engineering principles. By defining such a framework, the SSE-CMM, provides a way to measure and improve performance in the application of security engineering principles. The SSE-CMM is now ISO/IEC 21827 standard and version 3 is now available. Further information about the model is available at http://www.sse-cmm.org [Redwine 04].

The stated purpose for developing the model is that, although the field of security engineering has several generally accepted principles, it lacks a comprehensive framework for evaluating security engineering practices against the principles. The SSE-CMM, by defining such a framework, provides a way to measure and improve performance in the application of security engineering principles. The SSE-CMM also describes the essential characteristics of an organization's security engineering processes.

The model is organized into two broad areas: (1) Security Engineering and (2) Project and Organizational processes. Security Engineering in turn is organized

into Engineering Processes, Assurance Processes, and Risk Processes. There are 22 Process Areas distributed amongst the three organizations. Each Process Area is composed of a related set of process goals and activities.

SEE-CMM was last revised in 2005. The model became an ISO standard in 2008. The International Systems Security Engineering Association (ISSEA) maintains the SSE-CMM.

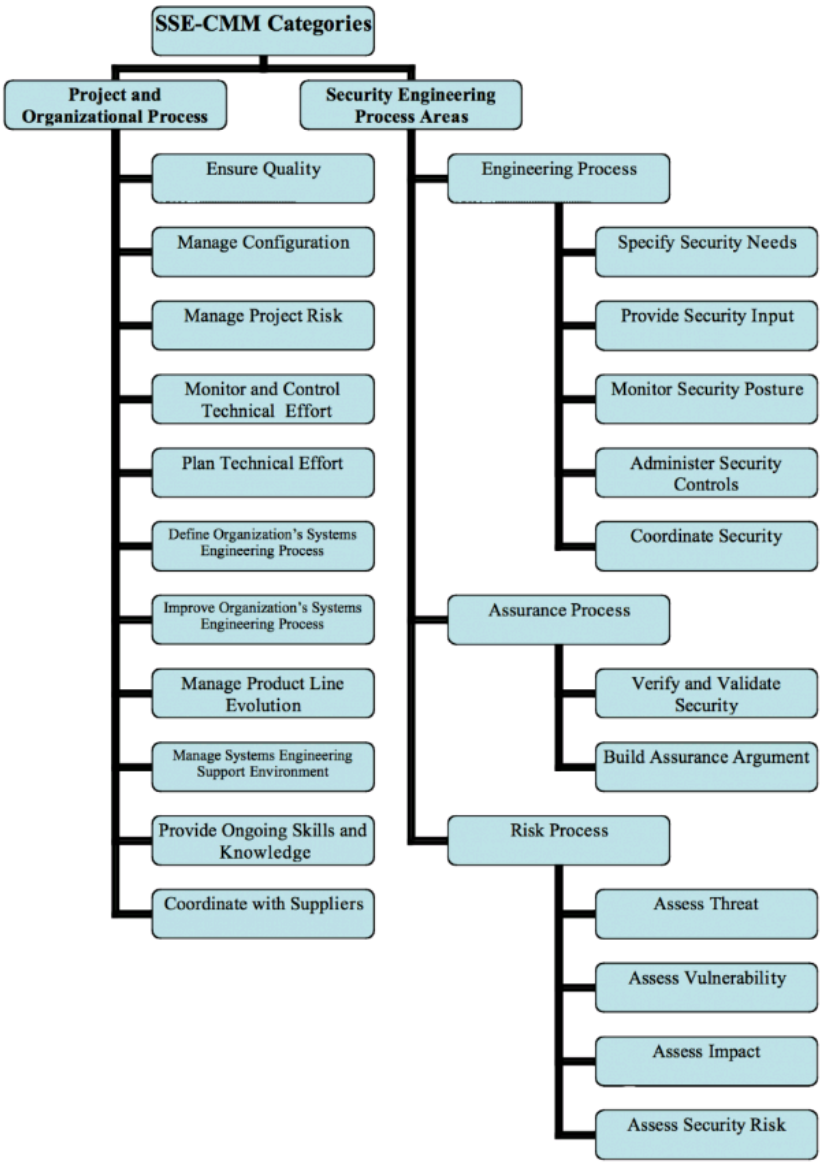

Figure 3. Process Areas of the SSE-CMM

Microsoft's Trustworthy Computing Security Development Lifecycle

The Trustworthy Computing Security Development Lifecycle (or SDL) is a process that Microsoft has adopted for the development of software that needs to withstand security attacks [Lipner 05]. The process adds a series of security-focused activities and deliverables to each phase of Microsoft's software development process. These security activities and deliverables include definition of security feature requirements and assurance activities during the requirements phase, threat modeling for security risk identification during the software design phase, the use of static analysis code-scanning tools and code reviews during implementation, and security focused testing, including Fuzz testing, during the testing phase. An extra security push includes a final code review of new as well as legacy code during the verification phase. Finally, during the release phase, a final security review is conducted by the Central Microsoft Security team, a team of security experts who are also available to the product development team throughout the development life cycle, and who have a defined role in the overall process.

Microsoft has augmented the SDL with mandatory security training for its software development personnel, with security metrics, and with available security expertise via the Central Microsoft Security team. Microsoft is reporting encouraging results from products developed using the SDL, as measured by the number of critical and important security bulletins issued by Microsoft for a product after its release.

