A Survey on Secure Software Development Lifecycles

José Fonseca[1], Marco Vieira[2]
[1]DEI/CISUC, University of Coimbra / Polytechnic Institute of Guarda, Portugal
[2]DEI/CISUC, University of Coimbra, Portugal

ABSTRACT
This chapter presents a survey on the most relevant software development practices that are used nowadays to build software products for the web, with security built in. It starts by presenting three of the most relevant Secure Software Development Lifecycles, which are complete solutions that can be adopted by development companies: the CLASP, the Microsoft Secure Development Lifecycle and the Software Security Touchpoints. However it is not always feasible to change ongoing projects or replace the methodology in place. So, this chapter also discusses other relevant initiatives that can be integrated into existing development practices, which can be used to build and maintain safer software products: the OpenSAMM, the BSIMM, the SAFECode and the Securosis. The main features of these security development proposals are also compared according to their highlights and the goals of the target software product.

INTRODUCTION
The Software Development Lifecycle (SDL) is a conceptual model used by software houses in the management of the process of analyzing, developing, controlling and maintaining software (Sommerville, 2010). Some of the most well-known models are the Waterfall (Royce, 1970), the Rapid Application Development (Martin, 1991) and the Spiral (Boehm, 1986). At the time when these SDLSs were developed, the software security awareness was not as relevant as it is today, so it was not a big concern to take into account. In fact, the typical approach of dealing only with development best practices is not sufficient for current applications that have to face the constant pressure of web attacks, although they can improve the overall quality and help mitigate some common issues.

These traditional SDLs are still in widespread use nowadays, but they are not effective when building secure systems that have to face the huge number of threats that can arise from anywhere, like those that come from the web and are so pervasive (Howard & LeBlanc, 2003). Both logic and coding bugs must be thoroughly addressed during all the phases of the development process, therefore reducing the cost of deploying unsecure application. This is of utmost importance for web applications that will be exposed to the growing number of hackers and organized crime that can strike at any time, from any place in the Globe. This is what an integrated Secure Software Development Lifecycles (SSDL) does from the start to the end of the life of an application. In fact, using a SSDL is one of the recommendations of the Verizon's 2009 data breach report in order to prevent the application layer type of attacks, including SQL Injection and XSS (Baker et al., 2009).

This chapter presents an overview of the most important SSDLs that are used nowadays to build software products that have to face the many threats that come from the web: the Open Web Application Security Project (OWASP) Comprehensive, Lightweight Application Security Process (CLASP), the Microsoft

Secure Development Lifecycle, and the Software Security Touchpoints. Although there is a general consensus about the advantages of using a SSDL, this subject is still in its early adoption by the industry. It takes time to implement and execute, it costs money and it implies a change in the way organization works, which is usually difficult to achieve. The way a secure software should be developed is still generating a growing number of discussions and there is a considerable number of proposals trying to gain adopters and overcome the problems and technical difficulties of applying them in the real world (Higgins, 2009). This chapter also introduces other relevant initiatives, which can be adapted to the existing SDL, devoted to building and maintaining a safer software product: the Open Software Assurance Maturity Model (OpenSAMM), the Building Security In Maturity Model (BSIMM), the Software Assurance Forum for Excellence in Code (SAFECode) and the Securosis building a web application security program.

This chapter also discusses the issue of selecting a software development lifecycle according to the reality of the software product being developed. This involves identifying the security issues that should be addressed from a development point-of-view and then map these issues with the features of existing lifecycles to make the right choice and tune any relevant aspects.

SOFTWARE DEVELOPMENT AND SECURITY

One important metric of software quality is assurance: “a level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its lifecycle, and that the software functions in the intended manner” (CNSS Secretariat, 2006). To achieve software assurance developers need to build assured software: “Software that has been designed, developed, analyzed and tested using processes, tools, and techniques that establish a level of confidence in its trustworthiness appropriate for its intended use” (CNSS Secretariat, 2006). To achieve this goal, developers must rethink the software development process and address all the phases of the SDL: design, code and documentation (Howard & LeBlanc, 2003). This is like applying the defense-in-depth strategy to the various phases of the software development lifecycle making it more security aware.

To understand the security measures that vendors use for software assurance, Jeremy Epstein analyzed eight software vendors with small to very large revenues (Epstein, 2009). The security measures analyzed were software developer training, software design review, execution of penetration testing using humans and tools during the SDL, and source code analysis. The study showed that almost every company is conscious of the risks of insecure software and performs all these activities, to some extent. However, although their clients do not ask explicitly for security, software vendors implement security assurance mechanisms because they are aware that in case something goes wrong it will bring them negative consequences. Other companies should follow this practice, given that the web application scenario is so prone to vulnerabilities and it is so common for an application to be probed by possible hackers. In fact, “If you do not perform security testing for your application, someone else not working for your company will” (Howard & LeBlanc, 2003). Customers do not ask for security but, if security fails, they will move to another solutions provider.

Software developers frequently see functionality as more important than quality or security. This is natural since the functionality is what represents the need for a given product. Without it there is even no need for security because there will be nothing to be secured. However, security should be seriously considered. A Gartner report says that 75% of attacks take place through the application level and predicts that, by 2009, around 80% of companies will have suffered a security incident due to application vulnerabilities (Lanowitz, 2005). The report also adverts for the need to build secure applications and test applications for security from the early start of the project during the application development lifecycle.

One major problem is that typical SDLs in use nowadays are not effective when building secure systems. Security is not integrated in the SDL and it is seen just as an additional process activity (Marmor-Squires & Rougeau, 1988). To obtain a secure product, the typical development approach needs to be extended. For example, the OWASP's Enterprise Security API Project (ESAPI) addresses this problem by providing a set of APIs to interface all security controls needed to build a secure application including input validation, output encoding, error logging and detection (Williams, 2008). These APIs include toolkits for the major programming languages and can be used by software developers during the SDL to increase security with minimum intrusion in their development process.

Some other researchers propose secure development guidelines (Auger, 2007), like the OWASP's "A Guide to Building Secure web Applications and web Services" (Wiesmann, Curphey, Stock, & Stirbei, 2005) and the "Complete web Application Security: Phase 1–Building web Application Security into Your Development Process" (SPI Dynamics, Inc., 2002). SANS series of working papers in Application Security describe a checklist of twelve methods to avoid two of the most important classes of mistakes done by developers: poor input validation and output filtering (Kim & Skoudis, 2009). These bugs affect the input and the output of web applications, protecting both the back-end mechanisms including the storage of malicious data, and the user through what is presented and executed by the web browser. In fact, solving these two problems would mitigate SQL Injection and XSS, as well as many other common web application problems. They propose a set of 10 best coding practices, and they also highlight the need to perform static analysis and penetration testing to secure the web application, which are common procedures among SSDLs.

