Secure Software Development Life Cycle Processes: A Technology Scouting Report

Noopur Davis

December 2005

Software Engineering Process Management

Technical Note
CMU/SEI-2005-TN-024

Contents

List of Figures

List of Tables

Acknowledgements

The author thanks Steve Lipner, Shawn Hernan, and Michael Howard of Microsoft, James Moore of MITRE, and David Carrington and Julia Mullaney of the Software Engineering Institute for their review comments on earlier drafts of this report. Thanks also to Pamela Curtis and Susan Kushner for their editorial support.

Abstract

As the use of the Internet and networked systems become more pervasive, the importance of developing secure software increases. The purpose of this technical note is to present overview information about existing processes, standards, life cycle models, frameworks, and methodologies that support or could support secure software development. Where applicable and possible, some evaluation or judgment is provided.

The target audience for this technical note includes software engineering process group (SEPG) members, software developers, and managers seeking information about existing software development life cycle (SDLC) processes that address security.

1 Introduction

The purpose of this document is to collect and present overview information about existing processes, standards, life cycle models, frameworks, and methodologies that support or could support secure software development. Where applicable and possible, some evaluation or judgment may be provided for particular life cycle models, processes, frameworks, and methodologies.

The target audience for this document includes software engineering process group (SEPG) members who want to integrate security into their standard software development processes. It is also relevant for developers and managers looking for information on existing software development life cycle (SDLC) processes that address security. Technology or content areas described include existing frameworks and standards such as the Capability Maturity Model® Integration (CMMI®) framework, the FAA-iCMM, the Trusted CMM/Trusted Software Methodology (T-CMM/TSM), the Systems Security Engineering Capability Maturity Model (SSE-CMM), in addition to existing processes such as the Microsoft Trustworthy Computing Software Development Lifecycle, the Team Software Process℠ for Secure Software Development (TSP℠-Secure), Correctness by Construction, Agile Methods, and the Common Criteria.

Future technical notes can focus on secure engineering practices and tools such as threat modeling, use of secure design principles, and use of static analysis tools.

1.1 Definitions

There are some terms used in this technical note for which a common understanding is useful. They are

- **Process** – The Institute of Electrical and Electronics Engineers (IEEE) defines a process as “a sequence of steps performed for a given purpose” [IEEE 91]. 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

® CMM, Capability Maturity Model, and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.
℠ Team Software Process and TSP are service marks of Carnegie Mellon University.

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 that 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 create common measures of organizational processes throughout the SDLC. These models identify and employ many best 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].

There is no guarantee that even when organizations conform to a particular process model, 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.

Some additional terms used in this document are

- **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.
- **Appraisals, evaluations, assessments** – 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 if the processes being practiced are adequately specified, designed, integrated, and implemented sufficiently to support the needs, including the security needs, of the software product. They are also an important mechanism for selecting suppliers and then monitoring supplier performance.
- **Security assurance** – Although the term "security assurance" is often used, there does not seem to be an agreed-upon definition for this term. In the Capability Maturity Model for Software (SW-CMM), 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]. The SSE-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*, the National Aeronautics and Space Administration (NASA) defines a minimum security assurance program is 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 that the change evaluation processes prevent security violations.
- Physical security for the software and the data is adequate.

Security assurance usually also includes activities for requirements, design, implementation, testing, release and maintenance phases of a SDLC [NASA 89].

1.2 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 those 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 methods. A good source of information about secure engineering activities is the Department of Homeland Security (DHS) *Build Security In* Web site [DHS 05].
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. Security risk is also addressed in the DHS *Build Security In* Web site [DHS 05].

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

- Organizations need to define organizational processes. To do so, they use process standards but they also consider industry customs, regulatory requirements, customer demands, and corporate culture.
- Individual projects apply the organizational processes, often with appropriate tailoring. When 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.

2 Capability Maturity Models (CMMs)

Capability Maturity Models provide a reference model of mature practices for a specified engineering discipline. An organization can compare their 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, project management, 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. For example, intentionally added malicious code is a security vulnerability not caused by common software defects.

Of the three CMMs currently in widespread use, the CMMI framework, the FAA-iCMM, and the 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. This section discusses each of these CMMs in more detail.

2.1 Capability Maturity Model Integration (CMMI)

The Capability Maturity Model Integration (CMMI) framework helps organizations increase the maturity of their processes to improve long-term business performance. The CMMI 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. The CMMI has been in use for more than three years and will eventually replace its predecessor, the Capability Maturity

Model for Software (SW-CMM), which has been in use since the mid-1980s. As of June 2005, the Software Engineering Institute (SEI) reports that 782 organizations and 3250 projects have reported results from CMMI-based appraisals [SEI 05a]. Beginning in 1987 through June 2005, 2,859 organizations and 15,634 projects have reported results from SW-CMM-based appraisals and assessments [SEI 05b].