The book titled The Security Development Lifecycle [Howard 06] further expands information about SDL from the article referenced above. Emphasis is given to the approach an organization must use for effective adoption of SDL. Management commitment to improved product security is essential. In addition to training developers and designing and building the product with appropriate security, the SDL incorporates planning for security failures after release so the organization is ready to swiftly correct unforeseen problems. The SDL is articulated as a 12 stage process as follows:

Stage 0: Education and Awareness

Stage 1: Project Inception

Stage 2: Define and Follow Design Best Practices

Stage 3: Product Risk Assessment

Stage 4: Risk Analysis

Stage 5: Creating Security Documents, Tools, and Best Practices for Customers

Stage 6: Secure Coding Policies

Stage 7: Secure Testing Policies

Stage 8: The Security Push

Stage 9: The Final Security Review

Stage 10: Security Response Planning

Stage 11: Product Release

Stage 12: Security Response Execution

Team Software Process for Secure Software Development (TSP)

The Software Engineering Institute's (SEI) Team Software Process (TSP) provides a framework, a set of processes, and disciplined methods for applying software engineering principles at the team and individual level. Software produced with the TSP has one or two orders of magnitude fewer defects than software produced with current practices—that is, 0 to .1 defects per thousand lines of code, as opposed to 1 to 2 defects per thousand lines of code.

TSP for Secure Software Development (TSP-Secure) extends the TSP to focus more directly on the security of software applications. The TSP-Secure project is a joint effort of the SEI's TSP initiative and the SEI's CERT program. The principal goal of the project is to develop a TSP-based method that can predictably produce secure software. TSP-Secure addresses secure software development in three ways. First, since secure software is not built by accident, TSP-Secure addresses planning for security. Also, since schedule pressures and people issues get in the way of implementing best practices, TSP-Secure helps to build self-directed development teams and then put these teams in charge of their own work. Second, since security and quality are closely related, TSP-Secure helps manage quality throughout the product development life cycle. Finally, since people building secure software must have an awareness of software security issues, TSP-Secure includes security awareness training for developers.

Teams using TSP-Secure build their own plans. Initial planning is conducted in a series of meetings called a project launch, which takes place over a three- to four-day period. The launch is led by a qualified team coach. In a TSP-Secure launch, the team reaches a common understanding of the security goals for the work and the approach they will take to do the work, produces a detailed plan to guide the work, and obtains management support for the plan. Typical tasks included in the plan are identifying security risks, eliciting and defining security requirement, secure design, secure design and code reviews, and use of static analysis tools, unit tests, and fuzz testing. (Fuzz testing involves sending random inputs to external program interfaces during black-box testing. The term origi-

nates from the fuzz testing application that was developed and is maintained by the University of Wisconsin [Fuzz 06, Michael 05]).