Contrary to some beliefs, using a SSDL becomes profitable in the long term: "it is much cheaper to prevent than to repair" (McGraw, 2006). The use of a SSDL reduces the overall cost of development because it allows finding and eliminating vulnerabilities early in the process (Howard & Lipner, 2006; Microsoft Corporation, 2009a). A case study of client's data presented by Fortify suggests that the cost of fixing critical vulnerabilities later in the process, after releasing the software, is about 100 times more onerous than fixing vulnerabilities earlier in the requirements phase (Meftah, 2008). This trend was also observed in the data on software errors collected by Barry Boehm, although the benefit may be "only" 5 times higher (Boehm & Basili, 2001). Boehm's data covers the more general case of fixing all software errors, whereas the Fortify data is specific to critical vulnerabilities. Another report, this time prepared by RTI for the National Institute of Standards and Technology states that the cost of eliminating vulnerabilities increases all the way from design stage to post release (RTI, 2002). In the report, the cost at post release is double than at beta test and 30 times more than at design stage. The DIMACS Workshop on Software Security report refers to the following relative cost expenditures for lifecycle stages: design is 15%, implementation is 60% and testing is 25%. The same report also quotes an IBM study stating that the relative cost weightings are (Mead & McGraw, 2003): design = 1, implementation = 6.5, testing = 15 and maintenance = 100. According to a group of researchers from MIT, Stanford University and @Stake quoted by (Berinato, 2002) it is possible to have a Return Of Investment (ROI) of 21% at the design stage, 15% at the implementation stage and 12% at the testing stage. Although the values may vary, all these studies support that it is far less expensive to fix errors and vulnerabilities early in the start of the software development than after the software has become operational. It is, therefore, quite clear that the most effective security investment is the one spent in earlier phases of the lifecycle, although it must also be present through all the process till the end of the life of the software.

These requirements are present in development lifecycles with security built in. The following sections present an overview of important SSDLs in use nowadays and four initiatives aimed at providing security and control that can be integrated in existing SDLs with lesser effort.

OWASP COMPREHENSIVE, LIGHTWEIGHT APPLICATION SECURITY PROCESS (CLASP)

CLASP has been, since 2006, an OWASP project led by Pravir Chandra (who also developed OpenSAMM and BSIMM that will be discussed later). CLASP consists of a set of components with formalized security best practices that covers the entire SDL (not just development), so that security concerns can be adopted from the early stages of the SDL used by the organization (OWASP Foundation, 2006). This set of 24 security related activities that can be easily integrated into the SDL of the application allows systematically addressing security vulnerabilities. Eleven CLASP resources provide tools and other artifacts to help automate the process wherever possible. This SSDL heavily relies on the organization of the project team in roles where each one has a perfectly defined set of activities and responsibilities that they have to take care of. This is done in contrast with other SSDLs where these activities are part of a development step of the SSDL.

The CLASP is organized into high-level perspectives of the CLASP Process called CLASP Views (Fig. 1):

1. **Concepts View** - Defines that the basic security services that must be satisfied for each resource are: authorization, confidentiality, authentication (identity establishment and integrity), availability, accountability, and non-repudiation. This is done following seven application security best practices:
    a. Institute awareness programs.
    b. Perform application assessments.
    c. Capture security requirements.
    d. Implement secure development practices.
    e. Build vulnerability remediation procedures.
    f. Define and monitor metrics.
    g. Publish operational security guidelines.
2. **Role-Based View** - Shows how a project team should execute security issues depending on the specific responsibilities of every role (project managers, security auditors, developers, architects, testers, and others). The designer, architect and project manager roles are the ones that need to be trained specifically for security, mainly logical bugs. Developers only need to code right, without bugs, following the policies, standards, and guidelines in place in the organization.
3. **Activity-Assessment View** - Maps the various roles with the specific security related process activities (there are 24 of them) they have to implement. These activities are: Institute security awareness program, Monitor security metrics, Specify operational environment, Identify global security policy, Identify resources and trust boundaries, Identify user roles and resource capabilities, Document security-relevant requirements, Detail misuse cases, Identify attack surface, Apply security principles to design, Research and assess security posture of technology solutions, Annotate class designs with security properties, Specify database security configuration, Perform security analysis of system requirements and design (threat modeling), Integrate security analysis into source management process, Implement interface contracts, Implement and elaborate resource policies and security technologies, Address reported security issues, Perform source-level security review, Identify, implement and perform security tests Verify security attributes of resources, Perform code signing, Build operational security guide, Manage security issue disclosure process.
4. **Activity-Implementation View** - Details each one of the 24 role-based security related process activities.
5. **Vulnerability View** - Detailing the consequences, problem types, exposure periods, avoidance and mitigation techniques of security vulnerabilities. It considers 104 vulnerability

types, their categories, exposure periods, consequences, platforms affected, resources, risk assessment, avoidance and mitigation periods.

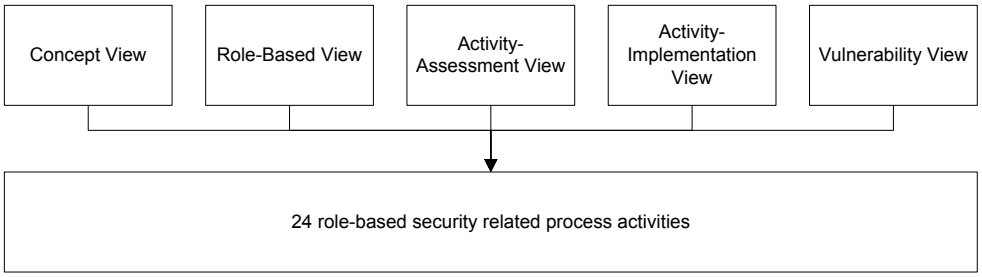

Fig. 1. The CLASP organization. (Adapted from (OWASP Foundation, 2006)).

The set of 24 activities is detailed in the free to download book (OWASP Foundation, 2006). This book contains which activities are bound to each role, among all the information needed to implement this SSDL into an organization, like the taxonomy used, vulnerabilities, detailed actions, use cases, etc. It also has a section on CLASP resources explaining the most important concepts, which can be used as a starting point to improve security training and security awareness: basic principles, examples, core security services, worksheets covering sample coding guidelines and system assessment, sample roadmaps, etc. To help move from the current SLD to CLASP, the roadmap section provides a set of steps for organizations that want a minimum impact on their ongoing projects (containing only 12 activities) and for organizations that want to apply it holistically (containing 20 activities).