The CMMI addresses four categories for process improvement and evaluation. Each category includes several Process Areas. As shown in Figure 1, the CMMI addresses project management, supplier management, organization-level process improvement as well as 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 each of these are special cases of practices already addressed by the CMMI. Further information about the CMMI framework is available at http://www.sei.cmu.edu/cmmi/.

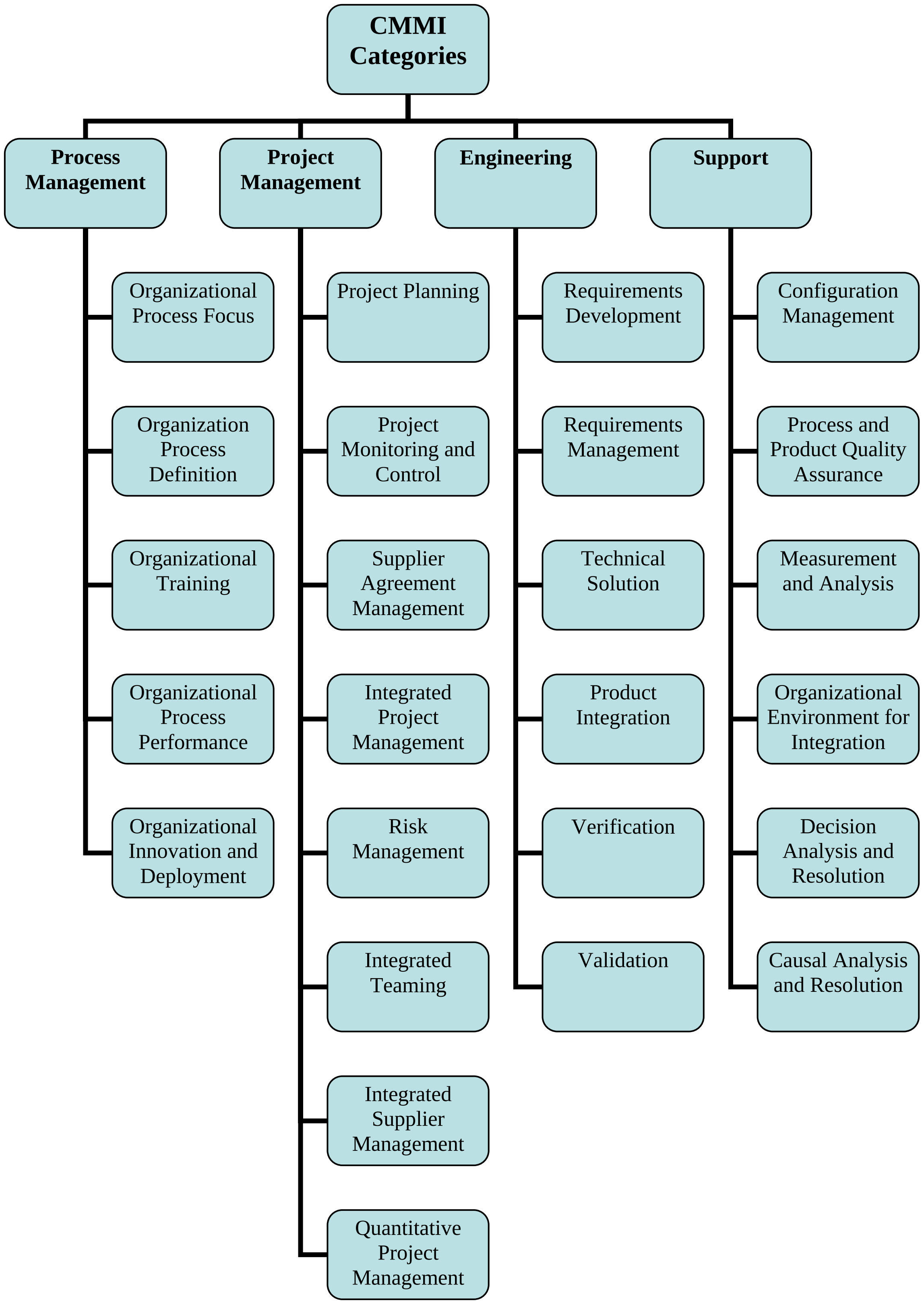

CMMI Categories
Process Management
Project Management
Engineering
Support
Organizational Process Focus
Organization Process Definition
Organizational Training
Organizational Process Performance
Organizational Innovation and Deployment
Project Planning
Project Monitoring and Control
Supplier Agreement Management
Integrated Project Management
Risk Management
Integrated Teaming
Integrated Supplier Management
Quantitative Project Management
Requirements Development
Requirements Management
Technical Solution
Product Integration
Verification
Validation
Configuration Management
Process and Product Quality Assurance
Measurement and Analysis
Organizational Environment for Integration
Decision Analysis and Resolution
Causal Analysis and Resolution