Each team member of a TSP-Secure team selects at least one of nine standard team member roles (roles can be shared). One of the defined roles is a Security Manager role. The Security Manager leads the team in ensuring that product requirements, design, implementation, reviews, and testing address security; ensuring that the product is statically and dynamically assured; providing timely analysis and warning on security problems; and tracking any security risks or issues to closure. The security manager works with external security experts when needed.

After the launch, the team executes its plan and ensures that all security-related activities are taking place. Security status is presented and discussed during every management status briefing.

Visits to web sites such as the SANS Institute's Top 20 list of security vulnerabilities, the MITRE Common Vulnerabilities and Exposures (CVE) site, the US-CERT Technical Cyber Security Alerts site, and the Microsoft Security Advisory site show that common software defects are the leading cause of security vulnerabilities (buffer overflows have been the most common software defect leading to security vulnerabilities) [Microsoft 06, MITRE 06, SANS 05, US-CERT 05]. Therefore, The TSP-Secure quality management strategy is to have multiple defect removal points in the software development life cycle. The more defect removal points there are, the more likely one is to find problems right after they are introduced, enabling problems to be more easily fixed and the root cause to be more easily determined and addressed.

Each defect removal activity can be thought of as a filter that removes some percentage of defects that can lead to vulnerabilities from the software product (see Figure 4). The more defect removal filters there are in the software development life cycle, the fewer defects that can lead to vulnerabilities will remain in the software product when it is released. More importantly, early measurement of defects enables the organization to take corrective action early in the software development life cycle.

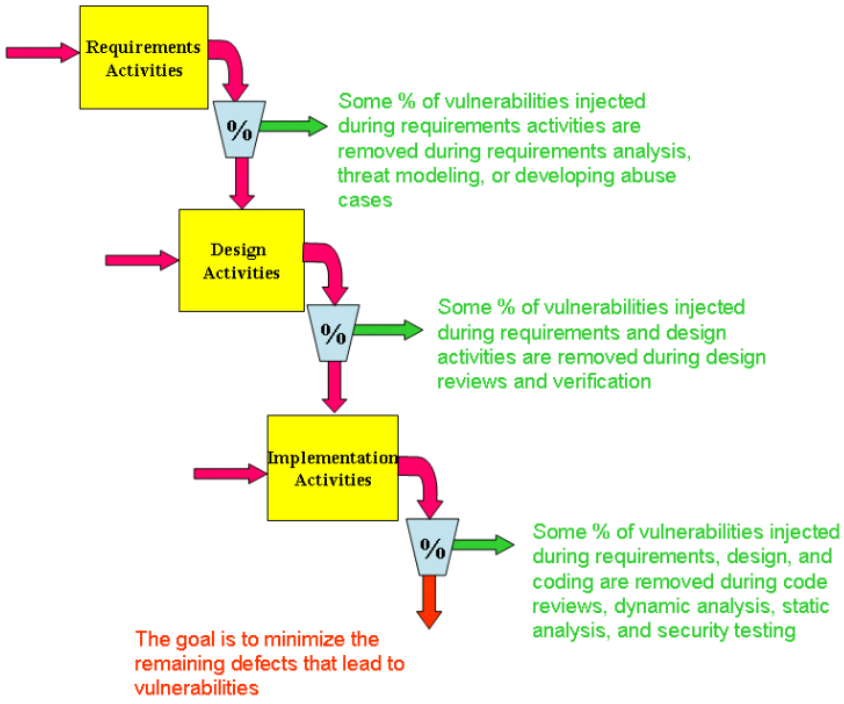

Figure 4. Vulnerability Removal Filters

Each time defects are removed, they are measured. Every defect removal point becomes a measurement point. Defect measurement leads to something even more important than defect removal and prevention: it tells teams where they stand against their goals, helps them decide whether to move to the next step or to stop and take corrective action, and indicates where to fix their process to meet their goals.