MICROSOFT SECURE DEVELOPMENT LIFECYCLE

The Microsoft SSDL is a mandatory methodology in use by Microsoft since 2004, used to deliver more reliable software with security and privacy built in (Howard & LeBlanc, 2003; Howard & Lipner, 2006; Microsoft Corporation, 2009a). Over 50% of Microsoft flaws were design flaws (Mead & McGraw, 2003), so it is not a surprise that their SDL is heavily based on threat modeling (also known as threat analysis or risk analysis) done in the early stages of development. Threat modeling is an application security auditing procedure consisting in formally identifying and mapping all the possible attack vectors of the application. It helps reduce the number and severity of vulnerabilities in the application code, including design ones, according to results provided by Microsoft (Microsoft Corporation, 2009a).

The Microsoft SSDL is based on the following guiding principles (Microsoft Corporation, 2008):

1. **Secure by Design** - Secure architecture, design and structure; Threat modeling and mitigation; Elimination of vulnerabilities; Improvements in security.
2. **Secure by Default** - Least privilege; Defense in depth; Conservative default settings; Avoidance of risky default changes; Less commonly used services off by default.
3. **Secure in Deployment** - Deployment guides; Analysis and management tools; Patch deployment tools.
4. **Communications** - Security response; Community engagement.

The Microsoft SSDL is based on a set of activities for each phase and it can be applied incrementally into an existing ongoing development process (Fig. 2).

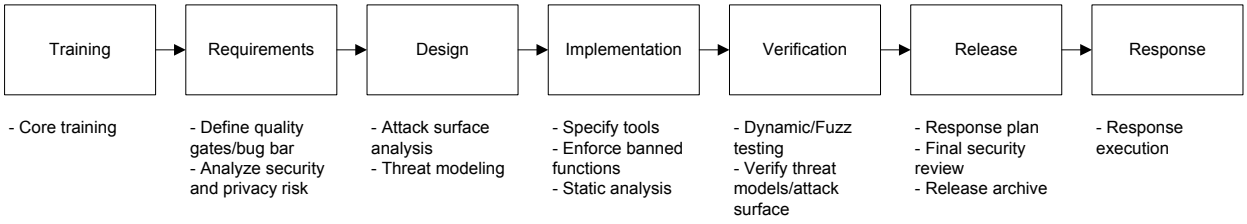

Fig. 2. The Microsoft Security Development Lifecycle. (Adapted from (Microsoft Corporation, 2009a)).

One of the core aspects of this SSDL is the use of the threat modeling theory. Threat modeling focuses on a high level of the development, in the design and architecture of the product, and helps uncover design issues and point out which components are at risk, before implementation (Howard, 2009). Threat modeling describes the attack surface, the threats of the system and the assets that may be compromised from the point of view of the attacker. The potential attack vectors (threats) are added to the model and this enables the fix of design flaws, therefore preventing such attacks. OWASP also uses this threat modeling process because it is easy to learn and adopt (Wiesmann et al., 2005). Other SSDLs also use threat modeling, although the actual implementation may differ from the Microsoft one.

The risk of each threat can be estimated using the DREAD (Damage potential, Reproducibility, Exploitability, Affected users, Discoverability) method, rating numbers from 1 to 10 for each DREAD item. The DREAD is rather subjective to apply, and needs a high degree of expertise. This is the main reason it was replaced by a heuristic model derived from Microsoft Security Response Center bulletin ratings. It contains four rankings: Critical, Important, Moderate and Low. These rankings are much easier to apply and are also more effective, since they are based on many years of experience. To help in the process of applying the Microsoft threat analysis and modeling and estimate the risk Microsoft provides a set of tools.

According to Rauscher and colleagues, threat modeling has limitations on the real ability to obtain the necessary data and the mitigation is based on previous known attacks, so a creative attacker may still be able to be successful (Rauscher, Krock, & Runyon, 2006). To cope with this, the authors propose a vulnerability analysis using the eight tenants already used by the communication industry: human, policy, hardware, software, networks, payload and power. This methodology allows the address of a finite number of general classes of vulnerabilities, instead of an infinite number of specific threats that can exercise those vulnerabilities. So, working together threat analysis and vulnerability analysis can produce better results.

To uncover design flaws the system is decomposed in components and each component is analyzed according to each one of the STRIDE approach where the effect of the bug is classified using the threat groups that have their related opposite security property (Table 1). When a threat is found, the bug enabling it is corrected in the code.

Threat	**Security Property**
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-repudiation
Information disclosure	Confidentiality
Denial of service	Availability
Elevation of privilege	Authorization

Table 1. STRIDE threat model.

To make this SSDL easier to apply and to help check the phases of the process, it is integrated into the Microsoft's development tool VisualStudio.NET (Microsoft Corporation, 2009b).

SOFTWARE SECURITY TOUCHPOINTS

The Cigital's Software Security Touchpoints is a manageable set of seven best practices, proposed in 2004, that can be applied to the SDL being used by the organization (waterfall, spiral, etc.). In the book "Software Security" (McGraw, 2004), the author presents the best practices procedures (touchpoints) showing how they can easily be applied during the existing SDL in use in the organization (Fig. 3).

For the touchpoints, two levels of software bugs are considered: source code level and architectural level (McGraw, 2004, 2006). The two most important touchpoints are source code analysis and architectural risk analysis, because they focus on bugs found in the code and in the design, respectively. The touchpoints, in order of effectiveness are the following:

1. **Code review** - Using static analysis tools.
2. **Risk analysis** - Based on attack patterns and threat models.
3. **Penetration testing** - Using the black-box approach that should also consider the architecture of the system.
4. **Risk-based security tests** - With traceability back to requirements.
5. **Abuse cases** - Describing the system behavior under attack.
6. **Security requirements** – Security must be present in the requirements, as well.
7. **Security operations** - Monitoring for security breaks during the use of the system.

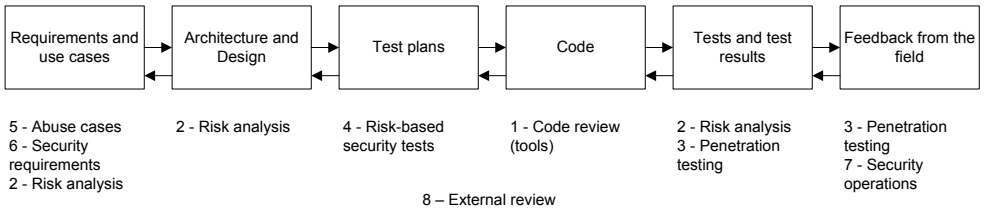

Fig. 3. The software security touchpoints. (Adapted from (McGraw, 2004)).

