Secure Software Education – A Contextual Model-Based Approach

J.J. Simpson, *M.J. Simpson, B. Endicott-Popovsky, V. Popovsky

August 25th, 2010

ABSTRACT

This article establishes a context for secure information systems development as well as a set of models used to develop and apply a secure software production pedagogy. A generic system model is presented to support the system context development, and to provide a framework for discussing security relationships that exist between and among information systems and their applications. An asset protection model is tailored to provide a conceptual ontology for secure information system topics, and a stable logical framework that is independent of specific organizations, technologies and their associated changes. This asset protection model provides a unique focus for each of the three primary professional communities associated with the development and operation of secure information systems: the systems/software engineering, information assurance, and the legal/justice/intelligence communities. It is also a vehicle for structured interfaces among these groups. A secure adaptive response model is discussed to provide an analytical tool to assess risk associated with the development and deployment of secure information systems and a security metric with which to determine coverage of topics to address and mitigate those risks. A pedagogical model for information assurance curriculum development is then established in the context and terms of the developed secure information system models. The relevance of secure coding techniques to the production of secure systems, architectures and organizational operations is also discussed.

KEY WORDS

Asset protection model, software security, secure information systems, system security, threat model, target model, system model, pedagogical model for information assurance, McCumber Cube, risk assessment.

INTRODUCTION

Within the software engineering community, there is an increasing recognition that secure coding practices are only a subset of the activities needed to create secure information systems. Information systems, including the software, hardware, and people that contribute to those systems, continue to change and adapt to new technologies and science. This paper is organized around the fundamental ideas that (1) all system and software security exists in an adaptable system context, and (2) a range of conceptual models are necessary to organize, discuss and understand these adaptable security aspects. The practice of secure information systems design, development, deployment and operation is shared by three professional communities, the systems engineering (including software engineering) community, the information assurance community, and the justice and intelligence communities. A generic system model is introduced and combined with a comprehensive, layered asset protection model to establish an abstract set of security concepts that are independent of any specific technology and/or application approach. The generic system model provides a common basis for the communication within the systems/software engineering communities regarding secure information systems. The asset protection model provides a focus point for each of the professional communities, and supports the clear communication of information associated with secure information systems.

As these two models are interrelated and expanded in a combined system security model, operational connections become evident. System operational effectiveness as well as operational suitability are defined, developed and discussed as they relate to security and security education. Potential information system risks need to be understood and addressed. To this end, a system security metric is introduced that enables an analysis of what risks might be present during the development and deployment of secure information systems. This tool also introduces a method to help determine what topics might need to be addressed within secure information systems or secure coding practices curriculum in order to mitigate those risks. This set of security concepts are then integrated with a pedagogical model for information assurance curriculum development. . Common system attack patterns, as well as software weaknesses and vulnerabilities, are used as examples to illustrate the processes necessary to increase software quality by improving software security education and development. Standard secure software approaches are mapped over the combined system security model, and discussed in the proper system context. A rich conceptual topology is developed and used to frame and communicate the many aspects of secure software engineering education. Next the generic system model will be introduced and outlined to support the introduction and discussion of the asset protection model.

GENERIC SYSTEM MODEL

The practice of systems engineering has produced a number of technical, organizational and process-based approaches to the solution of large-scale, socio-technical engineering and process problems. One of the key aspects associated with systems engineering is the development of system context models and system functional models. The Generic System Model (GSM) is based on the fundamental idea of a system boundary that distinguishes a boundary between inside the system and outside the system. The system context exists outside of the system boundary; the system concept is used to organize the internal system content. The system boundary is composed of an outward-looking portion called the boundary context, and an inward-looking portion called the boundary concept that captures the controlling system values, rule sets, and structural view. As depicted in Figure 1, a specific system is composed of system functions, requirements, architecture and tests (Simpson, 2004).

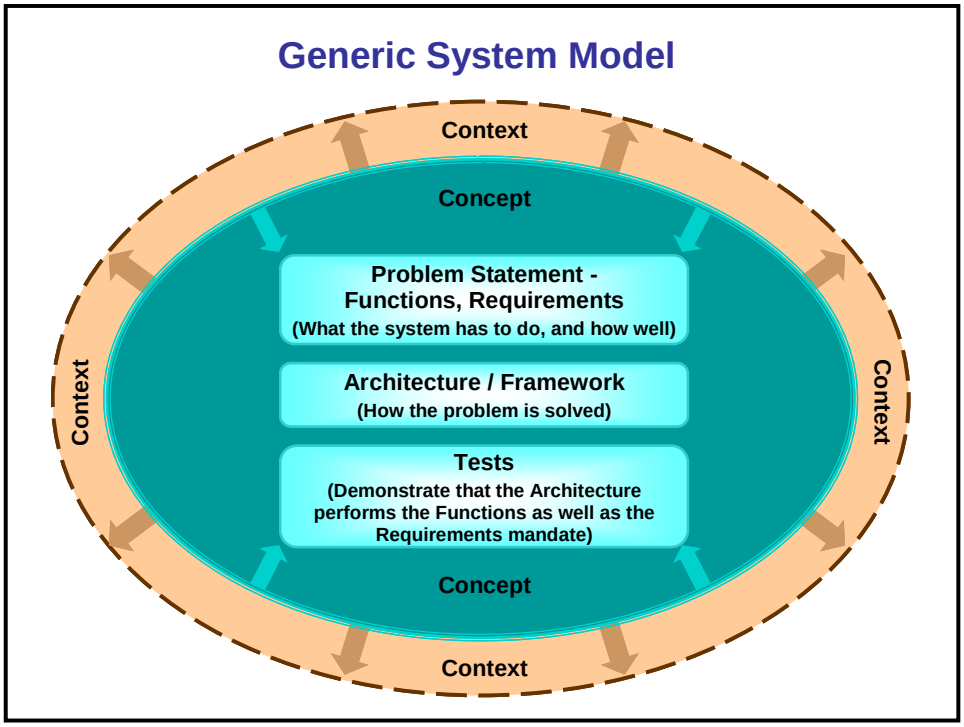

Figure 1. Generic System Model (Adapted from Figure 5, Simpson, 2004)

System functions are the actions or activities that the system is designed to perform. System requirements describe how well the system must do its required functions. In combination, the system functions and the system requirements create the system problem statement. This fact further defines the system concept. The system architecture is the mechanism that performs the system functions as well as stated by the system requirements. Successful system tests indicate that the system architecture performs the functions as well as the requirements stipulate. This Generic System Model is used to develop the structured process and control mechanisms needed to effectively address the many aspects that impact the production and deployment of a secure information system. When a software intensive system is defined at an architectural level, the resulting definition details the functional interactions between the manual-based, hardware-based and software-based processes. Consequently, system security and functional requirements can be allocated and assigned to these respective processes, and documented and recorded at the system level. Then, system security functions and requirements can be evaluated for implementation at (1) a manual, administrative level, (2) a software level, or (3) a hardware level. Using this approach, the system security functions are clearly identified, and assigned to a system architectural element that will perform the system security function. Further, each system component is either assigned a documented role in the system security function, or is documented as having no role in the system security function.

Generally, the practice of system security requirements elicitation is not well integrated into the development of software intensive systems. This is largely the result of environmental and contextual factors that impact the security of a deployed, operational system. These factors are commonly outside the scope and/or control of the software development process. As a consequence, typical security deployment assumptions are not well documented or communicated. The Security Quality Requirements Engineering (SQUARE) process model was developed to address some of the software definition and production issues associated with development of secure software systems (Meade, 2010). However, a large set of issues

associated with a more comprehensive specification of the security context and security requirements remain for any given software intensive system. The Generic System Model is applied to help address primary system security context and requirements issues, by defining a general set of systems within which security mechanisms can be applied, maintained, and evaluated.

GENERIC SYSTEM MODEL, AND THREE PRIMARY SYSTEMS

The Generic System Model is used to describe three primary systems: (1) the product system, (2) the process system and (3) the environment system (Mar & Morais, 2002; Simpson, 2002). Based on a very simplified model of industrial production, these three systems are identified and defined by first locating the product system which is the output of the industrial process. The process system is the system that produces the product system. When considering software intensive systems, the environment system contains both the process system that creates the software product, as well as the software product system (see Figure 2). The distinctions between system production, and system operation and maintenance, must be well defined and clearly understood.

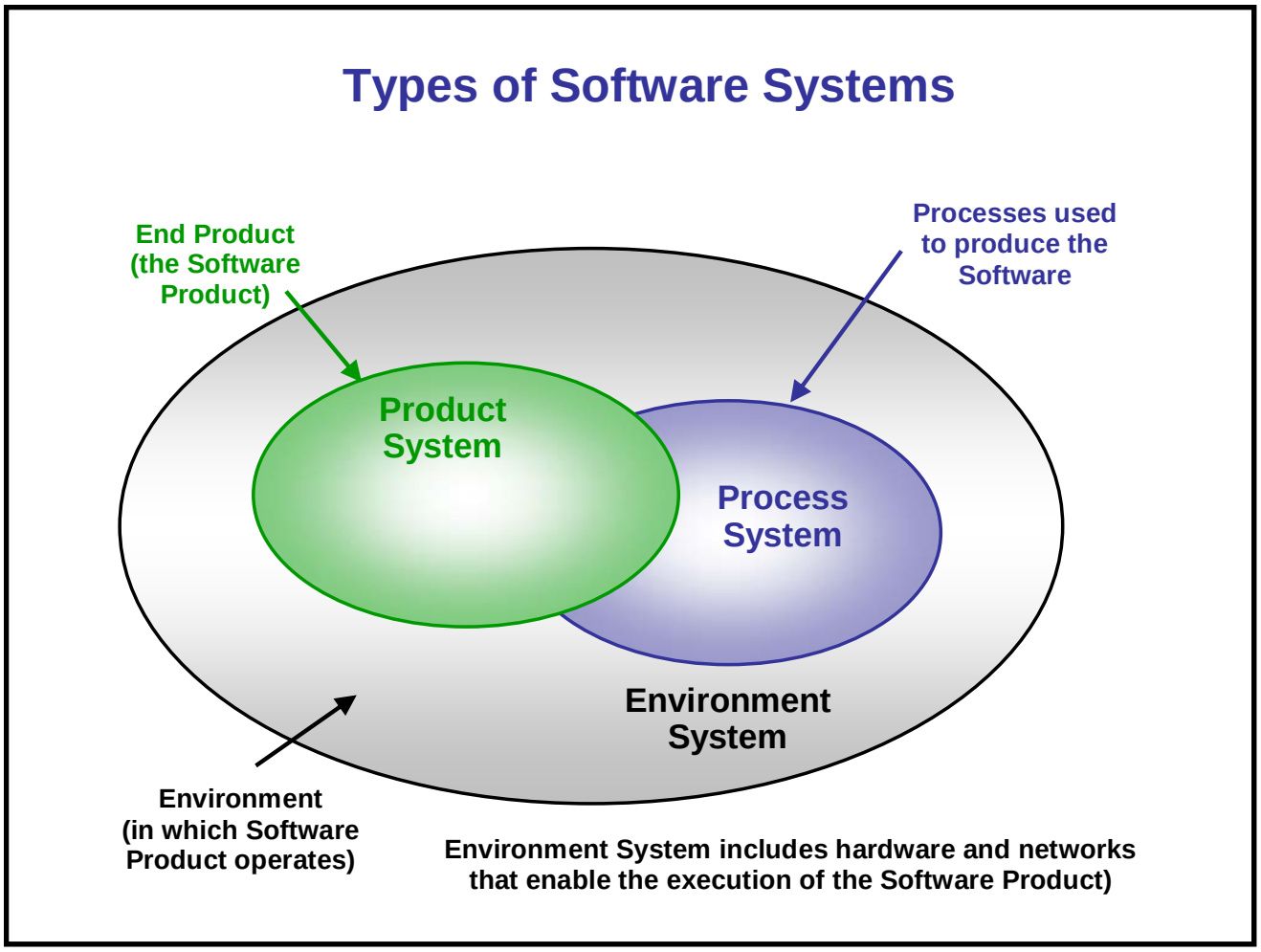

Figure 2. Types of Systems. (Adapted from Figure 6a, Simpson & Simpson, 2003)