The team considers the following questions when managing defects:

- What type of defects lead to security vulnerabilities?
- Where in the software development life cycle should defects be measured?
- What work products should be examined for defects?
- What tools and methods should be used to measure the defects?
- How many defects can be removed at each step?
- How many estimated defects remain after each removal step?

TSP-Secure includes training for developers, managers, and other team members.

Correctness by Construction

The Correctness by Construction methodology of Praxis High Integrity Systems is a process for developing high-integrity software [Hall 02]. It has been used to develop safety-critical and security-critical systems with a great degree of success [Ross 05]. It delivers software with very low defect rates by rigorously eliminating defects at the earliest possible stage of the process. The process is based on the following tenets: do not introduce errors in the first place, and remove any errors as close as possible to the point that they are introduced.

The process is based on the strong belief that each step should serve a clear purpose and be carried out using the most rigorous techniques available to address that particular problem. In particular, the process almost always uses formal methods to specify behavioral, security, and safety properties of the software. There is a belief that only by using formality can the necessary precision be achieved.

The seven key principles of Correctness by Construction are

1. Expect requirements to change. Changing requirements are managed by adopting an incremental approach and paying increased attention to design to accommodate change. Apply more rigor, rather than less, to avoid costly and unnecessary rework.
2. Know why you're testing. Recognize that there are two distinct forms of testing, one to build correct software (debugging) and another to show that the software built is correct (verification). These two forms of testing require two very different approaches.
3. Eliminate errors before testing. Better yet, deploy techniques that make it difficult to introduce errors in the first place. Testing is the second most expensive way of finding errors. The most expensive is to let your customers find them for you.
4. Write software that is easy to verify. If you don't, verification and validation (including testing) can take up to 60% of the total effort. Coding typically takes only 10%. Even doubling the effort on coding will be worthwhile if it reduces the burden of verification by as little as 20%.
5. Develop incrementally. Make very small changes, incrementally. After each change, verify that the updated system behaves according to its updated specification. Making small changes makes the software much easier to verify.
6. Some aspects of software development are just plain hard. There is no silver bullet. Don't expect any tool or method to make everything easy. The best tools and methods take care of the easy problems, allowing you to focus on the difficult problems.

7. Software is not useful by itself. The executable software is only part of the picture. It is of no use without user manuals, business processes, design documentation, well-commented source code, and test cases. These should be produced as an intrinsic part of the development, not added at the end. In particular, recognize that design documentation serves two distinct purposes:
   - To allow the developers to get from a set of requirements to an implementation. Much of this type of documentation outlives its usefulness after implementation.
   - To allow the maintainers to understand how the implementation satisfies the requirements. A document aimed at maintainers is much shorter, cheaper to produce and more useful than a traditional design document.

Correctness by Construction is one of the few secure SDLC processes that incorporate formal methods into many development activities. Where appropriate, formal specification languages such as Z are used to specify functional behavior and security properties. The SPARK programming language (a design-by-contract subset of Ada) is often used to facilitate deep and constructive static verification. More details about this approach are available in the BSI article Correctness by Construction.

Agile Methods

Over the past few years, a new family of software engineering methods has started to gain acceptance amongst the software development community. These methods, collectively called Agile Methods, conform to the Agile Manifesto [Agile 01], which states:

> *"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> - *Individuals and interactions over processes and tools*
> - *Working software over comprehensive documentation*
> - *Customer collaboration over contract negotiation*
> - *Responding to change over following a plan*
>
> *That is, while there is value in the items on the right, we value the items on the left more."*

The individual Agile methods include Extreme Programming (the most well known), Scrum, Lean Software Development, Crystal Methodologies, Feature Driven Development, and Dynamic Systems Development Methodology. While

there are many differences between these methodologies, they are based on some common principles, such as short development iterations, minimal design up-front, emergent design and architecture, collective code ownership and ability for anyone to change any part of the code, direct communication and minimal or no documentation (the code is the documentation), and gradual building of test cases. Some of these practices are in direct conflict with secure SDLC processes. For example, a design based on secure design principles that addresses security risks identified during an up front activity such as Threat Modeling is an integral part of most secure SDLC processes, but it conflicts with the emergent requirements and emergent design principles of Agile methods.