Although External Review (the last, in the effectiveness order) is considered outside the design team, it is also necessary. Security training of each best practice is also a concern of this model.

An extensive set of articles themed Software Security Best Practices Building Security In for IEEE Security & Privacy were written by Gary McGraw, detailing the several best practices of the model ((Arkin, Stender, & McGraw, 2005; Barnum & McGraw, 2005; B. Chess & McGraw, 2004; Hope, McGraw, & Anton, 2004; McGraw, 2004; Potter & McGraw, 2004; Taylor & McGraw, 2005; Verdon & McGraw, 2004)).

INITIATIVES ON BUILDING AND MAINTAINING A SECURE SOFTWARE PRODUCT

For an organization, it is difficult and costly (at least in the short term) to change their development methodology from the ground up, even if the objective is to deliver a higher quality product concerning security. To help with this migration process, several initiatives arouse aiming to integrate security in the SDL currently being used in the organization. Some of the most important initiatives are the

OpenSAMM, the BSIMM, the SAFECode and the Securosis. One interesting concern among all of them is the lesson learned from the industry, so they are based on empirical data collected over the years from relevant software development houses. Some of these initiatives present not only an improvement in security, but also the means to measure or benchmark the current state of the evolution of the software development process inside the organization, concerning security. This helps verify and compare the current state of the development process and to specify goals for the future that can be measured.

Open Software Assurance Maturity Model (OpenSAMM)

The OpenSAMM was developed by software security consultant Pravir Chandra and it is intended to be easy to follow even by non-security experts. It includes a simple, well-defined and measurable maturity model for the organization (Pravir Chandra, 2009). It was originally founded by Fortify but it is now part of the OWASP. It was proposed in 2009, it is not tied to vendors but has a lot of industry participation, it is open and driven by the community.

The OpenSAMM model is based on four core Business Functions involved in the software development, each one with a set of three Security Practices (Fig. 4). The Security Practices are activities related to security that build assurance for the related Business Function. Each Security Practice has three Maturity Levels (or objectives) with well-defined specific Objectives, Activities, Results, and increasingly stringent Success Metrics, Costs, Personnel and Related Levels.

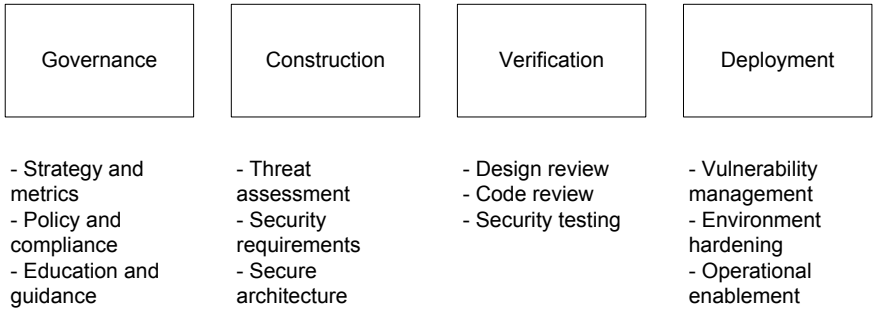

Fig. 4. The Software Assurance Maturity Model (SAMM). (Adapted from (Pravir Chandra, 2009)).

The OpenSAMM model can be used as a benchmark to assess a security assurance program and create a scorecard showing its evolution. This ability to precisely score the security level of an organization and its evolution over time is a major advantage of the model. The assessment can simply be done for each Practice by scoring the answers, but a more detailed assessment can be done with additional auditory work. As an example, the interview template that helps determine the organization's current maturity level can be easily obtained from the web (Coblentz, 2009). This model is also prepared to ease the implementation of a software assurance program, by providing a roadmap that can be tailored for each organization need.

Building Security In Maturity Model (BSIMM)

The Building Security In Maturity Model (BSIMM) is a model developed in 2009 and derived from a beta version of the OpenSAMM (McGraw, Chess, & Migues, 2009). It is a practical approach based on empirical evidence and data observation of nine software security initiatives from financial services, independent software vendors, and technology firms (Adobe, EMC, QUALCOMM, Google, Wells Fargo, Microsoft, DTCC and two other undisclosed companies). Unlike other SSDL methodologies, the BSIMM does not contain a theoretical compilation of best practices. It is a real-world collection of actual practices

performed in the field. The nine underlying organizations follow different SSDLs and the best practices are derived from their experiences. So, regardless of methodology, most of the theoretical best practices proposed by other SSDLs are actually present in the BSIMM, and they all share a common ground. In fact, Cigital (that proposed the Software Security Touchpoints already discussed) is one of the partners of the BSIMM, along with Fortify (founder of the OpenSAMM). BSIMM is also considered the standard for financial firms by the Financial Services Technology Consortium and used by the U.S. Department of Homeland Security.

The BSIMM framework is called Software Security Framework (SSF) and consists of twelve normalized Practices each one with several activities associated to them (with objectives and activities) and grouped in four Domains (Fig. 5).

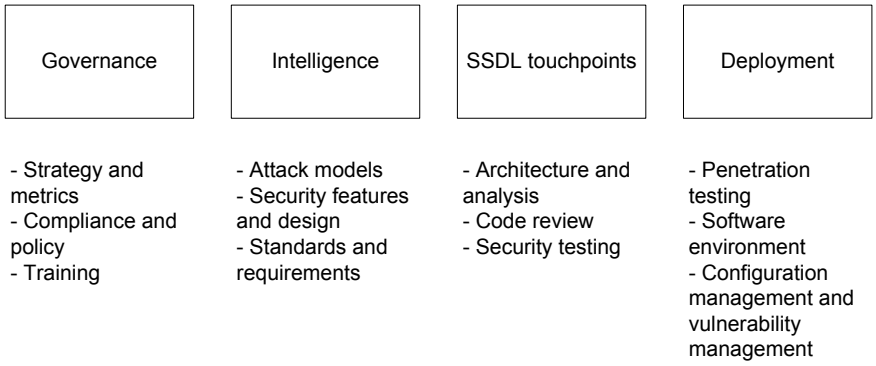

Fig. 5. BSIMM's Software Security Framework. (Adapted from (McGraw et al., 2009)).

Like the OpenSAMM, the BSIMM considers three levels of maturity with increasing security demanding and each one contains a set of activities within each practice, from an overall of 110 activities. This model can also be used to benchmark different organizations and prioritize changes according to their score in the maturity level of each one of the twelve practices. Like the OpenSAMM, this ability to allow bechmarking the maturity of the security practices and its evolution is one of the advantages of this model. However, BSIMM is the result of the underlying field study showing the practices that are really used by some good referenced activities common to all of the leading software security initiatives in order to obtain a secure process of development.