Figure 1: Process Areas of the CMMI Framework

2.2 Federal Aviation Administration integrated Capability Maturity Model (FAA-iCMM)

The FAA-iCMM was developed and is widely used by the Federal Aviation Administration. It 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
- President's Quality Award
- CMMI-SE/SW/IPPD and CMMI-A
- ISO/IEC TR 15504, ISO/IEC 12207, and ISO/IEC CD 15288

As shown in Figure 2, the FAA-iCMM is organized into three main categories and 23 Process Areas [FAA 01]. 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 the CMMI, the FAA-iCMM includes a generic set of best practices that do not specifically address security concerns. Details about the FAA-iCMM model and each process area are available at http://www.faa.gov/aio or http://www.faa.gov/ipg.

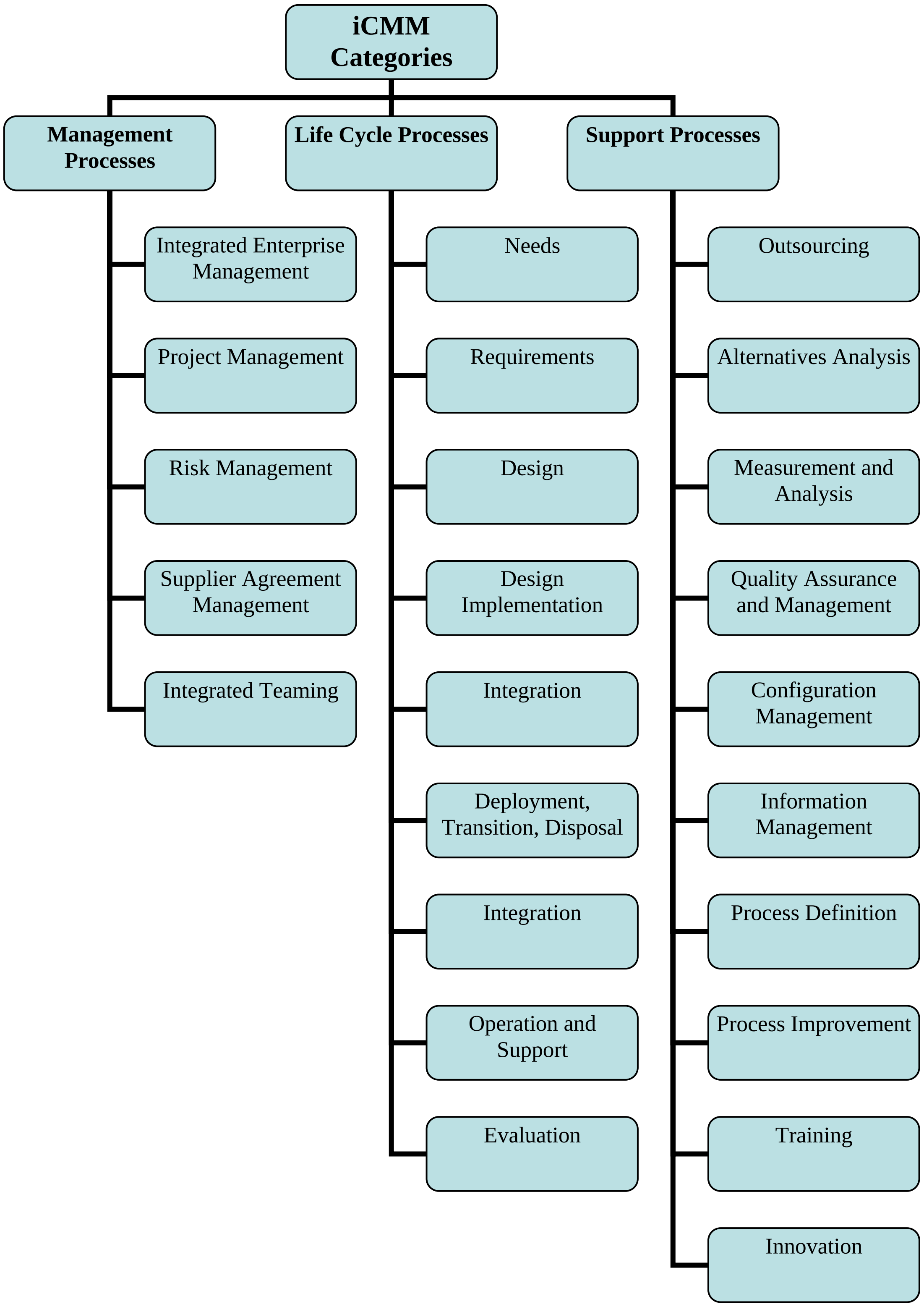


Figure 2: Process Areas of the FAA-iCMM

2.3 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 conducted experiments using the TSM to determine if 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.

2.4 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 [Redwine 04]. The SSE-CMM has been adopted as the ISO/IEC 21827 standard. Further information about the model is available at http://www.sse-cmm.org.

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: Security Engineering, and Project and Organizational processes. Security Engineering in turn is organized into Engineering Processes, Assurance Processes, and Risk Processes. There are 22 Process Areas distributed among the three categories. Each Process Area is composed of a related set of process goals and activities. The International Systems Security Engineering Association (ISSEA) maintains the SSE-CMM.