In their article "Towards Agile Security Assurance," Beznosov and Kruchten address this issue and make some proposals as to how security assurance activities could be merged into Agile development methods [Beznosov 05]. They created a table that shows the compatibility of common security assurance activities with Agile methods. Table 1 (replicated here with permission from the authors) shows that almost 50% of traditional security assurance activities are not compatible with Agile methods (12 out of 26), less than 10% are natural fits (2 out of 26), about 30% are independent of development method, and slightly more than 10% (4 out of 26) could be semi-automated and thus integrated more easily into the Agile methods.

Table 1. Agile Methods Compatibility with Security Assurance Practices

Security assurance method or technique		Match (2)	Independent (8)	Semi-automated (4)	Mismatch (12
Requirements	Guidelines		X		
	Specification Analysis				X
	Review				X
Design	Application of specific architectural approaches		X		
	Use of secure design principles		X		
	Formal validation				X
	Informal validation				X
	Internal review	X			
	External review				X
Implementation	Informal requirements traceability				X
	Requirements testing			X	
	Informal validation				X

	Formal validation				X
	Security testing			X	
	Vulnerability and penetration tsting			X	
	Test depth analysis				X
	Security static analysis			X	
	High-level programming languages and tools		X		
	Adherence to implementation standards		X		
	Use of version control and change tracking		X		
	Change authorization				X
	Integration procedures		X		
	Use of product generation tools		X		
	Internal review	X			
	External review				X
	Security evaluation				X

Others have started to explore the integration of security assurance with Agile Methods [Beznosov 04, Poppendieck 02, Wayrynen 04].

The Agile Security Forum was initiated in 2005 to provide a focal point for industry-wide collaboration. Additional information about the Forum, as well as other papers expanding on the approaches to security being taken in conjunction with Agile, is available on the Forum website.

THE COMMON CRITERIA

Canada, France, Germany, the Netherlands, United Kingdom, and the United States released a jointly developed security evaluation standard in January 1996. This standard is known as the "Common Criteria for Information Technology Security Evaluation" (CCITSE) but is more often referred to as the "Common Criteria" (CC) [CC 05]. The CC has become the dominant security evaluation framework and is now an international standard, ISO/IEC 15408.

The CC is documented in three sections. The introduction section describes the history, purpose, and the general concepts and principles of security evaluation and describes the model of evaluation. The second section describes a set of se-

curity functional requirements that users of products may want to specify and that serve as standard templates for security functional requirements. The functional requirements are catalogued and classified, basically providing a menu of security functional requirements product users may select from. The third section of the document includes security assurance requirements, which includes various methods of assuring that a product is secure. This section also defines seven pre-defined sets of assurance requirements called the Evaluation Assurance Levels (EALs).

There are two artifacts that must be created to go through a CC evaluation: a Protection Profile (PP) and a Security Target (ST). Both documents must be created based on specific templates provided in the CC. A Protection Profile identifies the desired security properties (user security requirements) of a product type. Protection Profiles can usually be built by selecting appropriate components from section two of the CC, since chances are the user requirements for the type of product being built already exists. Protection Profiles are an implementation-independent statement of security needs for a product type (for example, firewalls). Protection Profiles can include both the functional and assurance requirements for the product type. A Security Target (ST) is an implementation-dependent statement of security needs for a specific product.

The Protection Profiles and the Security Target allow the following process for evaluation

1. An organization that wants to acquire or develop a particular type of security product defines their security needs using a Protection Profile. The organization then has the PP evaluated, and publishes it.
2. A product developer takes this Protection Profile, writes a Security Target that is compliance with the PP, and has this Security Target evaluated.
3. The product developer then builds a TOE (or uses an existing one) and has this evaluated against the Security Target.