Organizations can use the BSIMM skeleton to obtain a glance of the maturity model level during an assessment. It consists of the twelve Practices organized into the three maturity levels with their Objectives and Activities. This model is simpler to apply than the OpenSAMM but it necessarily lacks some information, like the guidance on how to measure and rank the Activities in order to obtain a comparable benchmark that can be used across the organizations.

Software Assurance Forum for Excellence in Code (SAFECode)

Like the BSIMM, the Software Assurance Forum for Excellence in Code (SAFECode) is an industry-led consortium formed in 2007, including the following members: EMC, Juniper Networks, Microsoft, Nokia, SAP, and Symantec. It is dedicated to increase trust in information and communication technology products and services. This consortium produced some publications focusing on secure development methods and practices (SAFECode, 2008a), an overview of industry best practices like the BSIMM model (SAFECode, 2008b) and principles of secure software development during training (SAFECode, 2009).

The SAFECode guide contains a list of SSDL best practices, actually being executed by the industry, which are proven to help deliver secure products (Fig. 6). In the best practices for secure software programming, the SSDL focuses on the following aspects (SAFECode, 2008a):

1. **Requirements** - Including training in secure development and testing.
2. **Design** - With threat analysis before code commit.
3. **Programming** - Including static and dynamic code analysis tools and manual code review, input and output validation.
4. **Testing** - Consisting of fuzzing, penetration testing and external assessment.
5. **Code Integrity and Handling** - Focusing on access principles like the least privilege access, separation of duties and persistent protection.
6. **Documentation** - Defining software security best practices and how to configure the software for security.

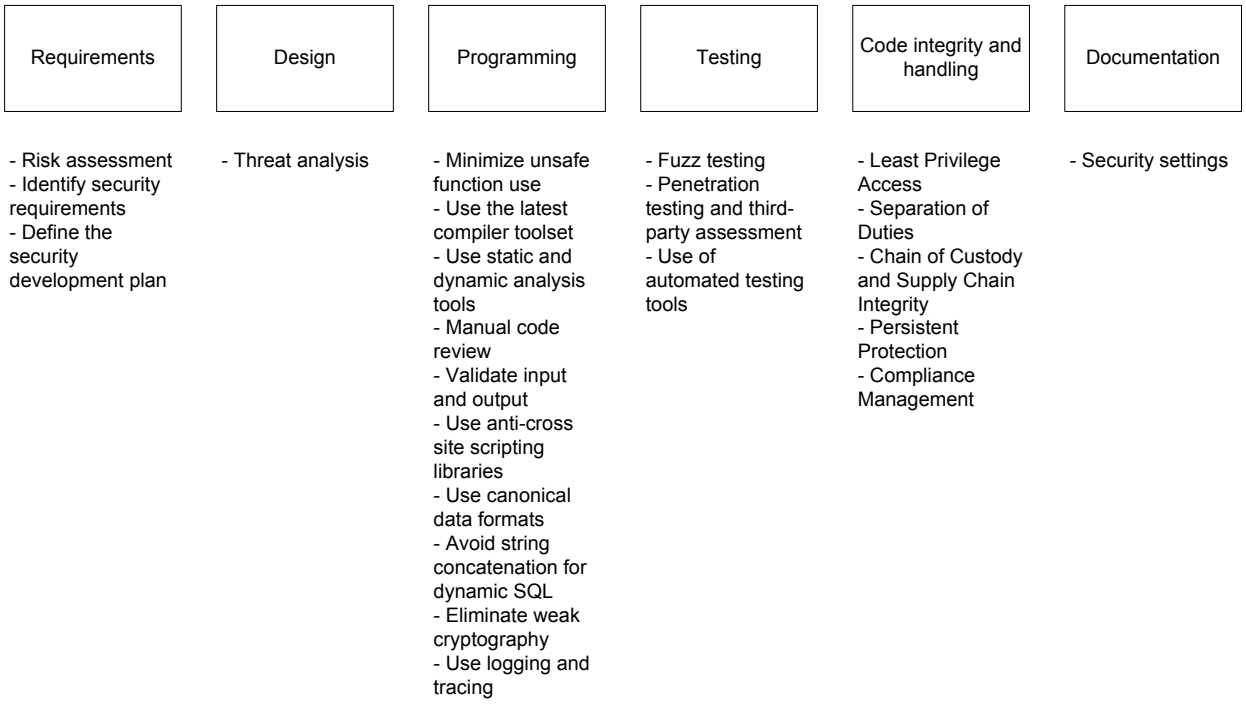

Fig. 6. SAFECode's Best Practices. (Adapted from (SAFECode, 2008a)).

The SAFECode also highlights the need for leaders to promote the use of these best practices to create a security aware conscience among everyone involved in the software process development.

Securosis building a web application security program

Securosis released in 2009 a whitepaper sponsored by Core Security, Imperva and Qualys, about building a security program targeted specifically for web application development (Securosis, 2009). The paper proposes a SSDL with focus on the underfunding problem of web application security. The proposed SSDL is practical, inexpensive and presents the type of tools that should be used in the three stages of the proposed development cycle (Fig. 7).

1. **Secure Development** - It focuses on initial procedures needed to develop the application like gathering the requirements, design, implementation and quality assurance. Static analysis and dynamic analysis tools are covered to partially help automate this stage.
2. **Secure Deployment** - It is at this stage where code complies with specifications and it is ready for vulnerability assessment (find security bugs) and penetration tests (classify and exploit security bugs).
3. **Secure Operation** - When the application is deployed, preventive tools must be used in order to monitor the operation. For this matter, web application firewalls, web application and database activity monitoring tools are used.

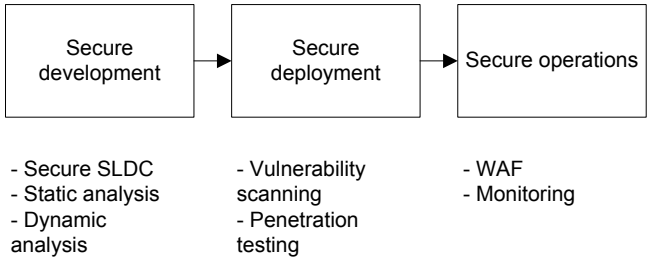

Fig. 7. Securosis model. (Adapted from (Securosis, 2009)).