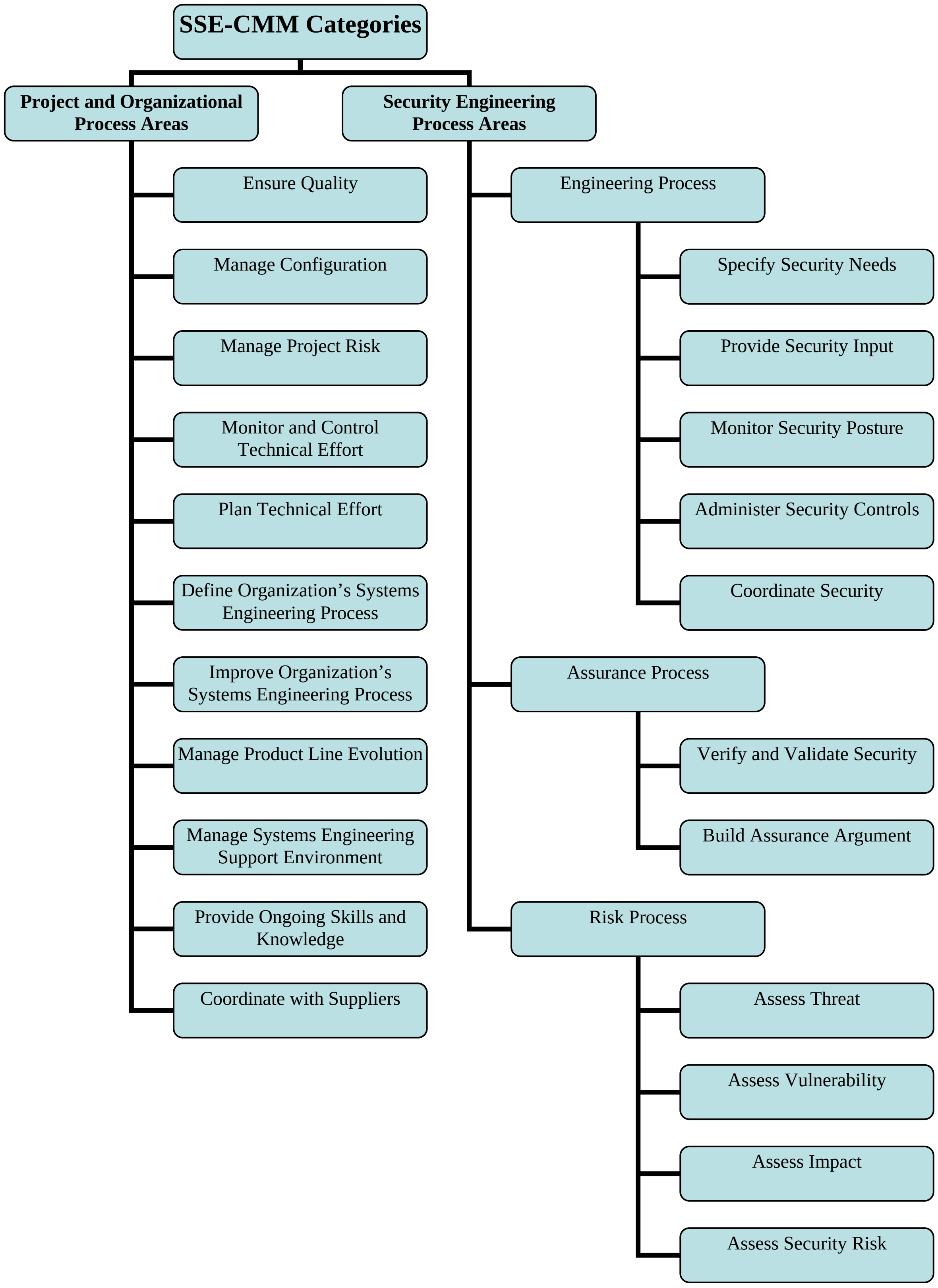

SSE-CMM Categories
Project and Organizational Process Areas
Security Engineering Process Areas
Ensure Quality
Manage Configuration
Manage Project Risk
Monitor and Control Technical Effort
Plan Technical Effort
Define Organization's Systems Engineering Process
Improve Organization's Systems Engineering Process
Manage Product Line Evolution
Manage Systems Engineering Support Environment
Provide Ongoing Skills and Knowledge
Coordinate with Suppliers
Engineering Process
Specify Security Needs
Provide Security Input
Monitor Security Posture
Administer Security Controls
Coordinate Security
Assurance Process
Verify and Validate Security
Build Assurance Argument
Risk Process
Assess Threat
Assess Vulnerability
Assess Impact
Assess Security Risk

Figure 3: Process Areas of the SSE-CMM

2.5 Proposed Safety and Security Additions to the CMMI and FAA-iCMM

Because of the integration of process disciplines and coverage of enterprise issues, the CMMI and the FAA-iCMM are used by more organizations than the SSE-CMM; yet the two integrated models contain gaps in their coverage of safety and security. As a result, some organizations within the FAA and the Department of Defense (DoD) have sponsored a joint effort to identify best safety and security practices for use in combination with the FAA-iCMM and the CMMI. The proposed Safety and Security additions to the FAA-iCMM and the CMMI identify 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 16 practices:

1. **Goal 1** – An infrastructure for safety and security is established and maintained.
    a. Ensure safety and security awareness, guidance, and competency.
    b. Establish and maintain a qualified work environment that meets safety and security needs.
    c. Ensure integrity of information by providing for its storage and protection, and controlling access and distribution of information.
    d. Monitor, report, and analyze safety and security incidents and identify potential corrective actions.
    e. Plan and provide for continuity of activities with contingencies for threats and hazards to operations and the infrastructure.
2. **Goal 2** – Safety and security risks are identified and managed.
    a. Identify risks and sources of risks attributable to vulnerabilities, security threats, and safety hazards.