The seven evaluation levels are

1. Evaluation assurance level 1 (EAL1) - functionally tested
2. Evaluation assurance level 2 (EAL2) – structurally tested
3. Evaluation assurance level 3 (EAL3) - methodically tested and checked
4. Evaluation assurance level 4 (EAL4) - methodically designed, tested, and reviewed
5. Evaluation assurance level 5 (EAL5) – semi-formally designed and tested
6. Evaluation assurance level 6 (EAL6) – semi-formally verified design and tested

7. Evaluation assurance level 7 (EAL7) - formally verified design and tested

The Common Criteria Evaluation and Validation Scheme (CCEVS) is administered in the United States by The National Institute of Standards and Technology (NIST) and the National Security Agency (NSA) under the National Information Assurance Partnership (NIAP). A list of validated products and their associated EAL level is kept up-to-date on the CCEVS website.

The Common Criteria is an internationally recognized standard. Information about the working groups and products internationally verified is available on the Common Criteria website.

SOFTWARE ASSURANCE MATURITY MODEL

A beta release of the Software Assurance Maturity Model (SAMM) came out in August 2008, and the official version 1.0 was just released in March 2009. This model was developed to aid organizations in formulating and implementing a strategy for software security. It is maintained through the OpenSAMM Project as part of the Open Web Application Security Project (OWASP). This model is designed to be tailored to the specific risk environment each organization faces. Available resources for this model [Chandra 09b] are designed to aid in the following:

- evaluation of an organization's existing software security program
- development of a balanced software security program using well-defined iterations
- demonstration of improvement of a security assurance program
- definition and measurement of security-related activities within an organization

SAMM is an open project that provides freely available content that is not vendor specific.

The model hubs on four core business functions that are involved in software development:

- Governance: processes and activities related to the way in which an organization manages its software development
- Construction: processes and activities related to the way an organization defines the goals for and the creation of software within development projects

- Verification: processes and activities related to the way an organization validates and tests artifacts created throughout software development
- Deployment: processes and activities related to the way an organization manages the operational release of software it creates to a runtime environment

The specific practice areas within each business function are listed in Table 2. A maturity level structure has been identified for each practice as follows:

- Maturity Level 0: starting point where activities in the practice are largely unfulfilled
- Maturity Level 1: practice area activities and processes are understood to an initial extent, but fulfillment is ad hoc
- Maturity Level 2: practice efficiency and/or effectiveness is increasing
- Maturity Level 3: practice area activities and processes are comprehensive, indicating full scale mastery of the area

Table 2. SAMM Structure

Governance	Construction	Verification	Deployment
Strategy and Metrics	Threat Assessment	Design Review	Vulnerability Management
Policy and Compliance	Security Requirements	Code Review	Environment Hardening
Education & Guidance	Secure Architecture	Security Testing	Operational Enablement

At this point in time, the model is too new to have reported usage results.

SOFTWARE SECURITY FRAMEWORK

Citigal and Fortify have partnered to develop the Software Security Framework (SSF). The structure of SSF was initially built on the content of SAMM and adjusted based on review of development in a set of organizations addressing secure development [Chandra 09a]. The authors of SSF have articulated a Building Security In Maturity Model (BSIMM) based on their analysis of projects in a set of organizations [Chess 09].

Table 3 shows the SSF structure. There are twelve practices organized into four domains. The domains are

- Governance: practices that help organize, manage, and measure a software security initiative

- Intelligence: practices for collecting corporate knowledge used in carrying out software security activities throughout the organization
- SDL Touchpoints: practices associated with analysis and assurance of particular software development artifacts and processes
- Deployment: practices linked to network security and software maintenance organizations

Table 3. SSF Domains and Practice Areas

Governance	Intelligence	SDL Touchpoints	Deployment
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

The practice areas group 110 activities that were identified in actual use within the nine organizations studied to develop SSF, though not all were used in any one organization. Nine activities were consistently reported in all of the studied organizations. These are listed in Table 4 [Chess 09].