The particular type of software product system being developed will determine which software development and deployment processes can be used to increase the security of the deployed software. The process system includes the development and test system used to produce and deploy the software product. An active software security testing and patching process also must be part of any effective, secure software deployment. The process system can be controlled and managed using a wide variety of software development processes and techniques. Within the context of the software product, the environment system contains the hardware and networks that support and enable the execution of the software product. The environment system also contains the organizational and administrative personnel that use and maintain the operational system. System and software security threats also reside in the system environment. Many of the most important system and software security considerations are dependent on factors that exist in the system environment. These

external security considerations have a direct impact on, and are reflected in the system architecture.

The dynamic and adaptive nature of the system environment establishes the need for a comprehensive, holistic view of the information system security problem. An expanded information system security model is used to effectively identify and communicate the relationship between the system, threats, and targets that exist in the environmental system.

INFORMATION ASSET PROTECTION MODEL

An asset protection model (APM) has been developed to organize and communicate the basic aspects and components of overall system security (Simpson & Simpson, June 7, 2010). The asset protection model intentionally is designed to be independent of specific organizations and technologies – entities that experience high rates of change over time. As a result, the asset protection model will remain stable for an extended period of time. Two additional features of the APM design establish real value to the secure information systems community; it supports human short-term cognition and methods for computer-enhanced reasoning. Warfield (1990) provides a reasoned basis for the application of behavioral pathology studies related to the span of immediate recall performed by Miller in 1956 at Princeton University, and to the magical number 7 by Herbert Simon in 1974 at Carnegie Mellon University. Warfield goes on to establish a law and its corollary within systems science.

> The Law of Triadic Compatibility quantifies the limitations of short-term memory as they related to human decision making: The human mind is compatible with the demand to explore interactions among a set of three elements, because it can recall and operate with seven concepts, these being the three elements and their four combinations; but capacity cannot be presumed for a set that has both four members and for which those members interact.
> A Corollary to this Law is the Principle of Division by Threes. This principle asserts that: Iterative division of a concept as a means of analysis is mind compatible if each division produces at most three components, thereby creating a tree with one element at the top, at most three elements at the second level, at most nine at the third level, and so on. (Warfield, 1990, p. 45)

As designed, the asset protection model supports recursive definition of levels of abstraction, allowing multiple methods of computer-enhanced reasoning to be applied. These methods include dynamic analysis between interfaces as well as contained and internal controls within any given area of focus.

The asset protection model within this paper is focused on information as its target. With individual nodes patterned after the structure and form of the McCumber Cube, the APM presents a hierarchy of logical abstractions and associated categories designed to decrease the chance of cognitive overload for individuals that must think and reason about asset protection issues from different points of view. The APM is depicted notionally in Figure 3. The asset protection model is organized in a hierarchy of concepts and logical categories starting with the highest level of abstraction at the top of the hierarchy named APM Level Zero. At Level Zero there is one cube, the asset cube, constructed from the three logical components associated with asset protection: (1) the system, (2) the target, and (3) the threat. These three logical components were selected to represent and focus the attention of the three primary professional communities associated with information asset protection: the systems engineering community (system), the information assurance community (target), and the legal, justice and intelligence community (threat). Each of these components is further elaborated at the next level of abstraction - Level One. At APM Level One, three cubes are used to further define the asset protection model in a more detailed manner: the system cube, the target cube and the threat cube.

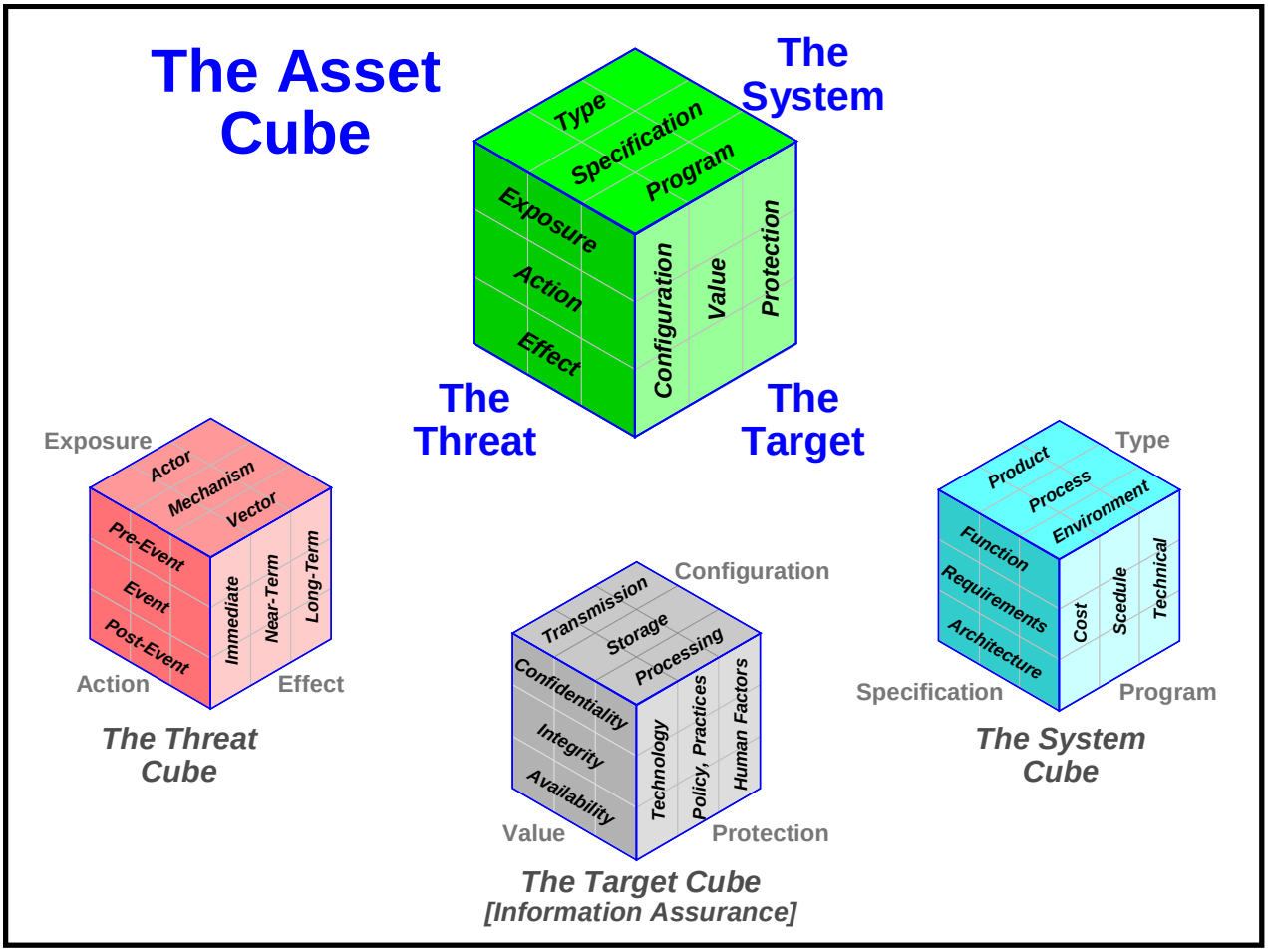

Figure 3 Asset Protection Model. (Adapted from Figure 3 of Simpson & Simpson, 2010)

The system cube is developed to address the system type, system specification and system program:

- System type is divided into three systems types: the product system, the production process system, and the environment system. Each of these three system types represent an important system category that must be evaluated during a secure system and software requirements development activity. Many system and software vulnerabilities are found at the interfaces between these system types. These vulnerabilities may provide the foundation for, and support of, a successful breach of the system security. As a result, the definition of each system type as well as the operational role assigned to each system and/or system component is a basic first step in the design and analysis of any given security problem.
- System specification is divided into three specification types: functions, requirements and architecture. Each of these individual system specification types represents an important area of security evaluation during the design, deployment, evaluation and test of a secure software-intensive system. Standard system and software specification practices are used to manage the relationship between and among these system specification types. However, many security vulnerabilities are introduced in deployed systems because of the wide variety of specification, requirement and configuration management systems. These different systems are often organized around (1) the software system, (2) the hardware system, or (3) the integrated deployed system. Incompatibility between system specification management systems is driving the development of system security-based life-cycle practices that are different from the standard specification practices. The Microsoft Security Development Lifecycle process is an example of this type of stand-alone security life-cycle process.
- System program is divided into program control types: cost, schedule and technical. These program management types each have well known metrics and procedures that are used to manage programs. One key aspect of program management that drives the introduction of system and software vulnerabilities is the lack of well-communicated, well-understood metrics for system and software security. The secure adaptive response potential (SARP) was developed to address this lack of program level security metrics (Simpson, J. J., Miller, A. & Dagli, C., 2008, June). SARP can be used to guide programmatic decision and resource allocation. Methods for using and applying SARP will be discussed in more detail later in the paper.

The target cube, at APM Level One, is expanded to target configuration, target value, and target protection. The target cube is patterned after McCumber cube, a well-known information assurance security evaluation model (McCumber, 2005). Every one of these target components is the source of essential software security requirements, and must be considered individually and in combination with the other factors.