    b. For each risk associated with safety or security, determine the causal factors, estimate the consequence and likelihood of an occurrence, and determine relative priority.
    c. For each risk associated with safety or security, determine, implement and monitor the risk mitigation plan to achieve an acceptable level of risk.
3. **Goal 3** – Safety and security requirements are satisfied.
    a. Identify and document applicable regulatory requirements, laws, standards, policies, and acceptable levels of safety and security.
    b. Establish and maintain safety and security requirements, including integrity levels, and design the product or service to meet them.
    c. Objectively verify and validate work products and delivered products and services to assure safety and security requirements have been achieved and fulfill intended use.
    d. Establish and maintain safety and security assurance arguments and supporting evidence throughout the life cycle.

4. **Goal 4** – Activities and products are managed to achieve safety and security requirements and objectives.
    a. Establish and maintain independent reporting of safety and security status and issues.
    b. Establish and maintain a plan to achieve safety and security requirements and objectives.
    c. Select and manage products and suppliers using safety and security criteria.
    d. Measure, monitor and review safety and security activities against plans, control products, take corrective action, and improve processes.

Further information about the proposed safety and security additions is available at http://www.faa.gov/ipg.

3 Additional Processes, Process Models, and Methodologies

3.1 Microsoft's Trustworthy Computing Security Development Lifecycle

Microsoft's Trustworthy Computing Security Development Lifecycle (SDL) is a process that they have adopted for the development of software that needs to withstand security attacks. 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 activity 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 [Lipner 05].

Microsoft has augmented the SDL with mandatory security training for its software development personnel, 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.

3.2 Team Software Process for Secure Software Development

The SEI's 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 [Humphrey 02]. Software produced using 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 [Davis 03].