Table 4. Activities Addressed in All SSF Reviewed Organizations

What	How
Build organizational support	Create security related evangelism role/internal marketing
Establish unified approach to address regulatory and customer support needs	Create security related policy
Promote an organizational culture of security	Provide security awareness training
Describe the organization's specific security issues	Create/use content specific to company history
Create security guidance through an articulation of the security features of a product	Build and publish detailed information about security features (authentication, role management, key management, audit/log, crypto, protocols)
Establish organizational capability in security architecture	Have security architect experts lead architectural and product functionality reviews
Evaluate the attacker perspective	Incorporate black box security tools into the quality review process
Identify organization-specific problem areas	Apply pen testing using external experts
Ensure a solid security infrastructure for software development and validation	Develop and test using appropriate host/network security

SUMMARY

Other key standards and methods that apply to developing secure software but have not been summarized in this technical note include

- ISO/IEC 15288 for System Life Cycle Processes, available from http://www.iso.org
- ISO/IEC 12207 for Software Life Cycle Processes, available from http://www.iso.org
- ISO/IEC 15026 for System and Software Integrity Levels, available from http://www.iso.org
- Cleanroom Software Engineering [Linger 94, Mills 87]

In conclusion, this survey of existing SDLC processes shows that several processes and methodologies which have been in broad use for many years could support secure software development. However, these were not designed specifically to address software security from the ground up. One of the major hurdles to instituting a comprehensive consideration of security in the SDLC has been the availability of security expertise for the developer as noted by Lipner in describing the first steps for Microsoft when instituting the Trustworthy Computing Initiative [Lipner 05]. Four years later, a survey by Forrester commissioned by Veracode indicates that most organizations (57%) still do not provide security training for developers [Veracode 09].

SSE-CMM, Trusted CMM, FAA-iCMM, Common Criteria, Correctness by Construction, and TSP Secure offered ways to address security within development but required security-knowledgeable resources within the process improvement group or that an organization adopt a different and more rigorous development approach. Few organizations were willing to embrace these changes.

Microsoft's Trustworthy Computing SDL was the first of a new group of life cycle approaches that seek to articulate the critical elements of security to be embedded within any existing development life cycle such that security is appropriately considered as part of normal development. Microsoft is reporting 60% fewer vulnerabilities in its operating systems released in 2008 than in 2002 [Mills 09].

The release of Version 1 of the Software Assurance Maturity Model and reports are the use of SSF in nine organizations indicate a new level of awareness of the value of embedding security into the SDLC. Organizations are showing increased response to security, but there is still a long way to go before considerations of security in the SDLC can be considered mainstream. The Veracode survey indicated 34% of the organizations included in the survey actively address

security within the SDLC, but only 13% know the security quality of their business code [Veracode 09].

REFERENCES

[Agile 01] Agile Alliance. Manifesto for Agile Software Development (2005).

[Beznosov 04] Beznosov, Konstantin. "Extreme Security Engineering: On Employing XP Practices to Achieve 'Good Enough Security' without Defining It." First ACM Workshop on Business Driven Security Engineering (BizSec). Fairfax, VA, Oct. 31, 2003.

[Beznosov 05] Beznosov, Konstantin & Kruchten, Phillipe. "Towards Agile Security Assurance," 47–54. Proceedings of the 2004 Workshop on New Security Paradigms. White Point Beach Resort, Nova Scotia, Canada, September 20-23, 2004. New York, NY: Association for Computing Machinery, 2005.

[CC 05] The Common Criteria Portal (2005).

[Chandra 09a] Chandra, Pravir. "What's up with the other model?" OpenSAMM, March 6, 2009.

[Chandra 09b] Chandra, Pravir. "Software Assurance Maturity Model," Version 1.0. http://www.opensamm.org

[Chess 09] Chess, B., McGraw, G., & Migues, S. "Confessions of a Software Security Alchemist." InformIT, March 16, 2009.

[CNSS 06] Committee on National Security Systems. National Information Assurance Glossary (CNSS Instruction No. 4009), June 2006.

[Fuzz 06] University of Wisconsin Madison. Fuzz Testing of Application Reliability (2006).