- Target configuration is applied to a well-defined information system that has an established boundary and is divided into three configuration types: transmission, storage and processing. The transmission component addresses information that is in the process of being transmitted between or among the other parts of the information system of interest. The storage component is designed to identify information that is at rest in the information system. The processing component is focused on the information that is being transformed by a computer based algorithm, with one kind of information being the input to the algorithm process and a new kind of information being the output from the algorithm process. It should be noted that encryption may be used in the transmission and storage states but cannot be used in the processing phase of information system operations.
- Target value is divided into three types: confidentiality, integrity, and availability. The confidentiality component is a basic information assurance security value and assures that only authorized agents (people and processes) have access to the information when needed. The integrity component is another foundational information assurance security value, and is associated with the fact that information must be accurate and maintain a known configuration. The availability component addresses the value produced when information can be accessed when needed. A significant system vulnerability exists when any of these foundational information assurance system security values is missing. As integrated operational values, the vulnerabilities in this area can be introduced from many sectors, including operational policy, system design, system architecture, software development and software deployment configurations.
- Target protection is divided into three types: technology, policy and human factors. The technology component is the primary protection method used to ensure that the critical information values are maintained in any information configuration and/or state. The policy component is focused on the use of organizational and system based policies and practices that are enforced to enable the protection of critical information. The human factors component is

The threat cube, at APM Level One, is defined by the threat exposure, threat action and threat effect. Each of these specific threat areas is key characteristics that drive the design and development of secure systems, software architecture, and software systems.

- Threat exposure is divided into three threat types: actor, mechanism and vector. The actor component addresses the autonomous agent that controls and guides the application of a specific threat. The mechanism component describes the process and equipment that is used to generate and maintain the threat behavior. The vector component identifies the channel and/or pathway used to deliver and/or activate the threat mechanism.
- Threat action is divided into three types: pre-event, event and post-event. The pre-event component is focused on system configuration and behavior prior to the initiation of any threat action. The event component is used to define a specific type of threat action. The post-event component is focused on system configuration and behavior after the threat action has occurred. The pre-event component would address actions necessary to prepare an information system to be "forensic ready." While the post-event component would address the actions necessary to successfully take legal action against a threat actor in a court of law.
- Threat effect is divided into three components: immediate, near-term and long-term. The immediate effect addresses instant impact created by a successful attack on an information system. The near-term effect is focused on identifying impacts to the organization and/or information system that happen between one and thirty days. The long-term effect is used to analyze and evaluate impacts and changes to the organization and/or information systems that happen after 30 days.

As a focal point for the legal community, the threat cube establishes an area wherein the system requirements needed to support effective legal action against threat actors can be addressed.

The APM provides a rich set of conceptual categories that cover the complete range of information system and software security concerns including aspects of program management, technological, and operational considerations associated with secure system and software development. The secure adaptive response potential (SARP) metric, used to evaluate system and software security, has components that closely correlate with the APM components. System security operations are evaluated using the SARP metric components that generally describe the operation and maintenance profile of the deployed system. SARP's key components are organizational, technical, operational, and context:

- The SARP organizational component includes consideration of security training, policy, resources, culture, values and resource priority.
- The SARP technical component considers cryptographic strength, mean-time-to-attack, attack tree probabilities, intrusion detection, intrusion prevention, and software metrics.
- The SARP operational component evaluates stability, safety, maintainability, reliability, confidentiality, integrity, availability and authentication.
- The SARP context component covers external threat, internal threat, threat monitoring, external stability and internal stability.

Integrating and combining the values associated with each of these SARP components provides useful system security measures that can be used to guide information system and software security decisions.

APM AND THE SQUARE PROCESS

The conceptual and logical structure provided by the Asset Protection Model provides a well-bounded system security domain that can provide clearly bounded scope for system and software security requirements engineering activities. The SQUARE process, introduced earlier in the paper, has nine basic steps that cover the complete secure software requirements development process. A synopsis of the way in which the APM supports the application of the SQUARE process is presented in Table 1.

Table 1. APM Contextual Support for SQUARE Process

SQUARE Process Steps	***APM Contextual Support***
Step 1: Agree on definitions	A. Gross indicates that "All communication takes place in a shared contextual space." (Gross, June 11, 2010). The activity of developing, identifying and gaining agreement on specific system security requirement definitions helps form and structure a shared contextual space for everyone participating in the SQUARE activity. In addition, the APM limits and bounds the valid concepts that can populate the shared contextual space. The APM also develops and reinforces a shared understanding of the secure software system requirements development contextual space.
Step 2: Identify security goals	The system security goals must be developed at a typical level of organizational deployment and operation, and/or a range of typical operational deployments. The definition of the deployment and operational environment will begin the process of scoping the operational and application security problem set. The APM assists in the identification of security goals by providing a well understood set of security constructs and concepts that can be used to define and record the needed set of security goals.
Step 3: Develop artifacts to support security requirements definition	The APM structure is used to identify, categorize and establish relationships between all requirement artifacts associated with the system development. The system component of the APM provides the framework upon which existing system requirements can be evaluated for system security impacts. The target component of the APM outlines the security configuration, protection and value characteristics that must be considered in the security artifact development and production. The threat component of the APM addresses possible exposure, effect and action that impact security requirements artifacts.
Step 4: Perform a risk assessment	The APM supports the activities of performing a risk assessment in a number of fundamental ways. The APM organizes the system information in terms of a system, threat and target. APM target content is based on a well known model for comprehensive system security and risk evaluation, the McCumber Cube (McCumber, 2005). This well known model supports a structured set of risk evaluation techniques and approaches.
Step 5: Select elicitation techniques	The well-balanced structure of the APM provides direct support for structured requirements elicitation techniques and processes.
Step 6: Elicit security requirements	The APM provides the structure, content organization and artifact framework necessary to support any type of structured security requirements elicitation and development process.
Step 7: Categorize requirements into levels (system, software) and type (requirements, constraints)	The APM system specification and program components directly support the evaluation and categorization of system security requirements. In a complete system development, the system software security requirements would be evaluated in the context of all the other system constraints and requirements.

SQUARE Process Steps	*APM Contextual Support*
Step 8: Prioritize requirements	The complete system scope as well as the attention to value, threat and function in the APM support the activity of requirements prioritization. The structured information and data associated with the APM can be used as direct inputs for formal ranking tools like the analytical hierarchy process (AHP) (Simpson, Miller, & Dagli, June, 2008).
Step 9: Inspect requirements	The inspection and evaluation of system and software security requirements is greatly aided using the APM structured framework that is conceptually organized around the fundamental concepts of the system security asset protection domain.

The SQUARE process is a valuable tool when it is deployed in an organizational environment that has strong process-based policies and practices (Meade, 2010). The APM works together with the SQUARE process to further define, refine, and structure the software security related activities needed to produce a secure software intensive deployed system.

ASSET PROTECTION MODEL, LEVEL ONE INTERFACES

The three APM cubes at Level One are designed to create three structured Level One interfaces. Depicted in Figure 4, these structured interfaces are: (1) the Threat-Target Interface, the Target-System Interface, and 3) the System-Threat Interface. These structured interfaces create a conceptual plane upon which the associated professional communities can arrange data and information in a manner that represents the identified items of interest for each type of interface exchange. In this manner, the structured interfaces support the development of operational security patterns by the professional communities. As noted earlier in the paper, the system cube is used to focus the information and expertise from the systems engineering community. The target cube is used to focus information, expertise, and all related aspects from the IA community. Similarly, the threat cube is designed to have the legal and justice community as its primary controlling group. While there may be strong conceptual overlap and/or conceptual conflicts in many areas of the APM, the model structure and the model interfaces are designed as areas where these conceptual conflicts can be effectively addressed. Operational security patterns associated with the APM interfaces are designed to support the development of suitable software and system security requirements that may be outside the detailed experience of the software developer and/or the system customer. General categories of system deployment patterns and system deployment contexts are used in the education of secure software engineers to identify the controlling contextual security aspects.

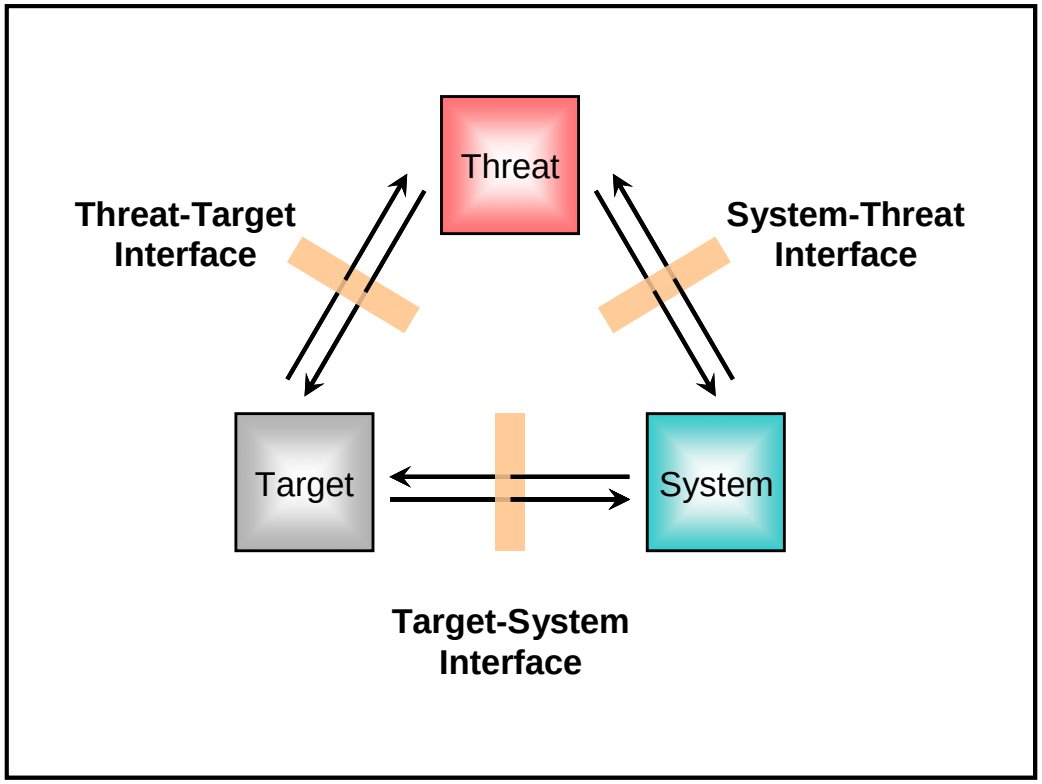

Figure 4. Asset Protection Model (APM) Level One Interfaces (Adapted from Presentation Slide 13, Simpson & Endicott-Popovsky, 2010)

Computer forensics is an example of the activity that takes place on the system-threat interface. While no fundamental software security requirements are generated directly from the threat cube, there are a large number of derived system and software security requirements associated with preparing a system to support successful criminal prosecution. The APM can be used to organize and evaluate system and security activities in a systematic fashion. This form of human and computer analysis highlights areas like the threat-system interface wherein an interdisciplinary combination of knowledge, skills, and abilities must be developed and applied. In the area of digital and computer forensics, system and software security topics associated with the development, deployment, and operation of networks, operating systems, file systems, and applications must be addressed. However, law and ethics topics must also be interwoven with the classical system and software topics to address rules of evidence, chain of custody of evidence, process and methods for search and seizure, as well as other items to prepare evidence for presentation in a court of law (Endicott-Popovsky, Popovsky & Frincke. June, 2004).

ASSET PROTECTION MODEL, LEVEL TWO INTERFACES

At APM Level Two, the system, target and threat components are further delineated into their three essential areas of focus, notionally depicted in Figure 5. Continuing with the systematic decomposition and evaluation of the component interfaces generates a well defined set of localized interaction areas that must be analyzed and evaluated to enhance the probability that all primary areas of concern have been addressed.