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 CERT® program. The principal goal of the project is to develop a TSP-

® CERT is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

based method that can predictably produce secure software. TSP-Secure addresses secure software development in multiple ways:

- Secure software is not built by accident – TSP-Secure addresses how to plan for security
- Schedule pressures and inter-personal issues can get in the way of implementing best practices – TSP-Secure helps to build self-directed development teams and then expects these teams to manage their own work
- Security and quality are closely related – TSP-Secure helps manage quality throughout the product development life cycle.
- People developing secure software must have an awareness of software security issues – TSP-Secure includes security awareness training for developers

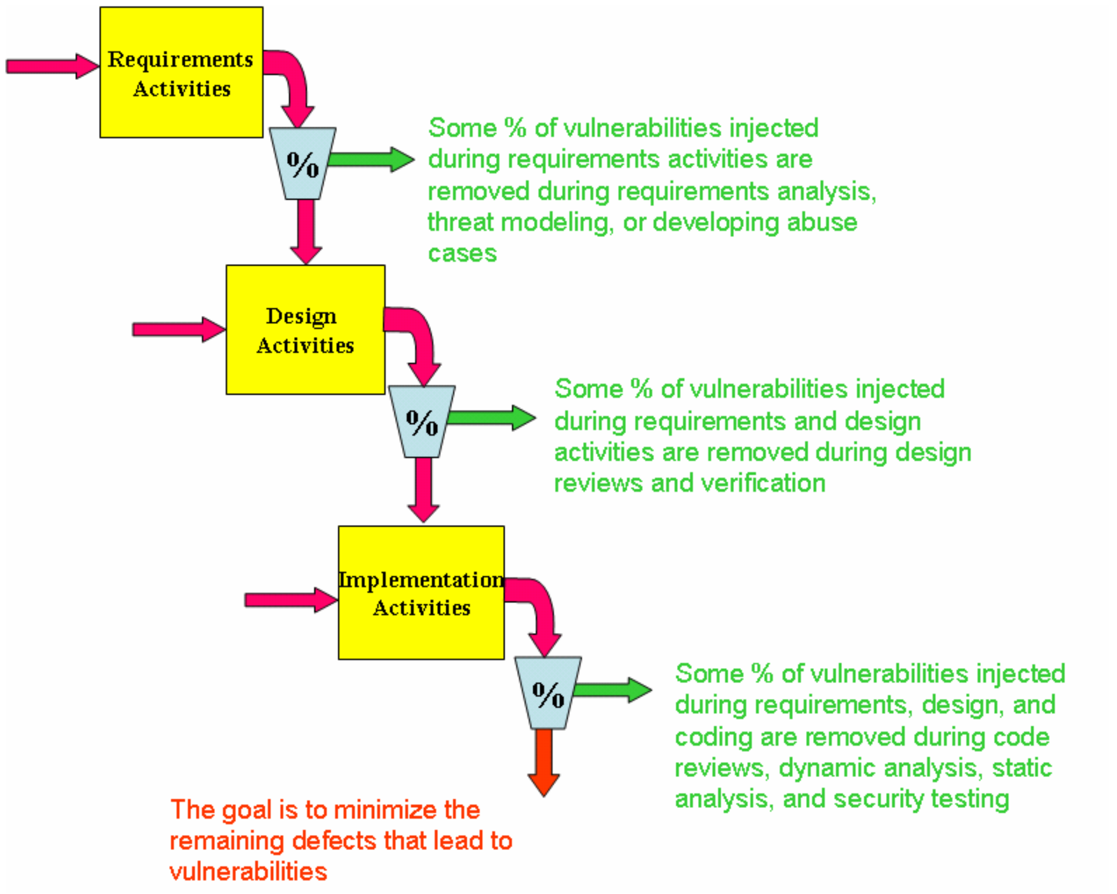
Teams using TSP-Secure create their own plans. Initial planning is conducted during a series of meetings called a “project launch,” which takes place over a three– to four–day period. The launch is led by a qualified team coach. During a TSP-Secure launch, the team reaches a common understanding of the 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 requirements, secure design and code reviews, use of static analysis tools, unit tests, and fuzz testing. (Fuzz testing is thought to enhance software security and software safety because it often finds odd oversights and defects which human testers would fail to find, and even careful human test designers would fail to create tests for [Wikipedia 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. He or she ensures that the product is statically and dynamically assured, provides timely analysis and warning about security problems, and tracks 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 all security-related activities take place. Security status is presented and discussed during every management status briefing.

Visits to web sites such as the SANS Institutes 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) [SANS 05, CVE 05, US-CERT 05, Microsoft 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 their root causes 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, as shown in Figure 4. The more defect removal filters there are in the software development life cycle, the fewer defects that can lead to vulnerabilities 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.