[Goldenson 03] Goldenson, D. & Gibson, D. Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results (CMU/SEI-2003-SR-009, ADA418491). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.

[Hall 02] Hall, Anthony & Chapman, Roderick. "Correctness by Construction: Developing a Commercial Secure System." IEEE Software 19, 1 (Jan./Feb. 2002): 18-25.

[Herbsleb 94] Herbsleb, J., Carleton, A., Rozum, J., Siegel, J., & Zubrow, D. Benefits of CMM-Based Software Process Improvement: Initial Results (CMU/SEI-94-TR-013, ADA283848). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1994.

[Howard 06] Howard, Michael & Lipner, Steve. The Security Development Lifecycle. Microsoft Press, 2006.

[Ibrahim 04] Ibrahim, L. et al. Safety and Security Extensions for Integrated Capability Maturity Models. United States Federal Aviation Administration, 2004.

[IEEE 90] IEEE Standards Coordinating Committee. IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990). Los Alamitos, CA: IEEE Computer Society, 1990 (ISBN 0738103918).

[Kitson 95] Kitson, David H. "A Tailoring of the CMM for the Trusted Software Domain." Proceedings of the Seventh Annual Software Technology Conference. Salt Lake City, Utah, April 9-14, 1995.

[Linger 94] Linger, R. C. “Cleanroom Process Model.” IEEE Software 11, 2 (March 1994): 50-58.

[Lipner 05] Lipner, Steve & Howard, Michael. The Trustworthy Computing Security Development Lifecycle (2005).

[Michael 05] Michael, C. C. & Radosevich, Will. Black Box Security Testing Tools (2005).

[Microsoft 06] Microsoft Corp. Microsoft Security Advisories (2006).

[Mills 87] Mills, H., Dyer, M., & Linger, R. C. “Cleanroom Software Engineering.” IEEE Software 4, 5 (September 1987): 19-25.

[Mills 09] Mills, E. “Secure Software? Experts Say it’s no Longer a Pipe Dream." CNET News, April 20, 2009.

[MITRE 06] The MITRE Corporation. Common Vulnerabilities and Exposures.

[NASA] NASA Software Assurance Technology Center. Software Assurance Guidebook, NASA-GB-A201.

[Paulk 93] Paulk, M., Curtis, B., Chrissis, M. B. & Weber, C. Capability Maturity Model for Software (Version 1.1) (CMU/SEI-93-TR-024, ADA263403). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.

[Poppendieck 02] Poppendieck, M. & Morsicato, R. “Using XP for Safety-Critical Software.” Cutter IT Journal 15, 9 (2002): 12-16.

[Redwine 04] Redwine, S. T. & Davis, N., eds. “Processes to Produce Secure Software.” Improving Security Across the Software Development Lifecycle (National Cybersecurity Partnership Taskforce Report), Appendix B. http://www.cyberpartnership.org/init-soft.html (2004).

[Ross 05] Ross, Philip E. “The Exterminators: A Small British Firm Shows That Software Bugs Aren’t Inevitable.” IEEE Spectrum 42, 9 (September 2005): 36-41.

[SANS 05] The SANS Institute. The Twenty Most Critical Internet Security Vulnerabilities (Updated) – The Experts Consensus (2005).

[SEI 09] Software Engineering Institute. Maturity Profile, 2009.

[US-CERT 05] United States Computer Emergency Readiness Team. Technical Cyber Security Alerts (2005).

[Veracode 09] Veracode. “Independent Survey Finds Enterprises At-Risk from Insecure Software.” April 19, 2009.

[Wäyrynen 04] Wäyrynen, J., Bodén, M., & Boström, G. “Security Engineering and eXtreme Programming: an Impossible Marriage?” Extreme Programming and Agile Methods - XP/Agile Universe 2004: 4th Conference on Extreme Programming and Agile Methods. Calgary, Canada, August 15-18, 2004. Berlin, Germany: Springer-Verlag, 2004 (ISBN 3-540-22839-X).

This material is based upon work funded and supported by Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

DM-0001120