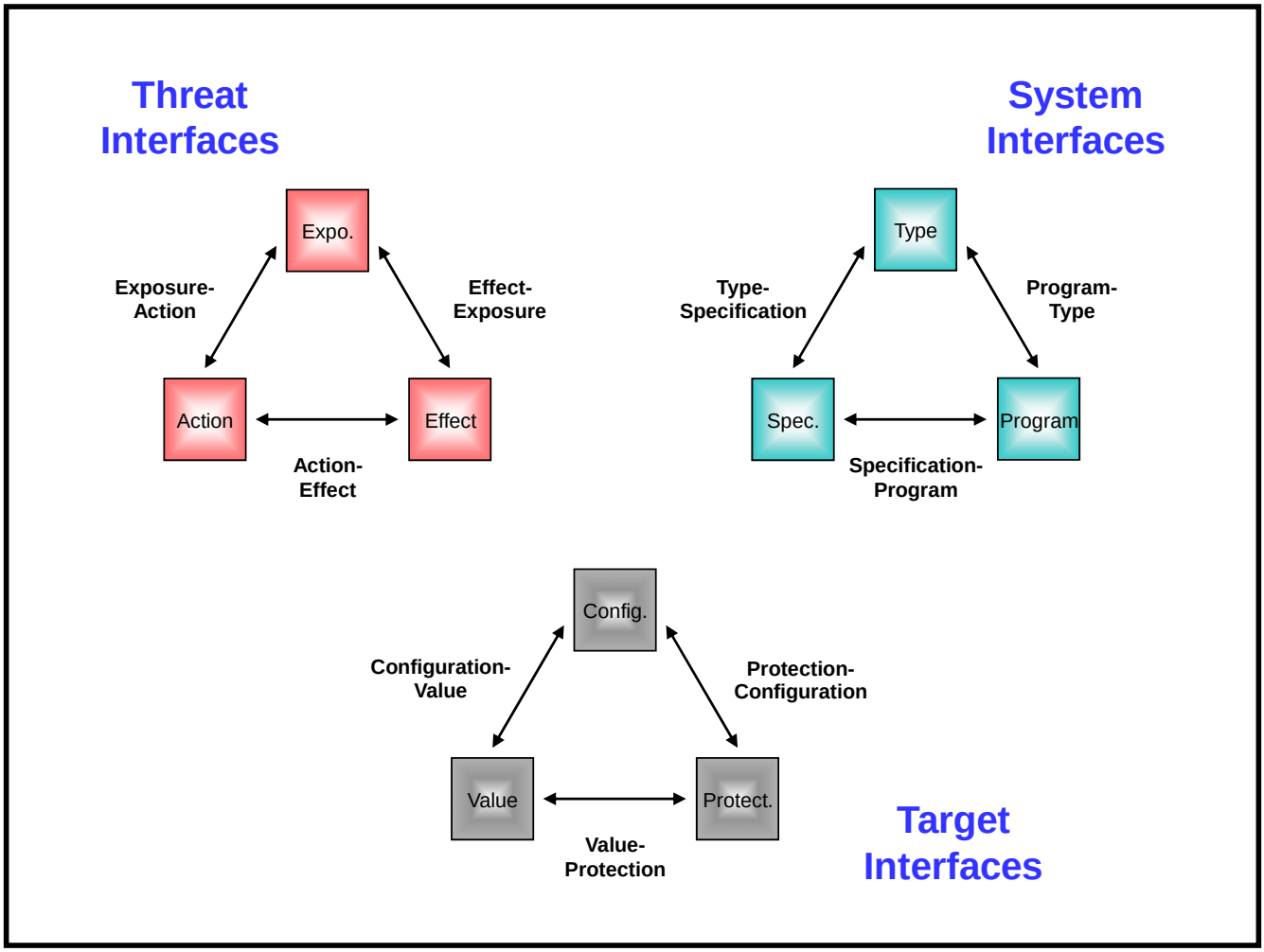

Figure 5. Asset Protection Model (APM) Level Two Interfaces

As described in the initial description of the Asset Protection Model, the system type component is expanded to environmental system, process system and product system. The system specification component is expanded into system functions, system requirements and system architectures. The system program component is developed by addressing system cost, system schedule and system technical components. These three system representations at Level Two are designed to create three structured interfaces. These three structured interfaces are: 1) *Type-Specification* Interface, 2)

Program-Type Interface, and 3) *Specification-Program* Interface. These interfaces are used by systems engineering community to organize the information exchange between the system representations at APM Level Two. For example, when considering the Type-Specification Interface of the system cube, the content and use of this interface will be greatly impacted by the general system type: product system, process system, environment system. When the system or software product system is the point of focus, the Type-Specification Interface is highly developed because the product development is controlled by the product system specification. When the product development process system is the point of focus, the Type-Specification Interface is also highly developed because the product quality and configuration depend directly on the product development process. However, when the environment system is the point of focus, the Type-Specification Interface is less well developed because the environmental system is not under the design control of the product developers. It is possible to describe an acceptable product system environment, but the real environment system is always different with varying levels of system threats.

Also in the earlier description of the APM, the target configuration component is expanded into transmission, storage and processing at Level Two. The target value is elaborated to confidentiality, integrity and availability. The target protection component is expanded into technology, policy and human factors. These three target representations at Level Two are designed to create three structured interfaces. These three structured interfaces are: 1) *Configuration-Value* Interface, 2) *Protection-Configuration* Interface, and 3) *Value-Protection* Interface. These interfaces are used by the information assurance community to organize the information exchange between the target representations at Level Two. For example, when considering the Protection-Configuration Interface the content of the interface will be determined by the information configuration. Each information configuration category will apply different types of technology, human factors, policies and procedures to the task of providing the required level of information system security and risk reduction. Groups of typical solutions, that address the areas defined by the Protection-Configuration Interface, can be developed into a set of security solution patterns that enhance the ability of educators to clearly communicate these topics among themselves as well as to students.

Again summarizing earlier descriptions from the APM, the threat exposure component is expanded to threat actor, threat mechanism and threat vector at Level Two. The threat action is elaborated to pre-event, event and post-event. The threat effect addresses immediate, near-term and long-term effects. These three threat representations at Level Two are designed to create three structured interfaces: 1) *Exposure-Action* Interface, 2) *Effect-Exposure* Interface, and 3) *Action-Effect* Interface. These interfaces are used by the legal, justice and intelligence community to organize the information exchange between the threat representations at Level Two. The Exposure-Action Interface addresses some of the digital forensic issues discussed earlier. Given the goal of applying legal remedies to computer security breaches, the Exposure-Action Interface can be used to organize and communicate the activities associated with preparing a computer system to collect evidence (pre-event actions) collecting evidence during a security breach (event) and preparing the evidence for the legal system after the security breach event (post-event actions).

Each of the Level Two interfaces, similar to the Level One interfaces, provides a bounded contextual plane that is used to develop operational security patterns. The primary purpose of these operational security patterns is the development of a more formalized mechanism for the communication of security risk in a specific context. While the software engineer and/or the system customer are not expected to be able to provide detailed security requirements in most areas, they should be able to identify the general area of application. Once the general area of application is identified, then the operational security patterns can be incorporated into the software development process. Information content, systems and value to an organization continue to evolve and expand at an increasing rate. Conceptual and operational patterns and models that are independent of a specific technology or specific organization are necessary if they are to remain valuable system security management tools over an extended period of time.

PEDAGOGICAL MODEL FOR INFORMATION ASSURANCE CURRICULUM DEVELOPMENT

The pedagogical model for information assurance curriculum development (PMIACD) is considered a high-level meta-system model from which a specific Information Assurance Curriculum system may be developed (Endicott-Popovsky, Popovsky, and Frincke, 2005). Secure information systems and software code development is one component of an information assurance (IA) curriculum. The PMIACD is depicted notionally in Figure 6.

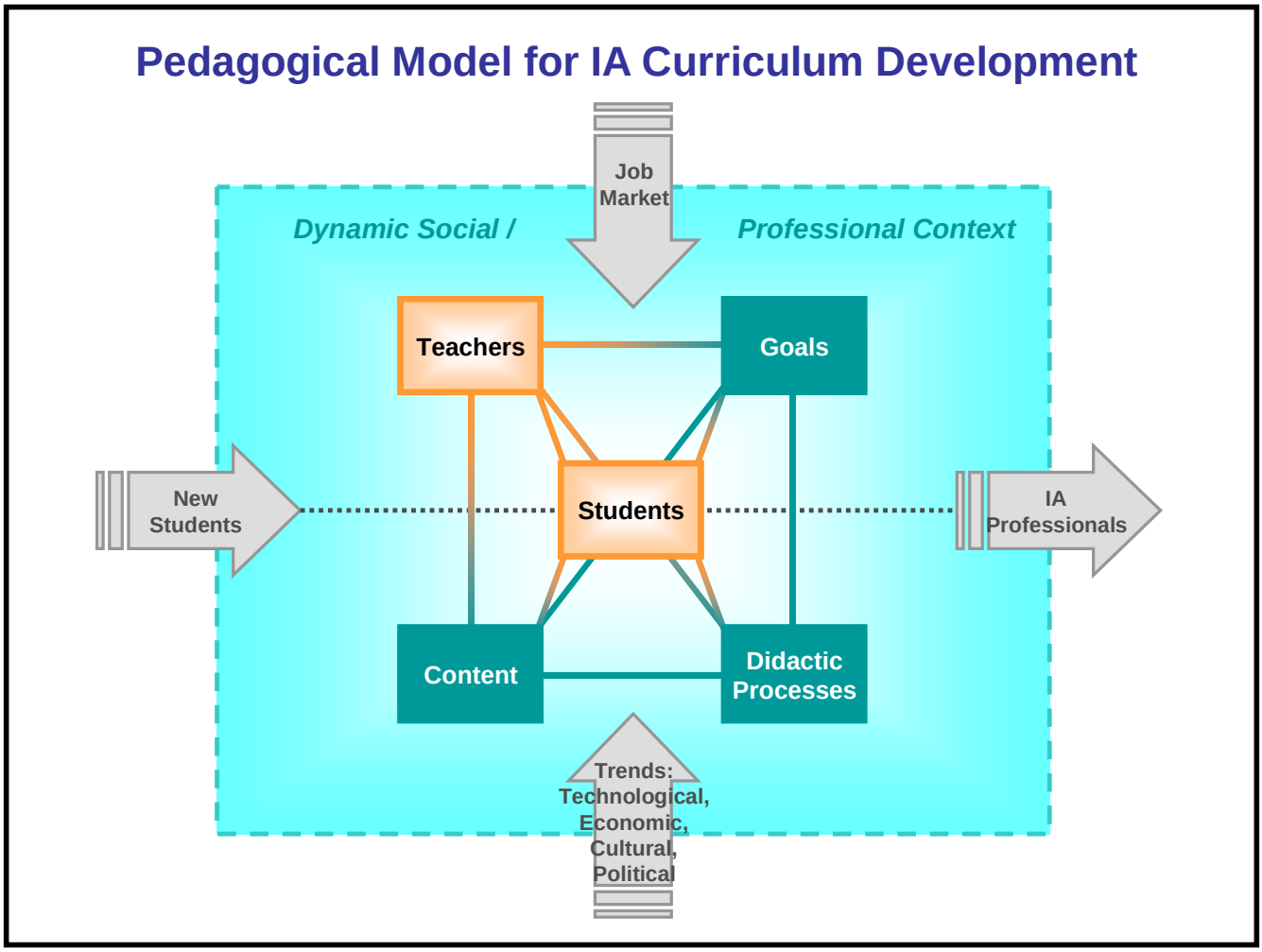

Figure 6. Pedagogical Model for Information Assurance (IA) Curriculum Development. (Adapted from Figure 1, Endicott-Popovsky, et. al., 2005)