COMPARING THE SECURITY DEVELOPEMENT PROPOSALS

Brook's Law on project management says (Brooks, 1995): "Adding manpower to a late software project makes it later". The same idea also applies to security: just adding security manpower to a late software project makes it later. And it still becomes unsecure, because patches do not always work very well (Arbaugh, Fithen, & McHugh, 2000).

A common expression among SSDL enthusiasts to highlight the differences between their thinking and current building methods is that software security is not the same as secure software (Gollmann, 1999). In fact, having security functions in a piece of software, like password authentication for example, does not make by itself the application secure. It is also not sufficient to follow a single security procedure, because all of them have their benefits, but also their weaknesses. Developers need to coordinate together several tasks to be able to achieve a secure application (Barnett, 2008). It is this type of integration that a SSDL is supposed to provide.

Security concern must be present during all the phases of the software development lifecycle and security cannot be seen just as a minor issue. In fact, it must be a design goal (Jayaram & Aditya, 2005) and this is represented well in OWASP's (OWASP Foundation, 2006), Microsoft's (Howard & LeBlanc, 2003) and McGraw's (McGraw, 2006) software development lifecycles.

Security vulnerabilities must be mitigated during the development lifecycle, before the software is released. Code Inspection and Penetration Testing represent two key quality assurance procedures that must be used to detect security vulnerabilities. Code inspection is a white-box approach that consists in the formal review of the application code by an external team. Penetration testing is a black-box approach consisting in a set of tests made from the point of view of the users, where the external team tries to find all the possible vulnerable entry points of the application. Penetration testing can be performed manually or it can be done using automated tools, although even top commercial products have a high rate of false positive (non vulnerabilities that are tagged as vulnerabilities) and false negative (vulnerabilities that are

not identified) values (Ananta Security, 2009; Fonseca, Vieira, & Madeira, 2007; WhiteHat Security Inc., 2008).

Because of the novelty of the development approach using a SSDL it is still hard to find results derived from real data that can compare the improvement of the quality of the final product. However, as one of the first solid establishments in the area, Microsoft has published the results of the number of critical bugs found in some of the products they developed (Fig. 8). Windows 2003 post-SDL had a decrease of over 60% of critical bugs compared to Windows 2000 pre-SDL, SQL Server 2000 had a decrease of over 80% compared to the releases pre-SDL and post-SDL and Exchange Server 2000 had a decrease of 75% compared to the releases pre-SDL and post-SDL. Taking into account the overall results, there is a 66% decrease of the number of critical bugs after applying the SSDL in their development.

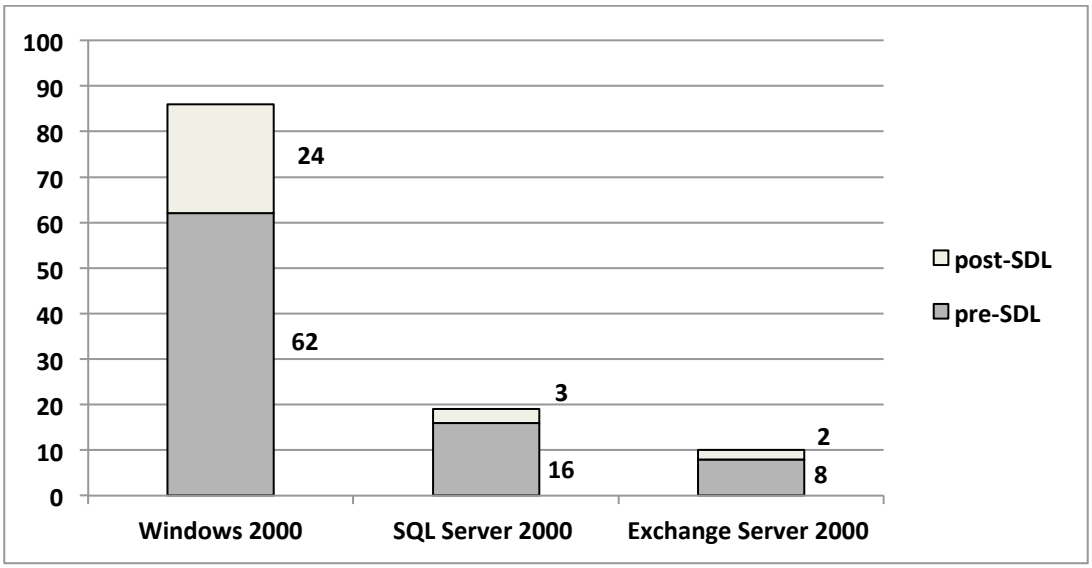

Fig. 8. Windows 2000, SQL Server 2000, Exchange Server 2000 pre- and post-SDL critical and important security bulletins. (Adapted from (Lipner & Howard, 2005)).

The research area of SSDL process is still in its early stages with lots of new ideas flowing from one proposal to another. There is a mixture of organizations involved in more than a single SSDL project like Microsoft, Cigital, Fortify, OWASP, etc.; and some SSDLs share the same guru, like Gary McGraw, Pravir Chandra, and Michael Howard. Some SSDL methodologies derive from a fork of the early stages of other SSDL methodologies and share some of the others core ideas, like the Touchpoints, OpenSAMM and BSIMM.

This interest in better secure code practices and tools is shown by big investments in the security area and large acquisitions. In fact, according to Brian Chess, 2007 was a turning point because "It was the first year there was a bigger market for products that help you get code right than there was for products that help you demonstrate a problem exists" (Brian Chess, 2008). This is a good direction to follow: prevent the problem instead of chasing it in order to fix it after the damage.

Table 2 presents an overview of the SSDLs described earlier in the chapter. The data shows that the focus on SSDLs is a new concern largely sparked after the 2002 Bill Gates trustworthy computing memo and the start of Microsoft security push. As time goes by, this subject is getting more interest from the industry: close to half of these major security development methodologies analyzed emerged in 2009.

	Name	Year	Highlights
SSDL	CLASP	2006	24 formalized security best practices that cover the entire SDL.
	Microsoft SDL	2004	Complete integrated SDL heavily based on risk analysis and based on a set of activities for each phase. Integrated in the Microsoft development suite. Suited for Operating System developers and large software houses.
	Touchpoints	2004	7 best practices that can be applied to the existing SDL.
Security Initiative	OpenSAMM	2009	Measurable maturity model with 3 maturity levels and 12 security practices. Can be used as a benchmark.
	BSIMM	2009	12 actual best practices used by the industry with 3 maturity levels. Can be used as a benchmark.
	SAFECode	2008	Overview of industry best practices with an emphasis on leadership.
	Securosis	2009	SSDL targeted specifically for web applications and focusing on automation by using tools.

Table 2. SDL security proposals comparison.

Although there is still not enough data to make a detailed comparison among the various SSDLs presented, some remarks can be made. We can see that one of the constraints in applying a SSDL has to do with the cost that includes the need of changing processes, training and delaying the deliverable of the final product. Some SSDLs could be used in larger projects, whereas others are more suitable to smaller companies that cannot afford to make profound changes in the way the software is built.

The Microsoft SSDL can be applied to very large projects, like Operating Systems (OS) and big applications. They provide a set of tools to help in the process, integrated in their Visual Studio environment, but they must be run in the Windows OS. CLASP is lighter and can be adopted for smaller projects involving a fewer set of resources. It can be used as an SSDL or it can be easily integrated into an existing SDL with a reduced set of steps. The Touchpoints can be perfectly integrated into an existing SDL within the organization, helping to provide security to its software projects. The Touchpoints present activities that should be applied to the various artifacts created during the development of software. They are independent of the target of the company, so they can be applied in all software development situations.

The other software initiatives focus on providing a set of best practices that are really being used by big references in the software development industry, which are proven to provide a good balance between the cost and the benefit they provide. OpenSAMM and BSIMM also provide means to benchmark the actual software development maturity concerning the adoption of the proposed practices. While SAFECode advises the need to have a leadership tailored for security and that this will drive the mentality change of the rest of the team in building a safer software product, Securosis focuses on existing tools that can help automate most of the processes required by implementing security in the SDL.

CONCLUSION

Every contribution towards building and maintaining a safer software product is welcomed, given the current state of insecurity, namely in web applications. All of the proposals analyzed in this chapter are worth mentioning and they all provide a step up in the level of security of the final product. Although they have different views on how to achieve their goals, in essence they have more in common than they have differences. This common ground constitutes the basic principles and best practices they all share, which can be seen as the core needed to increase security during software development:

1. **Training for security.**
2. **Architectural review.**
3. **Source code review.**
4. **Penetration testing.**
5. **Documentation and security policies.**

Applications, especially on the web, are a constant target for hackers and are never safe given that new vulnerabilities and exploitation techniques are being discovered constantly in the technologies and components they use. This evolving environment quickly turns thought to be secure applications into undoubtedly unsecure applications. Even for applications developed with a SSDL, they may become vulnerable shortly after delivery. This is one of the reasons why the development process does not end with delivery and continues though the maintenance. Maintenance after deployment and during the entire working life of the application should be a mandatory requirement.

To build secure software, security must be present from the early stages of the software development taking into account both secure mechanisms and design for security. The use of a software development lifecycle considering security is of utmost importance if the objective is not only the prevention of security bugs, but also higher-level problems, like architectural, component interaction and broken access control over tiers. The security work should be applied since the early stages of development, during the definition of the requirements, architecture, design, coding, testing, validation, measurement, and maintenance of the software. This way of developing secure applications is not only cheaper in the long term, but also has already proven results from the industry.

REFERENCES

Ananta Security. (2009). *Web Vulnerability Scanners Comparison*. Retrieved from http://anantasec.blogspot.com/2009/01/web-vulnerability-scanners-comparison.html

Arbaugh, W. A., Fithen, W. L., & McHugh, J. (2000). Windows of vulnerability: a case study analysis. *Computer*, *33*(12), 52–59. doi:10.1109/2.889093

Arkin, B., Stender, S., & McGraw, G. (2005). Software penetration testing. *IEEE Security & Privacy*, *3*(1), 84–87. doi:10.1109/MSP.2005.23

Auger, R. (2007). Writing Software Security Test Cases. Retrieved February 18, 2009, from http://www.qasec.com/cycle/securitytestcases.shtml

Baker, W. H., Hutton, A., Hylender, C. D., Novak, C., Porter, C., Sartin, B., Tippett, P., et al. (2009). *The 2009 Data Breach Investigations Report*. Verizon Business RISK Team.

Barnett, R. (2008). ModSecurity Blog: Is Your Website Secure? Prove It. Retrieved May 16, 2009, from http://www.modsecurity.org/blog/archives/2008/01/is_your_website.html

Barnum, S., & McGraw, G. (2005). Knowledge for software security. *IEEE Security & Privacy*, *3*(2), 74–78. doi:10.1109/MSP.2005.45

Berinato, S. (2002). CIO - Return on Security Spending, pp. 43–52.

Boehm, B. (1986). A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, *11*(4), 14–24. doi:10.1145/12944.12948

Boehm, B., & Basili, V. R. (2001). Software Defect Reduction Top 10 List. *Computer*, *34*(1), 135–137.

Brooks, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition* (2nd ed.). Addison-Wesley Professional.

Chess, B., & McGraw, G. (2004). Static analysis for security. *IEEE Security & Privacy*, *2*(6), 76–79. doi:10.1109/MSP.2004.111

Chess, Brian. (2008). Space Race. *My Security Planet » Fortify blog*. Retrieved May 12, 2009, from http://rgaucher.info/planet/Fortify_blog/2008/08/13/

CNSS Secretariat. (2006). *National Information Assurance (IA) Glossary*. Committee on National Security Systems.

Coblentz, N. (2009). SAMM Assessment Interview Template. *SAMM Assessment Interview Template*. Retrieved September 18, 2009, from http://spreadsheets.google.com/pub?key=rYpVqQR3026Zu4DNg8LBIwg&gid=3

Epstein, J. (2009). *What Measures Do Vendors Use for Software Assurance?* Build Security In. Carnegie Mellon University. Retrieved from https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/business/1093-BSI.html

Fonseca, J., Vieira, M., & Madeira, H. (2007). Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks. *13th Pacific Rim International Symposium on Dependable Computing, 2007. PRDC 2007* (pp. 365–372).

Gollmann, D. (1999). *Computer Security* (1st ed.). John Wiley & Sons. Retrieved from http://www.wiley.com/legacy/compbooks/catalog/97844-2.htm

Higgins, K. J. (2009). The Rocky Road To More Secure Code. *DarkReading*. Retrieved September 18, 2009, from http://www.darkreading.com/security/app-security/showArticle.jhtml?articleID=216403548&pgno=1&queryText=&isPrev=

Hope, P., McGraw, G., & Anton, A. I. (2004). Misuse and abuse cases: getting past the positive. *IEEE Security & Privacy*, *2*(3), 90–92. doi:10.1109/MSP.2004.17

Howard, M. (2009). A Conversation About Threat Modeling. Retrieved May 16, 2009, from http://msdn.microsoft.com/en-us/magazine/dd727503.aspx

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

Howard, M., & Lipner, S. (2006). *The Security Development Lifecycle*. Microsoft Press.

Jayaram, K. R., & Aditya, P. M. (2005). *Software Engineering for Secure Software - State of the Art: A Survey* (CERIAS TR 2005-67). Purdue University. Retrieved from https://www.cerias.purdue.edu/apps/reports_and_papers/view/2884

Kim, F., & Skoudis, E. (2009). *Protecting Your Web Apps: Two Big Mistakes and 12 Practical Tips to Avoid Them*. SANS Institute.

Lanowitz, T. (2005). *Now Is the Time for Security at the Application Level*. Gartner Group. Retrieved from http://www.sela.co.il/_Uploads/dbsAttachedFiles/GartnerNowIsTheTimeForSecurity.pdf

Lipner, S., & Howard, M. (2005). The Trustworthy Computing Security Development Lifecycle. *Microsoft Developer Network*. Retrieved September 24, 2009, from http://msdn.microsoft.com/en-us/library/ms995349.aspx

Marmor-Squires, A. B., & Rougeau, P. A. (1988). Issues in Process Models and Integrated Environments for Trusted Systems Development. *Proceedings of the 11th National Computer Security Conference* (pp. pp. 109–113). Presented at the Proceedings of the 11th National Computer Security Conference, United States Government Printing Office.

Martin, J. (1991). *Rapid application development*. Indianapolis, IN, USA: Macmillan Publishing Co., Inc.

McGraw, G. (2004). Software security. *IEEE Security & Privacy*, *2*(2), 80–83. doi:10.1109/MSECP.2004.1281254

McGraw, G. (2006). *Software Security: Building Security In*. Addison-Wesley Professional.

McGraw, G., Chess, B., & Migues, S. (2009). *Building Security In Maturity Model*. Fortify & Cigital. Retrieved from http://bsi-mm.com/

Mead, N. R., & McGraw, G. (2003). *The DIMACS Workshop on Software Security*. DIMACS Center.

Meftah, B. (2008). *Business Software Assurance: Identifying and Reducing Software Risk in the Enterprise*. Presented at the 9th Semi-Annual Software Assurance Forum. Retrieved from https://buildsecurityin.us-cert.gov/swa/downloads/Meftah.pdf

Microsoft Corporation. (2008). *MICROSOFT SECURITY DEVELOPMENT LIFECYCLE (SDL) Version 3.2*. Microsoft Corporation.

Microsoft Corporation. (2009a). The Microsoft Security Development Lifecycle (SDL). Retrieved March 23, 2009, from http://msdn.microsoft.com/en-us/security/cc448177.aspx

Microsoft Corporation. (2009b). SDL Process Template. *MSDN*. Retrieved May 22, 2009, from http://msdn.microsoft.com/en-us/security/dd670265.aspx
OWASP Foundation. (2006). *OWASP - CLASP* (1.2 ed.). OWASP Foundation. Retrieved from http://www.owasp.org/index.php/Category:OWASP_CLASP_Project
Potter, B., & McGraw, G. (2004). Software security testing. *IEEE Security & Privacy*, *2*(5), 81–85. doi:10.1109/MSP.2004.84
Pravir Chandra. (2009). *Software Assurance Maturity Model: A guide to building security into software development* (1.0 ed.). OpenSAMM Project. Retrieved from http://www.opensamm.org/
Rauscher, K. F., Krock, R. E., & Runyon, J. P. (2006). Eight ingredients of communications infrastructure: A systematic and comprehensive framework for enhancing network reliability and security. *Bell Labs Technical Journal*, *11*(3), 73–81.
Royce, W. (1970). Managing the Development of Large Software Systems. *Proc. IEEE Wescon* (pp. 1–9).
RTI. (2002). *Planning Report 02-3 The Economic Impacts of Inadequate Infrastructure for Software Testing*. NIST. Retrieved from http://www.nist.gov/director/prog-ofc/report02-3.pdf
SAFECode. (2008a). *Fundamental Practices for Secure Software Development*. SAFECode. Retrieved from http://www.safecode.org/publications.php
SAFECode. (2008b). *Software Assurance: An Overview of Current Industry Best Practices*. SAFECode. Retrieved from http://www.safecode.org/publications.php
SAFECode. (2009). *Security Engineering Training*. SAFECode. Retrieved from http://www.safecode.org/publications.php
Securosis. (2009). *Building a Web Application Security Program*. Securosis. Retrieved from http://securosis.com/research/publication/building-a-web-application-security-program/
Sommerville, I. (2010). *Software Engineering* (9th ed.). Addison Wesley.
SPI Dynamics, Inc. (2002). *Complete Web Application Security: Phase 1–Building Web Application Security into Your Development Process*. SPI Dynamics, Inc. Retrieved from http://cnscenter.future.co.kr/resource/rsc-center/vendor-wp/Spidynamics/Webapp_Dev_Process.pdf
Taylor, D., & McGraw, G. (2005). Adopting a software security improvement program. *IEEE Security & Privacy*, *3*(3), 88–91. doi:10.1109/MSP.2005.60
Verdon, D., & McGraw, G. (2004). Risk analysis in software design. *IEEE Security & Privacy*, *2*(4), 79–84. doi:10.1109/MSP.2004.84
WhiteHat Security Inc. (2008). *WhiteHat Website Security Statistic Reports* (No. 6th Edition). WhiteHat Security Inc. Retrieved from http://www.whitehatsec.com/home/resource/stats.html
Wiesmann, A., Curphey, M., Stock, A. van der, & Stirbei, R. (2005). *A Guide to Building Secure Web Applications and Web Services, V2.0.1*. OWASP Foundation. Retrieved from http://www.owasp.org/index.php/Developer_Guide
Williams, J. (2008). *Establishing a Security API for Your Enterprise (ALPHA version)*. OWASP Foundation.

KEY TERMS AND DEFINITIONS

Attack: Malicious and intentional interaction with the system exploiting a vulnerability in order to take advantage from it.

Best practices: Set of methodologies that should be followed during the software lifecycle in order to provide a better product.

Bug (in the software): Error in the software code that makes the software provide a service that deviates from the correct service, as stated in the specification.

Hacker: In this context, an attacker, a person that exploits the vulnerabilities of the system or tries to use it in other ways that were not intended by the developer.

Security: Set of properties of the software (Confidentiality, Integrity and Availability) that should be preserved, even when the system is under attack.

SSDL: Secure Software Development Lifecycle is a model to develop software with security embedded from the start to the end of the life of the software.

Vulnerability: Weakness in the system that may be exploited by an attacker to jeopardize one or more security properties of the system.