Requirements Activities
%
Some % of vulnerabilities injected during requirements activities are removed during requirements analysis, threat modeling, or developing abuse cases
Design Activities
%
Some % of vulnerabilities injected during requirements and design activities are removed during design reviews and verification
Implementation Activities
%
Some % of vulnerabilities injected during requirements, design, and coding are removed during code reviews, dynamic analysis, static analysis, and security testing
The goal is to minimize the remaining defects that lead to vulnerabilities

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 with regard to their goals, helps them decide whether to move to the next step or to stop and take corrective action, and indicates where to correct their process in order to meet their goals.

The TSP-Secure 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?

In addition, the TSP-Secure method includes training for developers, managers, and other team members that specifically focuses on security awareness.

3.3 Correctness by Construction

The Correctness by Construction methodology developed by Praxis Critical 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. Correctness by Construction 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. Specifically, the process almost always uses formal methods to specify behavioral, security, and safety properties of the software. The belief is 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:

    a. 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.

    b. 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. Requirements are specified using the formal specification notation Z (pronounced “zed”) and is verified. Code is written in SPARK (a subset of Ada), which can be statically assured and is then checked by verification software.

3.4 Agile Methods

Over the past few years, a new family of software engineering methods has started to gain acceptance in the software development community. These methods, collectively called Agile Methods, conform to the *Manifesto for Agile Software Development*, 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” [Agile Alliance 01].

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 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 classified existing software assurance activities into four categories: those that are a natural match for Agile methods, those that are independent of any development methodology, those that can be automated or semi-automated so that they could be incorporated into Agile methods, and those that are fundamentally mismatched with Agile methods. Table 1 (included with permission from the authors) shows that almost 50% of traditional security assurance activities are not compatible with Agile Methods (12 out of 26 mismatches), less than 10% are natural fits (2 out of 26 matches), 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		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 testing			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 [Poppendieck 02, Beznosov 03, Wäyrynen 04].

3.5 The Common Criteria

In January 1996, Canada, France, Germany, the Netherlands, the United Kingdom, and the United States released a jointly developed security evaluation standard. This standard is known as the “Common Criteria for Information Technology Security Evaluation,” but is more often referred to just 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 security 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 from which product users make a selection. 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 it is likely that 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 is an implementation-dependent statement of security needs for a specific product.

The PPs and the ST 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 PP. The organization then has the PP evaluated, and publishes it.
2. A product developer uses this PP to write an ST that complies with it and then has the ST evaluated.
3. The product developer then builds a Target of Evaluation (TOE) (or uses an existing one) and has it evaluated against the ST.

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

A current list of validated products and their associated EAL levels is available at http://niap.nist.gov/cc-scheme/vpl/vpl_type.html.

4 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]

This technical note demonstrates that although there are several processes and methodologies that could support secure software development, very few are designed specifically to address software security from the ground up. The notable exceptions are Microsoft's Trustworthy Computing SDL and the SSE-CMM. As software security becomes a more important issue in an increasingly networked world, more processes that explicitly address the four focus areas identified in this paper (security engineering activities, security assurance activities, security organizational and project management activities, and security risk identification and management activities) should achieve visibility.

Bibliography

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

[Agile Alliance 01]	The Agile Alliance. *Manifesto for Agile Software Development*. http://agilemanifesto.org (2001).
[Beznosov 03]	Beznosov, Konstantin. *eXtreme Security Engineering: On Employing XP Practices to Achieve 'Good Enough Security' without Defining It.* http://konstantin.beznosov.net/professional/papers /eXtreme_Security_Engineering.html (2003).
[Beznosov 05]	Beznosov, Konstantin & Kruchten, Philippe. *Towards Agile Security Assurance.* http://konstantin.beznosov.net/professional/papers /Towards_Agile_Security_Assurance.html (2004).
[CC 05]	Common Criteria. http://www.commoncriteriaportal.org/ (2005).
[CVE 05]	Common Vulnerabilities and Exposures. http://www.cve.mitre.org/ (2005).
[Davis 03]	Davis, Noopur & Mullaney, Julia. The Team Software Process (TSP) in Practice: A Summary of Recent Results (CMU/SEI-2003-TR-014, ADA418430). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/publications/documents/03.reports /03tr014.html
[DHS 05]	Department of Homeland Security. *Build Security In.* https://buildsecurityin.us-cert.gov/portal/ (2005).
[FAA 01]	Federal Aviation Administration. *The Federal Aviation Administration Integrated Capability Maturity Model® (FAA-iCMM®), Version 2.0*. Washington, DC: Federal Aviation Administration, September 2001. http://www.faa.gov/aio/common/documents/iCMM /FAA-iCMMv2.htm