The five elements of the PMIACD are composed of two intelligent elements, the instructor and the student, as well as three infrastructure elements that are subject to varying rates of change and adaption. All of the elements of the model function as an interconnected whole. The three infrastructure elements are the goals, content, and didactic processes of the pedagogy. The PMIACD exists in a dynamic professional and social context that includes economic and political environments as well as a constantly evolving set of threats, vulnerabilities and operational systems. Within this pedagogy, the instructor is responsible for developing a specific set of infrastructure components designed to address the needs of a specific type of student in a given context (Popovsky & Popovsky, 2008). For example, given an economic and employment environment wherein a new strategic emphasis is placed on the production and deployment of secure software code, an instructor needs to balance the type of students with economic forces and needs of the employer to produce software engineers that understand the principles and practices of the production of secure software code (Endicott-Popovsky, Popovsky, & Frincke, 2005). In the referenced example, the students were professional software developers that needed to understand the new software coding techniques, and to gain the ability to consistently produce secure software code.

Goals – The overall goal was to induce learning within a mature set of students, resulting in a change in behavior. The learning principles and objectives were designed to enhance individual personal and professional growth based on the values expressed, and the expertise demonstrated, by the student. The learning principles helped prioritize the learning objectives, and were based on targeted measurable outcomes. The learning objectives for the secure coding class were based on the information assurance and trustworthy computing best practices. These learning objectives included: understand and iterate information assurance principles and practices, understand and demonstrate threat modeling techniques, implement secure coding techniques, produce systems that protect information.

Content - Content for a secure software code class is drawn from a large body of knowledge that is based on standards produced by recognized groups and subject matter expert organizations. The specific content for a class must be tailored to the available time and goals of the class. Further, a range of learning levels and acquired skills must be

supported by the class content and sequence of presentation. Given the dynamic operational context of current computing systems, the student must learn to analyze new software security problems, develop and adopt new secure software practices and become independent in the practice of secure software development and deployment.

Didactic Processes - Didactic processes used to deliver the selected content for the secure software production class are tailored based on a number of environmental factors including class type (class room or online), student type (beginning or experienced), and learning outcome goals. Secure software techniques can be based on practices for specific types of computer operating systems, kinds of software operations (database), and/or types of system architecture (two tier or three tier). The didactic process is also adjusted to address the unique needs of specific software techniques.

Using the Pedagogical Model for Information Assurance Curriculum Development (PMIACD), the secure software code development (SSCD) curriculum development model retains the same two generic intelligent elements and three generic infrastructure elements as those found in the PMIACD. Instances of PMIACD and the SSCD curriculum development will have different content when different specific types of instructional activity are planned and developed. The primary output from the SSCD curriculum development process is the secure software education system (SSES) that will be more fully developed later in this paper. A set of specialized system models from the systems engineering domain are used to structure and encode processes and information associated with the development and delivery of these specialized secure software code development instructional materials. Defining the boundary between the security impact and control of (1) secure code practices versus (2) system deployment configuration and operational practices, is an area of current active research (Simpson et al, June 2, 2010).

There are a number of standard processes and approaches used to develop software and software-intensive systems (IEEE-Std-1471-2000, 2000, October; IEEE/EIA 12207.0, 1998, May; Howard & Lipner, 2006). Many of these processes are focused on the production of operational software that has not considered the security requirements associated with the use of the deployed software. Software development processes that do consider security requirements appear to be constrained to addressing security vulnerabilities that are introduced via software defects and/or system architectural defects. The operational effectiveness and suitability of secure coding practices must be evaluated in terms of the deployed system context. In many cases, the current deployed system configuration and content is the aggregated sum of decisions and actions made by a large group of individuals, many of whom may be unknown to the software and system developers. Specialized system models, like the APM, create a framework that supports the evaluation and documentation of the larger operational system context as well as development of the software system requirements. Such a framework can be used to capture, document and communicate the life-cycle security requirements associated with secure system deployment.

The Microsoft Security Development Lifecycle (SDLC) is an example of a well-defined process used to address the reduction of software defects and security vulnerabilities in software code product baselines. The SDLC process has the following steps: training, requirements, design, implementation, verification, release and response. These process steps are focused on software development, and consider threat analysis and attack analysis during the software design phase. SDLC process activities are distributed between the software product teams and the primary security organization. It is clear that the security pedagogy for a software development engineer is similar to, but different from, the security pedagogy for a security professional responsible for the total system security.

STABLE MODEL DEVELOPMENT AND APPLICATION

The ordered hierarchy of the APM and the generic system model provide the foundation for long term conceptual stability associated with secure system and software product production, deployment and utilization. The systems engineering community has developed methods and processes that address the total system including software, hardware and personnel. Standard systems engineering analysis techniques are used to create and evaluate a set of system products that are built upon the conceptual foundation established by the APM and GSM. While secure software coding techniques contribute to the total system security evaluation, other system operational aspects must also be considered. Given any system as depicted in Figure 7, the mission function, system function and system architecture comprise system analytical products that can provide insights to topics that need to be addressed, and to scope and identify system risks (Simpson & Simpson, 2006). As earlier defined within the Generic Systems Model, system functions are those actions or activities that the system is designed to perform. The system architecture is the mechanism that performs the system functions as well as stated by the system requirements. Both of these system analytical products – the system function and the system architecture – are part of the concept, or inward-looking, space of a system. A mission function is part of the context, or outward-looking boundary, for any given system. The mission function identifies what actions need to be performed to be successful. A mission often is supported by multiple system functions.

For example, the University of Washington has a Center for Information Assurance and Cybersecurity (CIAC). The primary mission of the CIAC is to identify, address and promote solutions for information assurance and cyber-security problems and issues. Specific CIAC systems, including curricula, are designed to support the fulfillment of specific aspects of the CIAC mission function. Each of these CIAC systems has system functions associated with them. Clearly, secure software coding techniques are only part of the knowledge and skills that must be developed in the students associated with the CIAC. The Secure Software Education System (SSES) within the CIAC is the system that focuses on secure software development, deployment and utilization, and has its own mission function. However, the SSES must reflect and encompass the areas of focus found in the asset protection model – that is, the context of the threat, system and information that are part of the complete deployed system of interest.

It should be noted that Figure 7 also provides some insights to the relationships between mission functions, system functions, and system architectures and the implications therein. Operational effectiveness is considered a property of the system function that determines how well the mission function is performed. Operational suitability and life cycle cost are both properties of the system architecture. Risk, however, is a function of them all – the mission function, the system function, and the system architecture. Mentioned briefly earlier in the paper and in the following pages, the Secure Adaptive Response Potential (SARP) metric was developed in part to determine where security risks might reside within an overall system, as well as to evaluate operational suitability.

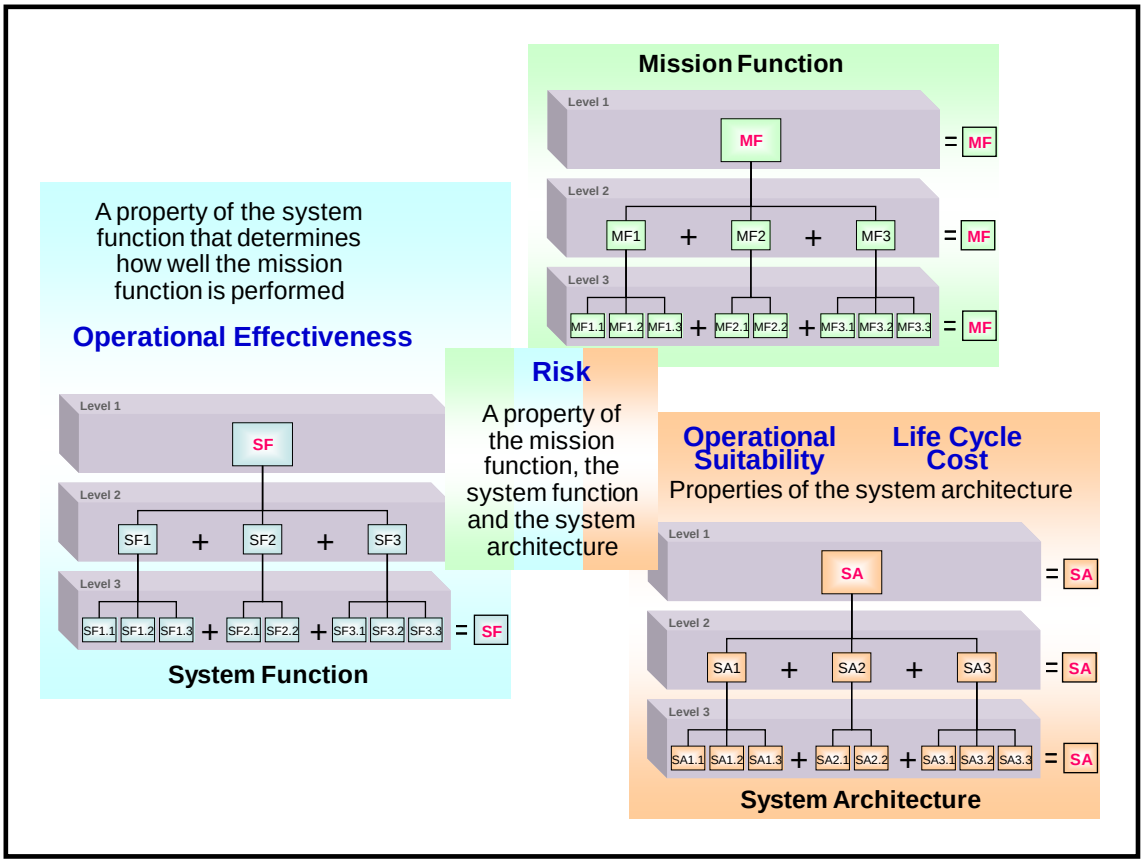

Figure 7. Mission Function, System Function, and System Architecture. (Adapted from Simpson, 2006)

The Secure Software Education System (SSES) is further developed in this paper to demonstrate the structured application of systems engineering techniques. The first step in this structured systems engineering process is to establish the SSES mission function, and its supporting mission sub-functions designed to achieve the mission. The next step is to analyze candidate SSES system functions to determine how each candidate system function supports the SSES mission function. The SSES system functions are based on a generic set of didactic functions associated with a standard educational process. The generic didactic functions are tailored by selecting specific secure software subject content, and complementing the content with applicable methods of demonstration and instruction. Instructional modules are developed to perform each of the selected didactic functions required by the SSES. This process allows for the development and evaluation of a set of alternative instructional and didactic module types. Given a specific educational area, with clearly documented goals and objectives, the instructor can select the modules that best meet the didactic goals and objectives of that specific area, or develop one that is more appropriate.

Considered in this paper are 14 categories of the SSES secure coding system functions that are based on the Common Weakness Enumeration research concept categories (CWE, 2010). These categories are listed and described in Table 2.

Table 2. Categories of Secure Coding Functions

Secure Code Functional Category	***Description***
Category 1: Functional Effectiveness	Functional effectiveness addresses the programming activities needed to produce software code that provides and/or supports the needed organizational mission function. This category covers what the software must do, without regard to the operational suitability of the software. Code developed using practices in this category is evaluated using techniques contained in the other categories.
Category 2: Software Engineering Standards	The software engineering standards category addresses the range of existing software engineering standards whose focus is applicable to the current class offering and set of students. The selected standard will be used to assess the correctness of the software architecture and code.
Category 3: Coding Standards Violations	The coding standards violations category addresses any violations in the selected coding standards and/or well known coding design principles. This category of secure coding practice addresses issues of poor code quality, violation of secure design principles, embedded malicious code, failure to fulfill an API contract, as well as the use of undefined, unspecified or implementation-defined behavior. Software architecture design, code development and system implementation are the most common areas where these software vulnerabilities are introduced. The specific type of system (operating system, network system, three-tier system) chosen for the secure software educational activity will determine the level and applicability of the coding techniques contained in this category.
Category 4: Exceptional Conditions Handling	The exceptional conditions handling category addresses the handling of exceptional conditions. Some of the specific details of this category are tied directly to the software architecture, and the software language used to implement the secure software system. The improper handling of syntactically invalid structures is an example of an area in this category that is architecture and language dependent. The improper or missing verification of unusual or exceptional conditions, as well as the ineffective handling of exceptions are also included in this category of secure software practices. The primary focus of this area is associated with the careful evaluation and analysis of the programmers assumptions associated with the system configuration and deployment state.
Category 5: Resources Accessed by Index	The 'resources accessed by index' category addresses the techniques used to manage indexed files, memory locations and other indexical resources. This category includes the classic buffer overflow associated with copying an input buffer to an output buffer without verifying that the size of the input buffer is less than the output buffer. These are language dependent issues associated with structured languages that do not have memory management support. Memory allocation, out-of-bounds read, out-of-bounds write, and pointer value outside of proper range, are all included in this category of secure software techniques.
Category 6: Resource Lifetime Control	The resource lifetime control category addresses the maintenance of proper resource control over the (creation, use, release) lifetime of the resource. A rich and deep set of software security issues are grouped in this category. This group includes covert channels, uncontrolled resource consumption, improper user authentication, incorrect type conversion, and invalid initialization. Secure software education must highlight the solutions to software security issues contained in this category.
Category 7: Message or Data Structure Processing	The data structure processing category includes the coding activities that are necessary to properly enforce the structure and content of messages and data structures as they move through the system processing thread of control. Data structure component deletion, improper null termination, and encoding errors are all a part of the secure coding activities included in this category. These issues are primarily language and architecture independent, and are associated with ineffective input and processing control.

Secure Code Functional Category	*Description*
Category 8: Improper Calculation	The improper calculation category is organized around software operations that generate unintended or incorrect calculation results. Due to the ubiquitous use of calculations in every aspect of secure coding activities, the effects of issues in this category may impact resource management, security-critical decisions, privilege assignments, arbitrary code execution, and other security sensitive operations. Software and system design, architecture, and implementation decisions are the main areas that introduce these types of software vulnerabilities that are associated with all types of software languages.
Category 9: Improper Comparison of Security-Relevant Entities	The improper comparison of security-relevant entities category is focused in the area of object comparison. Comparison of object references, instead of object contents, flawed regular expressions, missing default case in a switch statement, partial state distinction and partial comparison are all included in this category of secure coding focus areas. This secure coding area is a basic area of concern, and many other code vulnerabilities may be built on the secure coding flaws in this category.
Category 10: Thread-of-Control Flow Management	The thread-of-control flow management category is focused in the area of secure software architecture design and implementation. This category includes improper flow control scoping, incorrect iteration control, flawed thread synchronization, multiple types of race conditions, as well as uncontrolled recursion and uncaught exceptions. Secure coding issues associated with this category are fundamental in software and code control, generating significant aberrant behavior if they are introduced into a set of production code.
Category 11: Error in Interaction	The error in interaction category is focused around the flawed interaction between two software systems that each function correctly when operating independently. This category includes flawed application of memory addressing, compiler modification of security-critical code, as well as improper functional behavior associated with a new version and/or deployment environment. All languages and secure software development process are areas in which these types of secure coding issues may be introduced.
Category 12: Flawed Protection Mechanisms	The flawed protection mechanisms category is focused on instances of incorrect or missing protection mechanisms in the secure software code. This category focuses on cases where there are known secure software protection mechanisms available, but they are not applied, or are applied incorrectly. Cases of flawed authentication, verification, access control, encryption deployment, and use of untrusted inputs are all included in this category. These issues are applicable to all software language types and secure software development processes.
Category 13: Improper Application of Pseudorandom Values	The improper application of pseudorandom values category is focused on the use of predictable and insufficiently random values. Improper random seed, hard coded cryptographic key, hard coded credentials, hard coded password and use of cryptographically weak software components are all included in this secure code category. All system development stages and software languages are impacted by issues in this category.
Category 14: Catch-All	The catch-all category addresses existing and/or emerging security issues that do not fit in the other categories.

A basic course in secure software engineering would introduce the students to each of these categories, but would focus on the most basic and fundamental software security coding techniques. These beginning students would also be introduced to the automated tools, techniques, software languages and architectures that are used to produce a secure software deployed code base. Intermediate and advanced courses in secure software engineering would cover all of the categories, and build on the secure coding techniques introduced in the basic class.

The Secure Software Education System (SSES) addresses secure software engineering education using three levels of content difficulty: basic, intermediate and advanced. The SSES basic level educational content includes an introduction to organized responses to software security issues. These sources of publicly available data organize and categorize software security issues, and include the National Vulnerability Database and Common Weakness Enumeration among others. The basic level also includes a strong focus on fundamental software security issues and solutions that address buffer overruns, integer overflows, SQL injection, XML injection, command injection, improper use of cryptography, and failure to protect network traffic. Computer and network architecture security flaws and other issues are introduced at the basic level, but are

further developed in the intermediate level didactic material. Computer programmers and beginning software system architects are the primary types of students for the beginning level material.

At the intermediate level of the SSES secure software engineering education, the basic secure coding material is explored in more depth, and provides an additional emphasis on computer, application and network security practices. Also included in the intermediate level content are software security vulnerabilities that are rooted in the application and network architecture, such as cross-site scripting, cross-site request forgery, and HTTP response splitting. Security issues associated with trust boundaries and threat boundaries are emphasized at the intermediate level. Software security development life cycles and specialty security testing methods, like fuzz testing, are included at the intermediate level along with discussions of distributed cryptography system implementation and use. Differences between and among the three most deployed computer operating systems – Microsoft Windows, Mac OS X and Linux – are also discussed in terms of software and architectural deployment patterns, along with an introduction to mobile computing security issues. Lead computer programmers, senior system and software architects and chief information security officers (CISO) are the primary types of students for the intermediate level material.

At the advanced level of the SSES, the basic and advanced secure coding material is combined into problem sets and configurations that are commonly found in ongoing commercial, governmental and military practices of secure system and software design, production, deployment and operations. One specific application area may be emphasized more than another depending on the students interest and employment conditions. The primary focus of the advanced level of the SSES material is organizational control of policy, risk, training, and processes that impact the security of operational software intensive systems. Senior system engineers, software architects and CISO's are primary types of students for the advanced level material.

Specific instances of software security applications and techniques are mapped to a specific operational context, and are evaluated using a secure adaptive response potential (SARP) system security metric, which was introduced earlier in the paper. The SARP security metric was designed to address system security operational suitability throughout the system lifecycle (Simpson, Miller, & Dagli, June, 2008). Specific classes of system operational suitability may be developed and categorized using the SARP metric components that generally describe the operation and maintenance profile of the deployed system. This type of material would be introduced at the SSES intermediate level, and fully developed in the advanced level SSES courses. A typical set of system security metrics is shown in Figure 8.

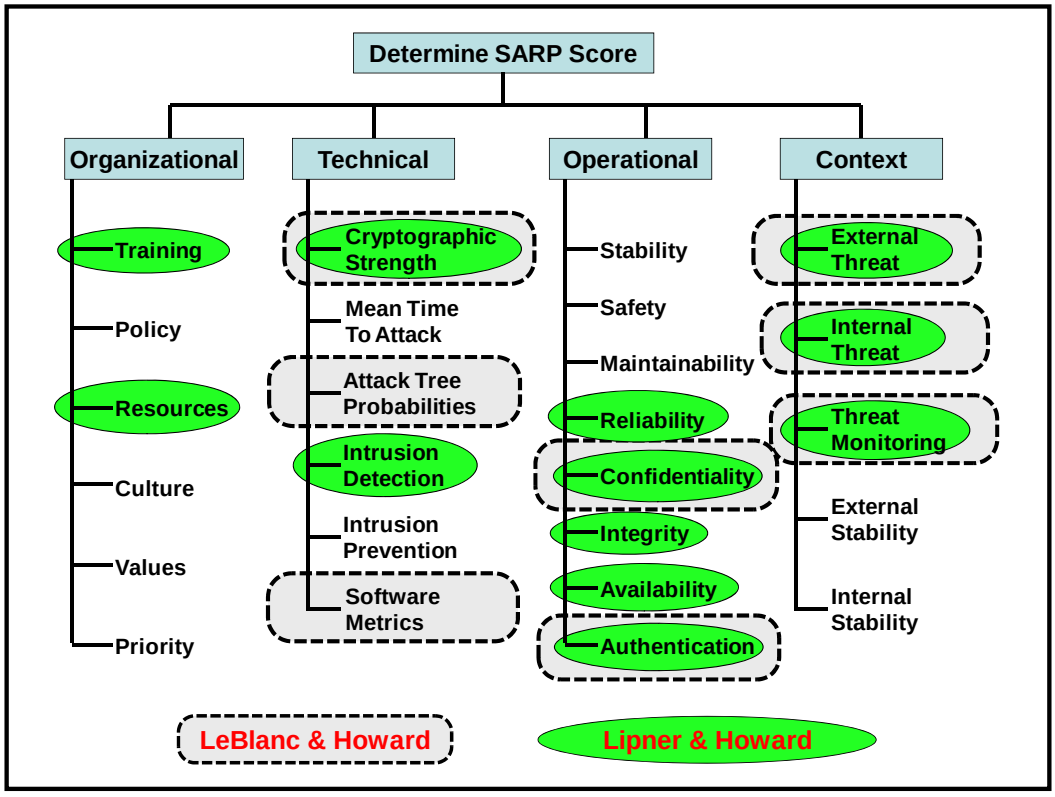

Figure 8. Secure Adaptive Response Potential (SARP), A System Security Metric (Adapted from Simpson, 2008)

Once these SARP categories are established, then specific system security architecture cost and risk decisions can be made based on these categories using standard systems engineering analysis techniques. The practice of secure software development can now be organized, discussed and communicated in terms of the Asset Protection Model, the Generic System Model, the Secure Software Education System, and the SARP categories. As shown in Figure 8, the topics in ovals associated with the Microsoft security development lifecycle (Howard & Lipner, 2006) cover only a portion of the SARP areas of interest. The topics associated with writing secure code (Howard & LeBlanc, 2003), shown by the rounded rectangles with dashed lines, cover fewer topics than those covered by the Microsoft security development lifecycle.

The secure software educator is challenged to properly scope, refine and establish context for effective instruction in the practice of secure coding. As security activities are focused on the production of secure code, there is a tendency to focus on only part of the enterprise information assurance problem and context. All of the items listed in Figure 8 are considered in general texts that are focused on information security for the enterprise (Schou & Shoemaker, 2007). As a result, an approach to secure software education that establishes basic, intermediate and advanced sets of knowledge, skills and abilities, as well as restricted system architectures, is used to organize and present the SSES curriculum.

SYSTEM AND SOFTWARE UTILIZATION CONTEXT

Software is utilized in a number of configurations and deployed system contexts. The basic system types considered in this paper are: (1) a single unconnected computing system, (2) an isolated network comprised of host computing systems, and (3) an interconnected group of networks comprised of host computers and associated networks. The terms, concepts and semantics that are contained in the security content automation protocol (SCAP) are also incorporated into the software utilization context (NIST SP 800-126, 2009). The four primary aspects of information technology security that are addressed by the SCAP are: (1) asset management, (2) configuration management, (3) compliance management and (4) vulnerability management.

- The asset management activity is supported by the common platform enumeration which standardizes an information technology system naming structure (DHS-NVD, CPE, 2009). The common platform enumeration is focused on individual platform identification and description.
- The configuration management activity is supported by the common configuration enumeration (DHS-NVD, CCE, 2009). The common configuration enumeration is organized around specific operating system or application recommended product configuration settings.
- The compliance management activity is supported by the extensible checklist configuration description format (NIST, 2008). The extensible checklist configuration description format is organized around a fundamental data model and targeted areas of application. The original area of application was automated security checklists. However, extensible checklist configuration description format uses have grown beyond these original application goals to support the generation of documentation of manual procedures, test results and many other text-based activities, including the production of training materials. The extensible checklist configuration description format has a fundamental data model that is composed of four main object types: 1) benchmark, 2) item, 3) profile and 4) test-result.
- Vulnerability management activities are supported by the common vulnerability enumeration, open vulnerability and assessment language, and the common vulnerability scoring system. The common vulnerability enumeration is an extensive list of publicly known information security vulnerabilities and exposures. The open vulnerability and assessment language standardizes the report and assessment of computer system security states and modes to support the reliable and reproducible automation of security system evaluation. The common vulnerability scoring system is a standardized vulnerability scoring system used for rating, evaluating and communicating system security vulnerabilities.

The secure software production techniques addressed in this document are directly associated with the known system vulnerability type that is being addressed. This direct connection has the benefit of placing the software engineer's activities in the context of a known threat environment as well as introducing the vast amount of information available on all types of software vulnerabilities.

SECURE SOFTWARE EDUCATION SYSTEM (SSES) PROCESS

As part of the Center for Information Assurance and Cybersecurity, the SSES contains the elements and processes necessary to guide a student on the journey from incomplete and unstable secure software development practices and techniques to highly-stable and comprehensive secure software development practices and techniques. All educational content and

components associated with the SSES are delivered using the general process outlined in Figure 9. Educational content and components for each specific subject area are inserted into the general process by the instructor which then creates a specific educational experience for the students learning about that secure software subject.

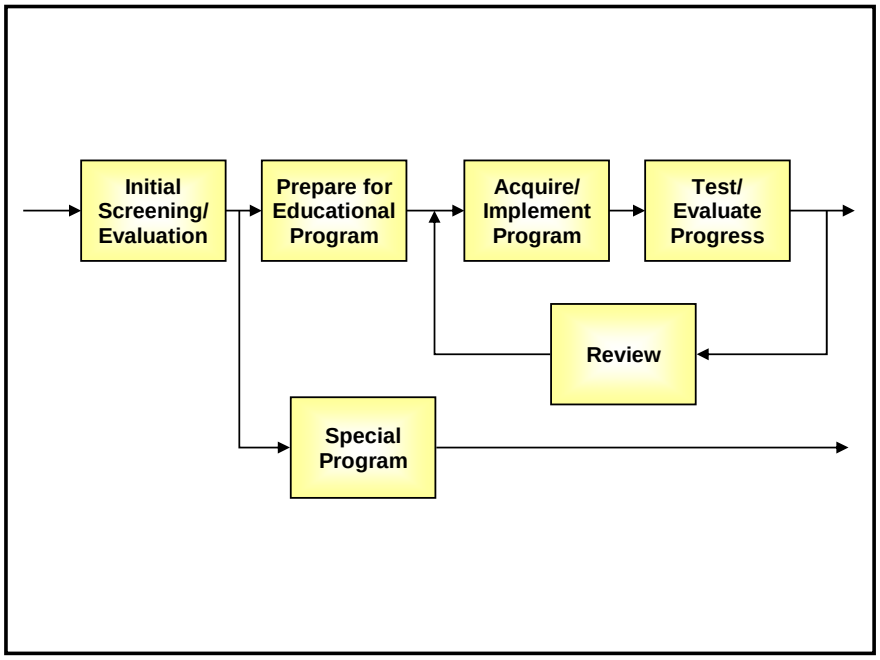

Figure 9. General Educational Process Steps

SECURE SOFTWARE PRODUCTION EDUCATIONAL FUNCTIONS

The secure software education system functions are designed to support the achievement of the Pedagogical Model for Information Assurance Curriculum Development mission functions by providing the required secure software production educational functions. The functions have been grouped into the following categories: basic software knowledge, advanced software knowledge, basic secure software knowledge, and advanced secure software knowledge. The basic software knowledge category contains secure code software knowledge, practices and techniques that should be known by everyone that produces software for others to use. The basic software knowledge category focuses on host-based systems with a very shallow introduction to network bases and distributed systems. The advanced software knowledge category covers the area of distributed, networked system code along with associated authentication and access controls.

For effective integration of this approach using these processes, a set of interfaces between the models is essential. Initially, aspects of the Pedagogical Model for Information Assurance Curriculum Development are mapped to the mission function, the system function, and the system architecture. The PMIACD goals – what the system is designed to accomplish – are mapped to the mission function. The PMIACD content are mapped to the system functions, and the PMIACD didactic processes are mapped to the system architecture**.** As detailed earlier in the paper, the information Asset Protection Model (APM) is a structured model of the information protection domain designed with a clear set of abstraction levels. These abstraction levels are used to identify and organize relevant security concepts in a manner that is accepted by experts in the system and software security field. Once a specific asset protection instance is identified that uses a system architecture, or a software-intensive system architecture, then the specific secure coding techniques can be mapped to the required system specification and threat exposure context. These mappings are shown notionally in Figure 10.

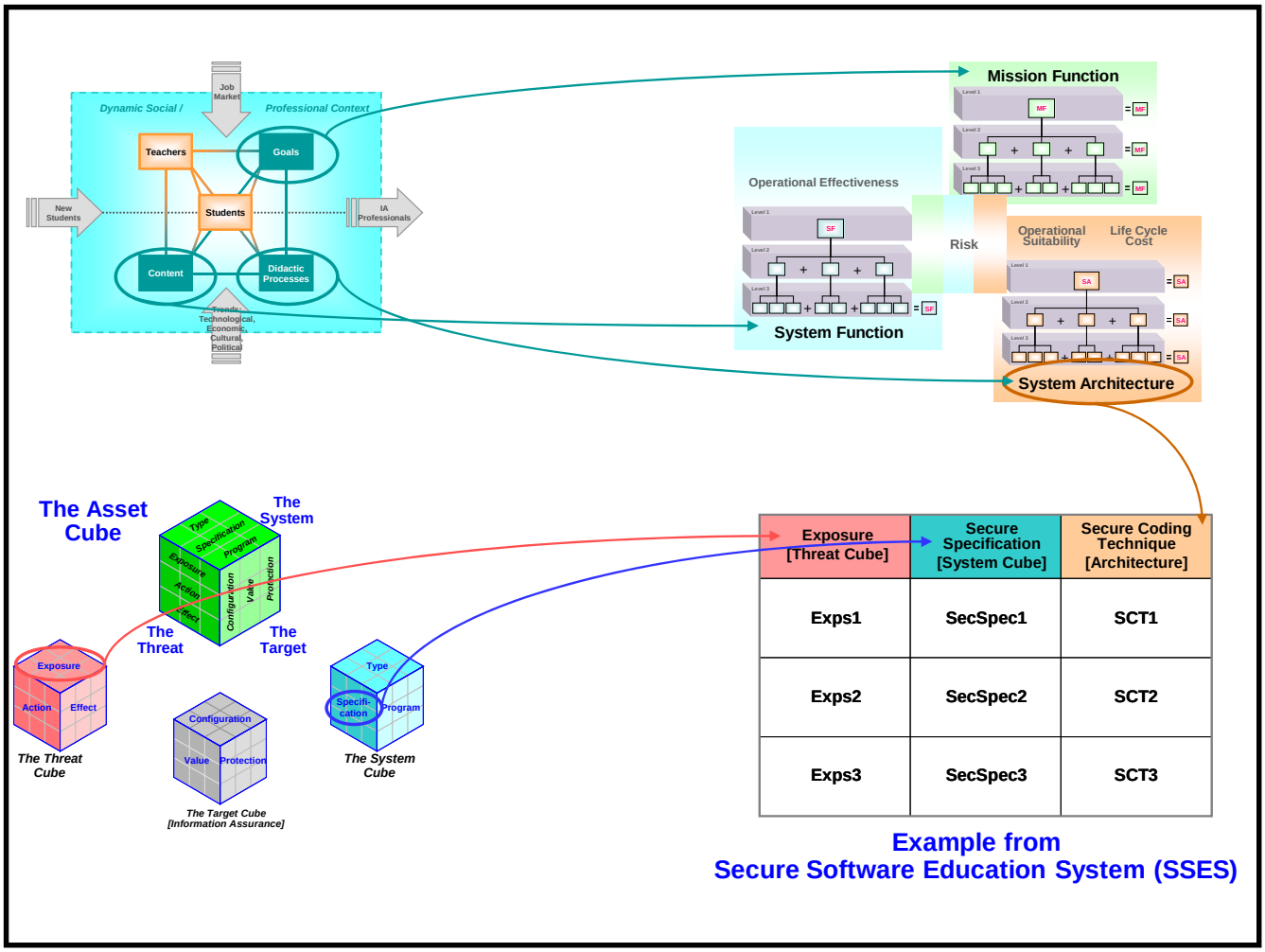

Figure 10. Secure Software Production

SPECIFIC SECURE SOFTWARE DEVELOPMENT TECHNIQUES

After the APM has been used to establish the relevant abstraction level, scope and application characteristics associated with a specific secure coding application, standard secure system and software production techniques, and other standard system security practices can be used to generate a system product that provides the required level of risk reduction and information assurance. Administrative, policy, operational, and technical system controls are all evaluated, selected and implemented in a manner that provides the proper level of cost-effective system security in the identified product system context. The following sections outline a proposed set of standard, secure system and software areas that should be included when secure software coding techniques are being taught to students.

Host Computer Operating System, Operating System (OS) Command Injection. A host computer supports many types of applications and operating system commands. This section of the SSES addresses the use of operating system calls in specific types of applications and application languages. A typical example of OS command injection is recorded in CERT Advisory CA-1996-06 Vulnerability in NCSA/Apache CGI example code. Common Weakness Enumeration (CWE) 78: Improper Sanitization of Special Elements used in an OS command, details this software security defect's construction, typical consequences, detection methods as well as providing code examples and methods needed to mitigate and/or eliminate this type of software security defect. There are a number of areas in the secure software development lifecycle where OS command injection may be effectively addressed. These areas include system architecture and design development, system implementation and code generation as well as testing and operation (DHS-NVD CWE, 2009).

Host Computer Operating System, Buffer Overflow. This portion of the SSES covers the security techniques necessary to address buffer overrun. Buffer overflow weaknesses are associated with computer operating systems and applications that are written in languages that do not have automated memory management support, and allow the

programmer to directly address, allocate and de-allocate memory registers and computer memory stacks. The consequences of performing an operation outside the bounds of a physical memory buffer vary dramatically depending on the computing language, computing platform and hardware architecture. There is a high potential for vulnerable systems to have their integrity, confidentiality and availability compromised when the buffer overflow weakness is exploited by an attacker (DHS-NVD CWE, 2009).

Three Tier Database System - Secure Software Access Control. This section of the SSES addresses proper access control and authentication. The access control matrix is introduced and explained using examples. Authentication basics are introduced first, followed by discussion of specific authentication methods and techniques. Data-driven application and web-enabled systems that include a database component that accepts externally-influenced data from an input component, are subject to this security weakness because SQL makes no operational distinction between control input and data input. The net results of these types of security flaws are similar to the net result of OS command injection and buffer overflows. Successful SQL injection attacks are analyzed, and effective secure software development techniques that prevent these types of attacks are identified. Both the attacks and methods of attack are communicated to the students.

Networked Data Systems - Secure Software Network Security. This section of the SSES addresses secure networking and socket techniques. This instructional unit starts with an overview of policy development, network organization and network security practices. Web applications and data-driven systems that use XML data structuring and accept externally-influenced input data are subject to this weakness, because XML makes no operational distinction between control input and data input. Successful SQL injection attacks can impact system availability, integrity and confidentiality.

General Secure Coding Techniques and Patterns. This section of the SSES integrates fundamental aspects and procedures contained in all portions of the SSES. These aspects and procedures are packaged as analysis, design and implementation patterns. The patterns from this section are the main educational and technical take-away from the SSES educational activities. These application patterns use the APM to properly scope the type of security problem addressed, as well as to organize the general set of secure coding techniques.

The Secure Software Educational System final project is used to provide the students an opportunity to practice and demonstrate the secure software methods and techniques that they learned in the SSES course of instruction. Secure software modules built during the previous class periods are used as the basis for solving a small- to medium-sized secure coding problem that is based on a three-tier data base architecture or another acceptable system and software architecture.

SUMMARY AND CONCLUSIONS

The dynamic nature of deployed software-intensive systems environmental threats and operational security requirements is addressed using a series of models that cover the art and science of cyber security instruction, asset protection, general systems, and secure software requirements development. A targeted range of models is required to support the discussion and analysis of educational techniques and processes that are used in the area of secure software intensive systems development, deployment, and operation. Even entry-level students must gain an understanding of how to acquire new knowledge, and how to adapt their standard software security approaches as a direct result of the fast flux of new technology development and services built on these new technologies.

The Pedagogical Model for Information Assurance Curriculum Development, as deployed at the University of Washington's Center for Information Assurance and Cyber Security, is a dynamic model that provides a focus for the center's operation. The Asset Protection Model is a structured model that focuses on three main aspects of secure systems: the target, the threat and the system. Proposed application processes focus specific professional groups in each of these different areas, and provide a structure through which they can collaborate and define a common view of the security issues. The Generic System Model, combined with other instructional models, is used to analyze and structure material for a specific type of system and a specific type of student.

Additional research is needed to identify, develop and verify instructional and didactic techniques that enable students to effectively address the production, deployment, configuration, and operation of secure software intensive systems in a dynamic threat environment. Further research in the application of information assurance to other domains, using the Asset Protection Model, is expected to support a wide variety of organizational and system design activities, including the development of clear, functionally-based information assurance job descriptions, and organizational roles and responsibilities.

REFERENCES

Bishop, M. (2005). *Introduction to Computer Security.* Boston, Massachusetts: Addison-Wesley.

Common Weakness Enumeration (CWE) Categories (2010). Sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security and hosted by The MITRE Corporation. Retrieved August 24, 2010, from http://cwe.mitre.org/data/definitions/78.html

Department of Homeland Security, National Cyber Security Division/US-CERT and National Institute of Standards and Technology, National Vulnerability Database, Common Configuration Enumeration (CCE) Reference Data. Retrieved August 24, 2010, from http://nvd.nist.gov/cce.cfm

Department of Homeland Security, National Cyber Security Division/US-CERT and National Institute of Standards and Technology, National Vulnerability Database, Official Common Platform Enumeration (CPE) Dictionary. Retrieved August 24, 2010, from http://nvd.nist.gov/cpe.cfm

Department of Homeland Security, National Cyber Security Division/US-CERT and National Institute of Standards and Technology, National Vulnerability Database, Common Weakness Enumeration (CWE). Retrieved August 24, 2010, from http://nvd.nist.gov/cwe.cfm

Endicott-Popovsky, B., Popovsky, V., & Frincke, D. (2004, June). *Designing a Computer Forensics Course for an Information Assurance Track.* Paper presented at the 8th Colloquium for Information Systems Security Education, West Point, NY.

Endicott-Popovsky, B., Popovsky, V., & Frincke, D. (2005, June). *Secure Code: The Capstone Class in an Information Assurance Track.* Paper presented at the 2005 Colloquium on Information Systems Security Education (CISSE): Pursuing Quality Solutions—Lessons Learned and Applied.

Gross, A. (June 11, 2010). Retrieved August 24, 2010, from http://languag2.home.sprynet.com/f/evishop.htm

Howard, M. & LeBlanc, D. (2003). *Writing Secure Code.* Redmond, Washington: Microsoft Press.

Howard, M. and Lipner, S. (2006). *The Security Development Life Cycle.* Redmond, Washington: Microsoft Press.

IEEE-Std-1471-2000 (2000, October). *Recommended Practice for Architectural Description of Software-Intensive Systems*. Retrieved August 24, 2010, from http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html .

IEEE/EIA 12207.0 (1998, May), *Standard for Information Technology – Software Life Cycle Processes.*

Mar, B.W. & Morais, B.G. (2002, August). *FRAT – A Basic Framework for Systems Engineering*. Paper presented at Twelfth Annual International Symposium of INCOSE, "Engineering 21st Century Systems: Problem Solving Through Structured Thinking," Las Vegas, Nevada.

McCumber, J. (2005). *Assessing and Managing Security Risk in IT Systems: A Structured Methodology*. Boca Raton, Florida: Auerbach Publications,.

Mead, N. R. (2010, January-March). Benefits and Challenges in the Use of Case Studies for Security Requirements Engineering Methods. *International Journal of Secure Software Engineering*, *1*(1),74-91.

National Institute of Standards and Technology (2008, January). Publication, The eXtensible Configuration Checklist Description Format, Release 1.1.4. Retrieved August 24, 2010, from http://scap.nist.gov/specifications/xccdf/

National Institute of Standards and Technology (2009, November). Publication # NIST SP 800-126, 2009, The Technical Specification for the Security Content Automation Protocol.

Popovsky, V., & Popovsky, B., (2008, June). *Integrating Academics, the Community and Industry.*
ISBN 978-5-903247-15-8

Schou, C. & Shoemaker, D. (2007). *Information Assurance for the Enterprise: A Roadmap to Information Security.* New York, New York: McGraw-Hill/Irwin.

Simpson, J.J. (2002, August). *Innovation and Technology Management*. Paper presented at the Twelfth Annual International Symposium of INCOSE, "Engineering 21st Century Systems: Problem Solving Through Structured Thinking," Las Vegas, Nevada.

Simpson, J.J. & Simpson, M.J. (2003, July). *Systems and Objects*. Paper presented at the Thirteenth Annual International Symposium of INCOSE, "Engineering Tomorrow's World Today", Crystal City, Virginia.

Simpson, J.J. (2004, April). *System Frameworks*. Paper presented at the Second Annual Conference on Systems Engineering Research, Los Angeles, California.

Simpson, J.J. & Simpson, M.J. (2006, July). *Foundational Systems Engineering (SE) Patterns for a SE Pattern Language.* Paper presented at the Sixteenth Annual International Symposium of INCOSE, "Systems Engineering: Shining List on the Tough Issues ", Orlando, Florida.

Simpson, J. J., Miller, A. & Dagli, C., (2008, June). *Secure Adaptive Response Potential (SARP): A System Security Metric*. Paper presented at the Eighteenth Annual International Symposium of INCOSE, "Systems Engineering for the Planet", Utrecht, The Nederlands.

Simpson, J. J., Votipka, S., Wang, T., Baklanoff, T., Sweers, N., (2010, June 2). *Final Project Report, Threat Incident Modeling Team*. Paper presented to IMT 553 – Establishing and Managing Information Assurance Strategies, University of Washington, Seattle, WA.

Simpson, J.J. & Simpson, M.J. (2010, June 3). Complexity Reduction: A Pragmatic Approach. *Systems Engineering Journal*, Article first published online: 3 June 2010 doi:10.1002/sys.20170.

Simpson, J.J. & Endicott-Popovsky, B., (2010, June 7). *A Systematic Approach to Information Systems Security Education*. Paper presented at the 14th Colloquium for Information Systems Security Education, Baltimore, Maryland.

Warfield, J.N. (1990). *A Science of Generic Design*. Ames, Iowa: Iowa State University Press.