[Goldenson 03]	Goldenson, Dennis R. & Gibson, Diane L. *Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results* (CMU/SEI-2003-SR-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/publications/documents/03.reports /03sr009.html
[Hall 02]	Hall, Anthony & Chapman, Roderick. “Correctness by Construction: Developing a Commercial Secure System.” *IEEE Software 19*, 1 (January/February 2002): 18–25.
[Herbsleb 94]	Herbsleb J.; Carlton, 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. http://www.sei.cmu.edu/publications/documents/94.reports /94.tr.013.html
[Humphrey 02]	Humphrey, Watts S. *Winning with Software: An Executive Strategy.* Boston, MA: Addison Wesley, 2002 (ISBN 0201776391).
[IEEE 91]	IEEE. *IEEE Standard Glossary of Software Engineering Terminology.* ANSI/IEEE Std 610.12-1990. February 1991.
[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.* http://msdn.microsoft.com/security/default.aspx?pull=/library /en-us/dnsecure/html/sdl.asp (2005).
[McAndrews 00]	McAndrews, Donald. *The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices* (CMU/SEI-2000-TR-015, ADA387260). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. http://www.sei.cmu.edu/publications/documents/00.reports /00tr015.html

[Microsoft 05]	Microsoft. Microsoft Security Advisories. http://www.microsoft.com/technet/security/advisory/default.mspx (2005).
[Mills 87]	Mills, H.; Dyer, M.; & Linger, R. "Cleanroom Software Engineering." *IEEE Software 4*, 5 (September 1987): 19–25.
[NASA 89]	NASA. *Software Assurance Guidebook, NASA-GB-A201*. http://satc.gsfc.nasa.gov/assure/agb.txt (1989).
[Paulk 93]	Paulk, M.; Curtis, B.; Chrissis, M.; & Weber, C. "The Capability Maturity Model for Software (Version 1.1) (CMU/SEI-93-TR-24). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993. http://www.sei.cmu.edu/publications/documents/93.reports /93.tr.024.html
[Poppendieck 02]	Poppendieck, M. & Morsicato, R. "Using XP for Safety-Critical Software." *Cutter IT Journal 15*, 9 (2002): 12–16.
[Redwine 04]	Redwine, Samuel T. & Davis, Noopur, Eds. *Processes to Produce Secure Software: Towards More Secure Software, Volume II.* http://www.cyberpartnership.org/Software%20Pro.pdf (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.* http://www.sans.org/top20/ (2005).
[SEI 05a]	Software Engineering Institute. *Process Maturity Profile: CMMI v1.1 SCAMPI v1.1 Class A Appraisal Results, 2005 Mid-Year Update*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, September 2005. http://www.sei.cmu.edu /appraisal-program/profile/pdf/CMMI/2005sepCMMI.pdf
[SEI 05b]	Software Engineering Institute. *Process Maturity Profile Software CMM 2005 Mid- Year Update*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, September 2005. http://www.sei.cmu.edu/appraisal-program/profile/pdf /SW-CMM/2005sepSwCMM.pdf

[US-CERT 05] United States Computer Emergency Readiness Team. *Technical Cyber Security Alerts*. http://www.us-cert.gov/cas/techalerts/ (2005).

[Wäyrynen 04] Wäyrynen, J.; Bodén, M.; & Boström, G. “Security Engineering and eXtreme Programming: an Impossible Marriage?” *Proceedings of XP/Agile Universe 2004: 4th Conference on Extreme Programming and Agile Methods*, Calgary, Canada, August 15–18, 2004. Germany: Springer-Verlag, 2004.

[Wikipedia 05] Wikipedia, The Free Encyclopedia. Definition of *Fuzz testing*. http://en.wikipedia.org/wiki/Fuzz_testing (2005).

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE December 2005	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Secure Software Development Life Cycle Processes: A Technology Scouting Report		5. FUNDING NUMBERS FA8721-05-C-0003
6. AUTHOR(S) Noopur Davis		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2005-TN-024
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

As the use of the Internet and networked systems become more pervasive, the importance of developing secure software increases. The purpose of this technical note is to present overview information about existing processes, standards, life cycle models, frameworks, and methodologies that support or could support secure software development. Where applicable and possible, some evaluation or judgment is provided.

The target audience for this technical note includes software engineering process group (SEPG) members, software developers, and managers seeking information about existing software development life cycle (SDLC) processes that address security.

14. SUBJECT TERMS CMM, CMMI, process, process area, process improvement, maturity, maturity model, maturity profile, methodology, metric, secure system, software development, software engineer, SW-CMM, test, TSP			15. NUMBER OF PAGES 38
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL