Developer-Friendly Software Security

Victoria McIntyre
Jungwoo Ryoo
Department of Information Sciences and Technology (IST), Penn State University, Altoona, Pennsylvania, U.S.A.

Kyung-Won Park
Hyoungshick Kim
Department of Computer Science and Engineering, Sungkyunkwan University, Seoul, Korea

Abstract

Software security is being increasingly considered important in the software development process. However, in practice, software security is often neglected due to inherent constraints such as time to market and budget. Various efforts have been undertaken to address the problem of enhancing the security quality of software. For example, managers can provide secure coding standards and guidelines, and encourage software engineers to follow these recommended practices. However, it is not feasible to expect every software engineer to become an expert in software security and apply one's security knowledge. A more realistic model is to minimize the software security–related burden imposed on software engineers by providing as much support as possible. A number of different approaches may be taken to provide this support, and the choice of methods depends on the size and nature of the organization developing the software. We refer to this new mind-set toward software security as "developer-friendly software security." This entry explores ways to promote the developer-friendly software security concept in a software development organization.

INTRODUCTION

Software security is a series of technical countermeasures adopted when building software to ensure that it is structurally secure and functions properly when subjected to malicious attacks. While traditional software engineering concentrates on dealing with errors due to accidents, software security should take malicious attackers into consideration. That is, software should be designed to be not only highly reliable against accidental errors but also secure against malicious attempts made by attackers through exploits.

To ensure security, software developers need to be aware of as many security concerns and threats as possible. For example, they should adopt the best secure coding practices to avoid introducing new vulnerabilities to their software. However, it is unrealistic to expect every software developer to become a security expert. A more realistic approach is to provide necessary guidance and support in the form of processes and tools to minimize the software security efforts made by developers. We advocate this more developer-friendly way of approaching software security and refer to it as developer-friendly software security. By making it easier for developers to introduce security controls, their overall software security practice is expected to improve, which can, in turn, lead to more secure software in general.

Our idea of developer-friendly software security stems from the works established in the usable security research community.[1] Several useful security practices are ignored, neglected, or not enforced due to the feasibility problems and difficulties associated with implementing the best security practices. For example, it might be more secure to change passwords every hour than every six months. However, this security principle seems to fail. In practice, it is even difficult for casual users to remember a new password every six months.

The same idea applies to software developers, too. Rather than giving software developers the additional burden of tackling software security challenges independently, it is significantly more systematic and realistic to provide an infrastructure to make it easier to adopt the best software security practices already available. Therefore, this entry will explore what could constitute this infrastructure to make the process of adopting the best software security practices as developer-friendly as possible.

DEVELOPER-FRIENDLY SOFTWARE SECURITY APPROACHES

Several approaches for software security do already exist, and these can be used to provide support for developers in their effort to make their software more secure. We use software development life cycle (SDLC) phases as our criteria to classify these existing support mechanisms. The SDLC phases are inception, elaboration, construction, and

Encyclopedia of Software Engineering DOI: 10.1081/E-ESE-120052061

transition.[2] During the inception phase, the emphasis is to elicit stakeholder requirements and to work on business modeling. During the elaboration phase, the focus is on the analysis of the collected requirements and the development of design solutions to satisfy the requirements. The construction phase concentrates on implementing the design decisions made during the elaboration phase. Finally, the transition phase focuses on deploying the software into the production environment, testing it, and maintaining the software product. In the following sections, we describe SDLC phase–specific support mechanisms for security. We also have a secure SDLC section in which we also discuss secure SDLC mechanisms that are more process-centric, and explain how mechanisms identified for each SDLC phase can fit together in a broader context in relation to the rest of the SDLC phases.

Inception Phase Support Mechanism

During the inception phase, there is an opportunity to elicit security requirements. Developers often miss this opportunity due to the lack of knowledge in how to effectively work with their stakeholders to collect security requirements. Methodologies are available to help fill in this gap, and we present two of them in the following sections.

SQUARE

Successful software engineering begins with a security mind-set from the project's inception. This mind-set literally saves organizations billions of dollars, along with the avoidance of the stresses of restoring vigor to a system plagued with security issues. The Software Engineering Institute's Networked Systems Survivability (NSS) Program has established a comprehensive suite of guidelines that address security during the inception stage of software development. This is known as the Security Quality Requirements Engineering (SQUARE) methodology, to be used for the elicitation and prioritization of security requirements.[3] SQUARE features a total of nine specific steps that work toward the goal of enhanced security as part of gathering requirements during the inception process.

The first step of the SQUARE process involves agreement on terms to be used during the entire development process.[3] A general list of words and their definitions are compiled using trusted resources, such as the Institute of Electrical and Electronics Engineers (IEEE) or the Software Engineering Body of Knowledge (SWEBOK). The purpose of this step is to ensure that all contributing developers understand how the terms should be used in a relevant context.

The next step of SQUARE is to identify and prioritize security goals for the software development project. Various contributors to a particular project may have differing opinions on the security aspect of software, but this step allows these individuals to reach a formal agreement on security goals. Security goals should be clearly defined, keeping the best interests of the organization.[3] To be effective, security goals must also consider the size and scope of the organization. The team of developers is responsible for providing feedback on the security goals and their prioritization.

The third step is to develop and collect artifacts. Artifacts include the basic network design diagrams, use or misuse case scenarios,[3] or attack trees. In many cases, however, such documents are not available. Therefore, the development team should stress the importance of keeping artifacts on hand. Artifacts are collected to ensure that all diagrams are complete and organized. Incomplete or poorly organized artifacts may lead to future issues during the later phases of software development.

The fourth step is to perform risk assessments with the consideration of system vulnerabilities. A risk assessment should be facilitated and completed by a risk expert, one that is most likely to be found outside of the organization and the team developers.[3] The assessment results should be analyzed with the identification of threat possibilities and their potential consequences. Following the assessment results, risks should be categorized according to the size and scope of the attack, speed of containment, and the extent of damage.[3] These results should be documented for future reference.

The fifth step is to select a suitable elicitation technique that is appropriate for the project and its sponsoring organization. Elicitation is the process of gathering essential information. Within the context of SQUARE, information related to security must be gathered. Several elicitation techniques are available for use, including use/misuse cases, structured/unstructured interviews, critical discourse analysis,[3] or the Accelerated Requirements Method (ARM).[3] While the specific technique used varies by the organization and its project, ARM has consistently proven to successfully elicit security requirements.[3] The chosen elicitation technique will be applied in the next step.

The sixth step is to apply the selected elicitation method to decide which security requirements will be necessary for the remainder of the software development. Elicitation must have clear and concise guidelines for security rather than providing security requirements themselves. These requirements should be quantitative. The purpose of elicitation is not to explain *how* security requirements should be implemented, but rather *what* should be required to make security effective. As the security requirements are developed, they should be documented for future reference.

SQUARE's seventh step is to take the requirements gathered in the previous step and categorize them.[3] The team of developers should create a primary set of categories based upon the needs of the software that will be developed in the future. Some categories that should be considered include systems level, software level, and architectural constraint. Although security elicitations should not be related to architecture, any architectural constraints would reveal that the security requirements are in relation with the modification of the system architecture.[3] Architectural

constraints attempt to explain what developers must do to reach effective security requirements. However, such constraints should be adjusted as part of elicitation.

Step eight of SQUARE analyzes the categorization of requirements and prioritizes these security goals.[3] Categorization and prioritization give software developers and stakeholders the opportunity to choose which security goals to implement and when to implement them. At this time, stakeholders have the choice to discuss the elimination of the security requirements that may appear to be entirely feasible or necessary. Some existing prioritization techniques include Triage, Win-Win, and the Analytical Hierarchy Process (AHP).[3]

The ninth and final step of SQUARE is to inspect requirements and to modify or remove any requirements that are ambiguous, contain inconsistencies, or unknowingly assume certain information.[3] Inspection should be a structured process that assesses the reasoning behind each proposed security requirement.

SREP

Another approach that could be used during the inception phase is the Software Requirements Engineering Process (SREP).[4] SREP is very similar to the SQUARE methodology outlined in the previous section, except that the steps are not chronologically outlined and the process incorporates some insights from the Common Criteria (CC).[5] CC is an objective set of security standards that are meant to enhance the overall quality of computer security and ensure that organizations are responsibly handling security.[4] The CC elements considered during the inception phase are composition, lifecycle support, and vulnerability assessment.

Approximately half of the organization's first-order requirements should be defined[4] during the inception phase. First-order requirements define the primary goals that the organization wishes to accomplish through a specific project. With acknowledgment to CC's vulnerability assessment guidelines, in SREP, a risk outline assesses and prioritizes risks. The CC composition requirements evaluate current risk controls and the effectiveness of countermeasures that are already in place. Ineffective countermeasures are reevaluated to determine how they may be improved. It may be necessary to adapt current protection profiles to meet security requirements.[4] The CC lifecycle support goals critically analyze which security measures are sufficient during the SREP process in the inception phase.

Elaboration Phase Support Mechanisms

In the elaboration phase, there are also many established ways to build security into the software design. During this phase, decisions on architectures and low-level design strategies are made. For the architectural-level security design decision-making, security tactics[6] and architectural patterns[7] are available, while design patterns[8] are available to help with low-level security design decision-making. In addition, it is beneficial for developers to be exposed to fundamental security design principles.[9] There exist various catalogs[9] of security principles, tactics, architectural patterns, and design patterns. As of this writing, the naming conventions used to refer to these are not clearly defined. Security principles, tactics, architectural patterns, and design patterns are all often referred to as simply security patterns.

Architectural analysis for security

It is often the case that software has already been built without considering security too much. In this case, an architectural analysis can be conducted to identify places in the architecture of the software to best apply security principles, tactics, architectural patterns, and design patterns.

There are many architectural analysis methodologies, but only very few of them focus on security. For example, Halkidis et al.[10] propose a methodology to conduct architectural risk analyses of software systems. One of the potential advantages of analyzing a software architecture for security is an opportunity to review whether the architecture addresses security concerns identified by stakeholders during the inception phase.

The architectural risk analysis methodology proposed by Halkidis et al.[10] does not take advantage of this stakeholder security requirements–driven opportunity but focuses more on the qualitative analysis of risks stemming from not using proven but generic security solutions. The security solutions advocated by Halkidis et al.[10] are designed to thwart well-known security threats identified in a threat model such as STRIDE[11] developed by Microsoft. Therefore, stakeholder involvement in their approach is minimal, and the possibility of adoption by practitioners is low due to the many technical intricacies to be overcome when trying to use the approach. Less technical and basic information such as security principles, tactics, architectural patterns, and design patterns could be more effective during the elaboration phase.

Secure Design Principles. One of the most fundamental ways of ensuring that stakeholder security concerns are addressed in the very early stage of design is to go over a basic checklist on security with the stakeholders because original stakeholder security requirements could be too simple to be truly helpful due to their lack of knowledge in security. The original stakeholder security requirements could be as simple as "we want the system to be secure," and may require more elaboration during the elaboration phase. Therefore, one of the tool developers could use to elicit security requirements more effectively is a list of published secure design principles. Saltzer and Schroeder[9] published such a list. One of the items in their list is "separation of privileges," which means that it is more secure to spread privileges over multiple users or

software components to enable them to trigger or perform critical software functions. Software developers can use this type of list as a conversation starter when attempting to elicit security requirements from stakeholders. When no security requirement is provided by stakeholders, the list can be an alternative for the stakeholder-driven security requirements.

Security Tactics. Security tactics[6] are self-contained security design solutions meant to be composed into an architecture to strengthen its security. Security tactics cannot exist by themselves and reinforce an architecture with respect to security. Each security tactic addresses a specific aspect of security. For example, "authenticate actors" is a security tactic that deals with limiting access to actors, which falls, in turn, under the security tactic category of "resisting attacks."

Architectural Patterns. Patterns refer to well-known solutions to recurring problems. Once combined into an architecture, security tactics can be manifested as architectural patterns for security. There are a number of widely known architectural patterns for security which developers can consider using.[7] One of such security architectural patterns is "single access point," which limits the number of access points to one so that it is easier to control access to resources to be protected in a software system.

Low-level Design Patterns for Security. Although their scope is not as comprehensive as that of architectural patterns, low-level design patterns are especially useful to frontline developers who do not have control over the overall architecture of the system but would still like to contribute to securing the system they are developing. On the other hand, software architects can enforce a certain set of these low-level design patterns for security across an entire software system, effectively making them architectural patterns. For example, a pattern such as parameterization used to prevent SQL injection attacks is a low-level design pattern, but if the same pattern is consistently used in an entire software application and strictly enforced by the architect, the parameterization pattern can be regarded as an architectural pattern for the particular software system.

Construction Phase Support Mechanisms

Compilers

A compiler is a program used by programmers. Compiling is necessary for successful code execution because the code is translated from a high-level computing language, such as C or C++, into low-level binary code comprised of 0s and 1s, to be understood by a computer. During the compilation process, the compiler will also optimize the code in attempts to allow the program to run effectively. Unfortunately, during this process, the program could incorrectly detect erroneous code, known as optimization-unstable code. In doing so, the compiler will usually eliminate any code deemed to be unstable, leaving the source code incomplete. As a result, the code may potentially run ineffectively and is subject to errors, such as buffer overflows (which occur when a program attempts to store more information than intended). This unpredictable code behavior is also known as undefined behavior.[12] Any code subject to undefined behavior could misbehave or lose complete functionality under such conditions.

Undefined behaviors can be prevented by the use of a model that statically identifies unstable code. STACK[13] is a static code checker designed by MIT to identify unstable code within the programs written in C and C++ programming languages. This process is completed prior to compiling to warn developers about true errors and prevent the mistake of false error detection. STACK has successfully detected coding errors in several systems that utilize C and C++, including Mozilla, Python, and Linux Kernel.[13]

There are several situations that could induce optimization-unstable code, and it is important to understand the underlying causes for its occurrence. STACK's methods assist in preventing future issues and vulnerabilities by showing developers which pieces of code are actually read during compilation. In knowing this, software developers save a significant amount of time and effort because the developed code is more concise.

One of the major causes of optimization-unstable code is that while the programmers rely upon the compilers to create an optimization-stable code, at the same time, the compilers rely upon the programmers to write this code. While the compilers automatically ignore any undefined behavior, at the same time, some programmers fail to recognize the errors present in their code. Because of this, the essential pieces of code are ignored by the compiler, resulting in an incomplete application.[13] This final product may contain several bugs or vulnerabilities that are difficult to identify and that may affect functionality. In many instances, some optimization-unstable code could be erroneously labeled as a bug. The programming languages, C and C++ tend to identify more cases of optimization-unstable code than the programming language, Java, for example. In some cases, this becomes an issue because optimization-unstable code could be detected with one language, but not with another. If the same programs use different languages and are intended to function identically, erroneous detection of optimization-unstable code may present some inconsistencies between applications. As a result, the issue of security becomes exploited.

Division by zero is occasionally used to determine whether an application will be executable. In C programming, division by zero is an undefined behavior. Therefore,

the compiler assumes that a nonzero number will always be used for division purposes.[12] However, under this assumption and in the case that zero is used as the divisor, the compiler determines that the entire line of code is unreadable and removes it from the program. This destroys the original protection of determining executable code and demonstrates the potential for security issues in the application.

In the C programming language, the use of a left or right shift of the n-bit by n or more bits is undefined behavior.[12] Depending on the compiler, the bits may be truncated by five to seven bits, not zero. Compilers assume that the largest shift will be $n-1$, resulting in a non-zero shift and unexpected behavior. Under this case, the compiler removes this portion of the code because it is seen as redundant.

Several measures may be taken to prevent compilers from making optimizations that compromise security. In many cases, the level of optimization may be adjusted, which could better identify code that is truly unstable. Additionally, these types of compilers can recognize unstable code at the various levels. However, several compilers do not remove optimizations to be used for security purposes, such as the division by zero code. Because of this, software developers opt for static code checkers like STACK.[12]

STACK functions by mimicking a compiler through two phases.[13] During the first phase, STACK operates under the assumption that no undefined behavior will result from the program. During the second phase, STACK operates under the assumption that the program will invoke undefined behavior. Following both phases, the results are compared, and any differences between the results reveal unstable code.[13] Thus far, these results have consistently produced a low false-warning rate. STACK serves to not only bring awareness to coding errors, but to also highlight the importance of fixing these errors to promote a secure coding environment. As a result, modifications will be made so that the compilers can better address coding issues, properly identify undefined behavior, and provide necessary warnings for developers.

Testing

Software security testing is a process to improve the quality of software by identifying software vulnerabilities that might be exploited by attackers and lead to security violations, and by validating the effectiveness of the security measures. Although secure coding practices can be applied to greatly reduce the chance of introducing vulnerabilities, it is hard for developers to directly follow best secure coding practices since they do not understand security well and mainly focus on meeting functional requirements in software.[14] Therefore, security testing became a critical step in producing secure software. Professional testing teams typically conducted security tests during the test phase of the SDLC. Several techniques[15,16] have been developed for security testing, including code reviews and penetration tests.

Code review is a security task that typically scans the source code of the application to identify software flaws. When doing a code review for complex software, we need to prioritize code review segments. Some heuristics can be used to determine the priority:[17] Code that runs by default, anonymously accessible code, and code that handles sensitive data. These priorities can help us reduce code review time. Code review generally proceeds with three phases. The first phase is code analysis. By running many static analysis tools, we can get some warnings and errors. This information is useful to determine where we have to focus at first and to guess which code snippets may have vulnerabilities. The second phase is finding vulnerability patterns. Vulnerability patterns are not declared exactly. But, in some cases, we can draw a graph that is to determine code vulnerabilities. For example, in the case of finding buffer-overflow vulnerabilities, a code snippet of copying buffers (e.g., str* family and mem* family) has to be checked. The third phase is exploring the risky code in detail. This phase checks logical errors and error handling. It can find unintended control and data flows, but this is difficult to automate. In general, manual auditing is very time-consuming, and human code auditors must know what security risks might exist before they extensively examine the code. Therefore, the use of automated tools is a preferred approach.[18] An automated code review is a cheaper and easier way to detect a broad range of security defects without requiring the operator of the tool to have the same level of security expertise as that of a human auditor, but it also faces an important challenge. Unlike the manual approach, code analysis tools often cannot definitely prove that attacks are possible. To overcome this weakness, the starting point of the review should construct mappings between insecure code and the potential security risks associated with them and explain the potential impact of the identified software bugs. We believe that a reasonable mapping can be established by analyzing common attack patterns and the related software bugs from known vulnerability databases (e.g., OWASP, CERT, US-CERT, NIST National Vulnerability Database, CVE, and CWE). We also need to provide a framework to describe how insecure code affects security risks and to gather feedback from developers.

Penetration test is a proactive and authorized attempt to evaluate the security of a target system by attempting to exploit various possible system vulnerabilities, including OS, service and application flaws, improper configurations, and even risky end-user behaviors. Such assessments are also useful in validating the efficacy of defensive mechanisms, as well as end-users' adherence to security policies. To do a penetration test, a threat model such as tampering or disclosing information, denial of service,

and deleting data should be established. According to the threat model, a test plan is built, and test cases are executed. However, there are several significant limitations[19] in conducting penetration tests: 1) Since penetration tests almost always identify problems too late at the end of the development cycle,[20] it is very expensive to fix security problems even when they are found. Surely, these problems could be fixed without incurring huge costs at the design and construction phases. 2) Furthermore, there are no established standard processes and tools. Software penetration tests depend on many factors such as testers' skills, knowledge, and experience. 3) In practice, under severe constraints of both time and budget, penetration tests can only cover a small attack surface rather than all possible security risks in a system. An attack surface quantifies vulnerable locations where attackers can access or modify data or software. To mitigate these problems, the most intuitive approach is to consider penetration tests at the early stages of the SDLC so that any security defects can be fixed with less cost.[19] Similar to unit testing, penetration testing can also be performed by developers to verify the security of each module in isolation. For doing this, we need to develop automated penetration analysis tools for generating test cases from artifacts at each stage of development.

Static Source Code Analysis Tools (SCATs)

Individual developers can test their software for security bugs as they develop the software. SCATs are helpful for this purpose, especially because of their ability to acknowledge subtle, or obvious, coding errors linked to security problems. SCATs save software developers valuable time because such tools were created especially for automatically notifying developers of coding errors before they become significant security issues.[21] Currently, numerous SCATs are available for developers, including PScan, FlawFinder, Rough Auditing Tool for Security (RATS), Secure Programming Lint (SPLINT), Extended Static Checking for Java (ESC/Java), and Model-checking Programs for Security Properties (MOPS).[21] Some of these tools are open source, meaning that they are created and updated on a volunteer basis, rely upon donations to upkeep, and are available free of charge. As seen in Table 1, each SCAT tool analyzes code written in a specific language, offers a variety of unique features, and detects coding errors in terms of security.

Veracode is a cloud-based SCAT that analyzes software code at the application layer.[22] Veracode uses a policy-based approach and is continuously updated to handle the increasing number of security vulnerabilities. This analysis tool refers to security standards, such as the OWASP Top 10 for web-based applications and the CWE/SANS Top 25 for non-web-based applications. Veracode is compatible with all major programming languages for web and mobile applications, including Java, C, C++, Objective C for iOS, and Java for Android.[22] Veracode is one of the few analysis tools that offer binary static analysis to allow developers to test software without needing the source code. Analysis results provide models of the data and control paths[22] of the application. These models are critically investigated for vulnerabilities in design. Software developers are also provided tools that prioritize software vulnerabilities.

Transition Phase Support Mechanisms

Vulnerability tracking

Vulnerabilities are weaknesses found in software. They are usually discovered once the software is transitioned into production. For example, penetration testers can expose

Table 1 SCAT software analysis.

SCAT name	Language(s)	Features	Issues
PScan	C	-Simple, easy to use	-Does not describe errors -Search scope is too narrow for business/complex applications
FlawFinder	C	-Finds more complex errors -Thorough, complete scan	-Only scans C code
RATS	C, C++, Perl, PHP, Python	-Flexible: Tests several coding languages -User may add XML reporting features	-Does not give as complete a scan -Possible false positives in results
SPLINT	C	-Lightweight, finds coding errors	-Does not provide a thorough security scan
ESC/Java	Java	-Finds coding issues at Compile, bugs at Runtime -Traces If-Then-Else issues	-Does not analyze security flaws specifically
MOPS	C	-Finds coding issues in a more sensible manner	-Still in early stages of development -All tests administered "by hand" by individuals -Java Runtime Environment must run while MOPS is in use

Source: Adapted from La.[21]

software vulnerabilities by launching simulated attacks while users can also reveal vulnerabilities by simply using the software. In addition, hackers or crackers reveal vulnerabilities by actually attacking the software through trials and errors.

Once the vulnerabilities are discovered, it is important to fix them since they can lead to successful future exploitations. The first step toward removing the vulnerabilities is to keep track of them. Due to the limitations in resources, it is not always feasible to fix all the known vulnerabilities, which is why prioritizing the vulnerabilities is critical. Vulnerability tracking can be done through a mechanism as simple as the use of a spreadsheet. However, there are also more sophisticated tools that attempt to automate many mundane tasks associated with tracking the vulnerabilities.

As part of an enhanced vulnerability management system, organizations employ automated scanning tools. These tools combine risk prioritization, identification, and remediation into effective, self-sustaining applications. KPMG, an IT auditing firm, offers an automated vulnerability tracking system that details real-time network information and vulnerabilities.[23] This approach is scalable to the scope of the organization, providing a personalized vulnerability management system. Symantec DeepSight is another valuable automated vulnerability tracking tool that blocks threats before they affect the organization.[24] With the use of automated vulnerability management tools, organizational productivity is increased because these tools provide meaningful risk management services with few remediation efforts required by the developers themselves.

Another approach to supplement company-wide vulnerability tracking is to incorporate a SCAT into the development process of coding. More details on SCATs are provided in the Static SCATs section.

One of the most common forms of manual vulnerability tracking is vulnerability scan reports produced by vulnerability scanning tools. By reviewing multiple scan reports, developers can monitor the appearance of new vulnerabilities and the disappearance of existing vulnerabilities because of remediation efforts.

Vulnerability scanning tools

There are mainly three major types of vulnerability scanning tools. The first type is a comprehensive suite of tools that examine not only software vulnerabilities, but also system vulnerabilities in general, including network security vulnerabilities. The other type is a scanning tool that analyzes the source code statically and detects coding errors leading to vulnerabilities. The third type is a tool that treats an application as a black box and tries various attacks against applications while they are running. We will discuss each type of these tools in more detail in the following subsections.

Vulnerability Assessment Tools. Vulnerability assessment and management are essential aspects of software security. New vulnerabilities become available on a daily basis, posing an increasing threat on software code. It is estimated that for every one thousand lines of code, approximately five to twenty bugs are present.[25] Once all software standards, requirements, and potential vulnerabilities are considered, the process of human vulnerability assessment becomes cumbersome and time-consuming for software developers. However, developers depend on vulnerability assessment tools for guidance in locating defects within the code at hand. Vulnerability assessment tools are also used to target weaknesses within a computer network. These processes are simplified with the use of automatic scanning mechanisms that consider the size and scope of the software, the context in which the software is applied, and the susceptibility of the software to have bugs in the code. Additionally, these vulnerability management tools have the ability to automate and enforce organization-wide policies during coding development and debugging. In addition to code analysis, the most useful vulnerability assessment tools should scan everything that connects to the organization's network.[25] This includes, but is not limited to, the operating system, web servers, SMTP/POP servers, FTP servers, firewalls, databases, e-Commerce servers, LDAP servers, load balancing servers, switches and hubs, and wireless access points. When administered appropriately, vulnerability assessment tools prove to be a feasible solution for software developers.

Commercial scanners are often distributed on the basis of an annual subscription agreement between the organization and the company that distributes the assessment tool. Furthermore, because these tools cost a considerable amount of money, they are updated frequently, and technical support is more readily available in comparison with open source software. There are commercial scanners available for nearly every platform, including Microsoft, Mac OS X, and Linux. Cloud-based scanners are also emerging as a viable platform for these vulnerability assessment tools. Some of the most common features of these tools include vulnerability tracking with detailed reports, vulnerability assessment and management, and threat remediation planning based upon scan reports and assessments. More comprehensive scanners offer features that may include patch management, threat prioritization, and asset organization. Some of the most successful scanners available today include GFI LanGuard, QualysGuard, Nexpose, CORE Impact, Secunia VIM, Nessus, and IBM AppScan. They are well known for their speed, ease-of-use, and thoroughness. ~~The figure below~~ outlines the features of each tool.

Source Code Analysis Tools (SCATs). Source Code Analysis Tools, or SCATs, are also available for use by software developers. SCATs analyze the source code statically, which means that the code is analyzed prior to

AQ1

Table 2 Commercial vulnerability assessment tools.

Vulnerability assessment tools	Platform(s)	Primary features
GFI LanGuard[26]	Windows, Mac OS, Linux	-Patch management -Vulnerability assessment -Vulnerability tracking -Network device vulnerability checks -Network and software auditing -Vulnerability reporting
QualysGuard Vulnerability Management[25]	Multi-Tenant, Private Cloud	-Scalable cloud platform -Host asset organization and categorization -Vulnerability assessment and management -Risk prioritization -Vulnerability reporting
Nexpose[27]	Windows, Mac OS, Linux	-Security assessment across physical, virtual, and cloud environments -Vulnerability tracking -Single assessment scan -Vulnerability reporting -Remediation planning -Control effectiveness measurement
CORE Impact[28]	Windows	-Penetration testing -End-user security awareness training -Efficacy testing -Password testing -Vulnerability tracking -Vulnerability reporting
Secunia VIM[29]	Windows, Mac OS, Linux	-Customizable vulnerability reporting -Vulnerability tracking -Proof-of-Concept -Asset management -Threat remediation -Vulnerability analysis -Dashboard interface -Displays available XML and RSS intelligence feeds
Nessus Enterprise[30]	Cloud	-Delegate scans to specific locations -Permission controls and limitations -Policy enforcement -Customizable scan schedule -Vulnerability reporting -Vulnerability analysis -User roles/levels -Integrated vulnerability, threat, and compliance management -Vulnerability auditing
IBM AppScan[31]	Windows, Linux, UNIX and Solaris	-Risk prioritization -Automated dynamic security testing -eXtensions framework allows for customization of open-source features -Vulnerability tracking -Detailed vulnerability reporting -Provides extensive remediation capabilities -Adaptive testing process mimics comprehensive human testing

execution. SCATs are automated to precisely locate errors by providing the line number and general location of the flaw. More details on SCATs are provided in the static SCATs section.

Dynamic Analysis Tools. Dynamic analysis tools detect coding vulnerabilities during runtime on a physical or virtual processor. The purpose of dynamic testing is to rigorously examine the software behavior at runtime and determine how the software would operate when applied in its true setting.

Veracode offers one of the most extensive Dynamic Analysis Security Testing (DAST) tools available on the market[22] as of this writing. DAST analyzes web applications during the preproduction process and release, and primarily looks for SQL injection errors, misconfiguration vulnerabilities, and cross-scripting issues. Veracode completes DAST using a three-step process: Discovery, Massively Parallel (MP) Baseline Scan, and Dynamic Deep Scan (DS). During Discovery, a general inventory of web applications is gathered. All websites that utilize outdated software are recognized and shut down prior to scanning. Those that must be scanned will undergo a baseline test during MP. If any vulnerabilities are exposed during MP, they are automatically patched almost immediately. Finally, with DS,[22] a deep scan is initiated for the applications that are behind firewalls and applications that run over the Internet. Veracode's tools will assist in providing remediation plans to ensure that proper security measures work in the event of a security issue.[22]

HP Fortify also utilizes dynamic analysis as a part of its vulnerability management suite. Fortify monitors applications during essential production stages, enforces risk prioritization, and integrates vulnerability management into the tools and processes currently enforced by the respective organization.[32] IBM Rational AppScan is a web-based dynamic analysis tool that becomes integrated into the SDLC.[31] AppScan provides real-time vulnerability management and prioritizes risk as a key component of the entire SDLC.

Dynamic analysis tools are effective because they reveal essential vulnerabilities in parts of the code that are "low traffic" or that are rarely used by the user. These areas are important because a number of vulnerabilities could potentially originate there.

Multiphase Support Mechanisms

Support mechanisms are used across multiple phases of an SDLC. This section reviews these multiphase software security support mechanisms available for developers.

Training and education

Security training and education for software developers are essential because the developers are in the best position to directly address security needs. Ideally, the developer organization is keen on ensuring that all developers are properly trained, are informed of rising risks, and understand how to address security issues.

In order to successfully train members of an organization or implement any kind of computer-aided software engineering (CASE) tool,[33] vulnerabilities are an essential aspect to consider. The reason for security training, education, and tools is to mitigate, or better yet, eliminate, as many security flaws as possible, which software development faces. There are several ways to combat the issue of vulnerabilities, and these methods may be adjusted to fit the scale of the organization and the needs possessed by the software.

The addressing of vulnerabilities begins with the organization and its associates. Information security expert Bruce Schneier[27] once said, "Security is a process, not a product. Products provide some protection, but the only way to effectively do business in an insecure world is to put processes in place that recognize the inherent insecurity in the products." Extensive planning and risk recognition in coding are oftentimes more important than using any kind of case tools. The organization must understand the importance of coding in a secure fashion. Security training and education for employees is essential because the employees from within the firm understand the security needs best. The organization is responsible for ensuring that all employees are properly trained, are informed of rising risks, and understand how to address security issues.

Standards. Currently, there are no industry standards directly concerned with secure coding practices.[34] However, there are several Information Technology (IT) standards related to security and quality assurance as they pertain to software. Some IT standards that express general concern in security include, but are not limited to, the ISO/IEC WD 15443, ISO/IEC 15408, ISO/IEC 17799, and ISO 27001. These standards, along with their titles and purposes, are outlined in Table 3.

Because no industry standards are currently related directly to practicing secure coding, an organization should create its own standards to be upheld by those within it. These may include a number of policies and standards, along with guidelines for implementation.[35] Policies are statements that are broad, yet specific, and their purpose is to clearly address *what* must be accomplished.[36] Implementation guidelines[35] are established to specify *how* the policies should be applied. It is the responsibility of the management to oversee all activities of the employees to ensure that employees are truly following the policies established by the company.[35]

Education. Unfortunately, in several instances, contemporary computer science education has not placed a great enough emphasis on the importance of coding in a secure manner.[37] Education focuses primarily on teaching

Table 3 Current information technology standards related to software security.

Standard	Title	Purpose
ISO/IEC WD 15443-1:2012	Information Technology- Security Techniques	-Concepts to understand IT security assurance
ISO/IEC 15408-1:2009	Evaluation Criteria for IT Security ("Common Criteria")	-For IT products and systems to be used as an evaluation tool -Strive to have developers seek satisfaction in their products and tools
ISO/IEC 17799:2005	Information technology- Security Techniques- Code of practice for information security management	-Organization of security policy at corporate level and how it should be applied -asset management and protection as it relates to security
ISO 27001:2013	Information Technology- Security Technology	-Defines Information Security Management System (ISMS) to address the scope of people, processes, IT systems, and policies

Source: Adapted from Mead[5] and Graff.[34]

future software developers about basic coding principles to make the code run properly. While basic coding education provides the foundation necessary for software developers, theoretical and application aspects of coding security should be taught simultaneously with coding fundamentals.

An important model that software developers should be taught is the Confidentiality-Integrity-Availability (CIA) triad. *Confidentiality*[9] highlights the security of private information. *Integrity* is the assurance that data are not manipulated, changed, or destroyed with improper intentions by an unauthorized person.[36] Finally, *availability* refers to maintaining the accessibility of resources for the authorized users.[9] The CIA triad is simplistic in nature, but its purpose is to address primary security goals for members of an organization. This model may be applied in a number of settings, including software development.

Training. The goal of developer-friendly software security is to make secure coding more effective and easier to do. The primary element of software coding is the coding itself. Therefore, if possible, it would be advisable for software developers to replace the classic, traditional coding languages with programming languages that prioritize security. If entirely replacing a coding language may not be feasible in terms of time, money, and other similar factors, the second most appropriate way to enhance security is to establish guidelines that ensure secure coding practices.[38]

Training should include at least two basic principles that would minimize risk before and during software deployment.[39] The first step is to train developers to create software that faces a controllable amount of vulnerabilities to be handled by Computer Security Incident Response Teams (CSIRTs). The second step is to train developers to eliminate these vulnerabilities as much as possible before this software is put into action.[39] The intention of software development is to deploy successful programs that are useful and meaningful. If a certain software possesses security defects before it is deployed, it does not fit the goals that software seeks to meet. Therefore, this software would not be worth running because of the risk that the users would face when trying the software.

Many corporations choose to cut corners on security because of upfront costs in both time and money.[37] It takes time for organizations to create and develop security measures, and it costs money to put such procedures into action. Furthermore, security does not generate any immediate profit for the software developers and their employers, and the profit is needed to keep the business operating.

Businesses want to invest in actions that will promote profit, and they see profit as having a major influence on growth. However, if secure coding is overlooked by an organization, there may be only limited growth for it. Many firms fail to recognize security until there is a significant problem at hand,[37] putting privacy at stake. If any organization wants to reach its maximum development potential, it should invest in security tactics before any issues crop up.[39] Such organizational smartness reduces the risk of coding errors, as well as potential costs accrued from an attack on such issues.[39]

With effective training for secure software development, the organization can centralize security instead of outsourcing security to a firm to address security defects. This solution could be more effective because the employees within the organization understand their security needs best, and this also reduces the monetary cost of outsourcing security issues.[36]

Secure software development lifecycles (SDL)

There are a number of useful ways for the developers to apply security principles into their line of work. One proposed method of security incorporation is to implement

secure practices into the SDLC. In many cases, security is not a relevant component of the SDLC.[37] In other words, organizations often do not effectively monitor and encourage secure coding principles. This relates back to the idea that security is a waste of time and money because it does not directly generate revenue. Security vulnerabilities are a consistent setback to organizations, and the organizations themselves often fail to recognize this.[37] In a study conducted by Russell Jones and Abhinav Rastogi to analyze security concerns of the organizations using IT security consultants, the most consistent issue was that security was not built into the SDLC.[40] Further research showed that if security was indeed implemented into the SDLC, any security issues would reveal themselves before the actual system's deployment. Otherwise, security issues would never appear at all.

The SDLC process includes: inception, elaboration, construction, and transition, which in turn involves testing, deployment, and maintenance.[2] Jones and Rastogi[37] proposed that security aspects can be incorporated as a part of each step. The diagram below (Fig. 1) represents the SDLC as a part of an incremental process.

Inception. The inception phase involves discussing software plans with stakeholders to gain their perspective on the proposed software. The thoughts of the stakeholders are considered because any major actions taken by the organization will directly affect this group. Additionally, the project budget is reached, and the project members are assigned their specific duties to establish assignment goals that members can reach efficiently.

AQ2

To ensure that security is a priority during the inception phase, all project members should have received proper training related to secure coding and understand how to use the security tools that will be used during software development. The budget should allocate some spending toward security assurance. Although an exact amount may vary, depending on the size and scope of the business, it is up to project managers to guarantee that an appropriate percentage of the budget is allocated to security.

Elaboration. During the elaboration phase, a security expert analyzes the security aspects that should be addressed while the software is being developed. Those who are part of the organization must understand the aspects of security, specifically within the context of the proposed software idea.

The security expert also ensures that the organization uphold industry security standards, practices specific to the organization, and security laws and regulations.[37] It is also during this elaboration phase that a team of developers considers the most potentially vulnerable areas and the tangible and intangible assets at stake.

An effective approach to analyzing risk is to model the threats with the use of a diagram (such as a UML or data flow diagram)[41] and the identification of attack targets. The elaboration phase considers potential security attacks and addresses the weakest points within the code. Such

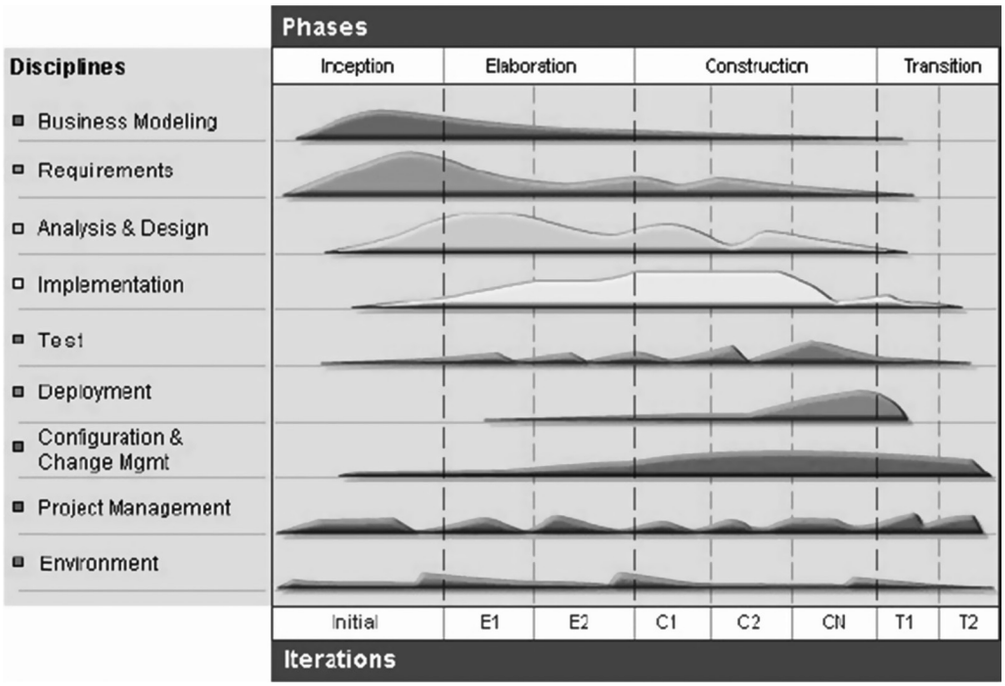

Fig. 1 IBM Rational Unified Process (RUP).
Source: From "IBM Rational Unified Process (RUP)."[2]

issues are taken care of with the provisioning of several security controls and the establishment of a trust management system.[37]

The software design done during the elaboration phase intends to reach an enterprise information security architecture (EISA)[37] while keeping it as simple as possible. The principles expressed through EISA promote IT security while maintaining a strong business focus. After the drafted design is generated, it is ideal for the project's security experts to analyze the design to ensure that proper protection measures are reached through the overall design.

Construction. Construction is a phase involving the development of all components of the code. This stage is crucial because it is during this time that vulnerabilities make their way into the code in a tangible way. One security issue that finds its way into the code during this stage is buffer overflow.[37] Buffer overflow is responsible for at least 50% of all coding issues, and the best way to prevent its occurrence is to ensure that coding developers understand the principles of security within the context of code development.

A code must be developed with the assumption that it will be deployed within an extremely aggressive environment. This is the best way of developing a code. The project managers are responsible for stressing the importance of the organization-wide policies and procedures during the entire SDLC. Once the code is developed, the software prototypes are tested through integration into current available systems.

One of the more emphasized disciplines during the construction phase of the SDLC is testing, which traditionally involves ensuring that the code executes properly in a technological setting. While testing, in the deployment environment, the primary objective is usually to test the functionality. A variety of security and risk analysis tools should be used to supplement this testing effort.

One security tool that may additionally be utilized as a part of the testing discipline is to expand the use of least privilege, or LP. LP ensures that administrative guidelines are applied in an appropriate manner. The purpose of LP principles is to confirm that users are not assigned permissions that are not required. LP is applied to prevent privilege creep, which occurs when administrative rights become diffused into the rights held by standard users.[42] The security principles, as they relate to LP, must consider the controllable units, as well as how the access policy should be implemented. The policy must specify the component's interface methods and the rules that apply to these methods.[43]

In addition to LP, a variety of security tests can be used to further enhance the code. Software developers can apply the SCATs, such as RATS, MOPS, and PScan,[21] in this discipline. The static SCATs section provides more information on these analysis tools. Third-party tests during the transition phase give outside software developers the opportunity to perform penetration tests, involving ethical hacking to test against the code's security.[37]

Transition. Once the software reaches the development phase of the SDLC, it is ready for use by humans.[37] The software development team executes the software to see that it is ready for use and makes gradual modifications during the transition process. Such modifications are known as patches, which are meant to improve the software. Each patch affects the version of software at hand, which must be tracked and managed.[37] To improve software security, the software developers should ensure understanding of the security measures built within the code. Additionally, project managers must track and manage software keys and certificates as necessary.[37]

Maintenance. The last phase of the SDLC is maintenance, which is an extension of transition. Following transition, maintenance is an ongoing phase that continues during the lifetime of the software. Project managers are responsible for continuing to track all patches, keys, and certificates, and create updates as necessary. Proper security tactics that should be considered during maintenance include monitoring security alerts and making adjustments as necessary, reviewing user control procedures and operations, and evaluating physical protection and offsite storage as they relate to usage, service, and availability.[37]

CONCLUSION

Security is a major coding aspect that is frequently overlooked by developers due to a wide range of issues. For assurance that proper security measures are being taken by software developers, those developers must be aware that security issues exist. The most realistic and practical approaches involve supporting developers by providing them with tools and processes to assist their endeavors related to enhancing software security. If things are made easier for developers to introduce security controls to their software design and implementation, the overall quality of the software, especially with respect to security, is expected to improve.

We anticipate conducting research on improving the tools and processes outlined in this entry. We believe that legislators should consider promoting the development of standards dedicated to ensuring that developers and their organizations handle software security appropriately on the basis of existing and emerging tools and processes. Additionally, we wish to see more organizations provide developer-friendly security tools and processes for their software developers while adopting stringent security standards.

ACKNOWLEDGMENTS

The STARTUP Program presented by the Pennsylvania Space Grant Consortium, an organization through the National Aeronautics Space Administration (NASA), made this research on Developer-Friendly Software Security possible. Space Grant is "devoted to the development, facilitation, and implementation of research and public outreach activities related to space." We would like to extend our thanks to the STARTUP Program for all grant funds.

REFERENCES

1. Symposium On Usable Privacy and Security. *Symposium On Usable Privacy and Security*. [Online], http://cups.cs.cmu.edu/soups/2014/ (accessed June 04, 2014).
2. IBM Rational Unified Process (RUP). February 19, 2014. [Online], http://www-01.ibm.com/software/rational/rup/ (accessed June 06, 2014).
3. Mead, N.; Hough, E.; Stehney II, T. Security Quality Requirements Engineering (SQUARE) Methodology, Technical Report CMU/SEI-2005-TR-009, November 2005.
4. Daniel Mellado, E.F.-M. A common criteria based security requirements engineering process for the development of secure information systems. Comput. Stand. Interfaces **2007**, *29*, 244–253.
5. Mead, N. The common criteria. *United States Department of Homeland Security*, August 10, 2006. [Online], https://buildsecurityin.us-cert.gov/articles/best-practices/requirements-engineering/the-common-criteria.
6. Bass, L.; Clements, P.; Kazman, R. *Software Architecture in Practice*, 3rd ed.; Addison-Wesley Professional: Boston, 2012.
7. Fernandez-Buglioni, E. *Security Patterns in Practice*, 1st ed.; Wiley: New York, 2013.
8. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed.; Addison-Wesley Professional: Boston, 1994.
9. Saltzer, J.; Schroeder, M. The protection of information in computer systems. Proc. IEEE **197AD**, *63* (9), 1278–1308.
10. Halkidis, N.T.S.T.; Tsantalis, N.; Chatzigeorgiou, A.; Stephanides, G. Architectural risk analysis of software systems based on security patterns. IEEE Trans. Dependable Secure Comput. **2008**, *5* (3), 129–142.
11. Microsoft. The STRIDE threat model, 2005. [Online], http://msdn.microsoft.com/en-us/library/ee823878(v=CS.20).aspx (accessed June 04, 2014).
12. Wang, X.; Zeldovich, N.; Kaashoek, M.F.; Solar-Lezama, A. Towards optimization-safe systems: Analyzing the impact of undefined behavior. In SOSP 13 Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, 2013; 260–275.
13. Johnson, P. How your compiler may be compromising application security. [Online], http://www.itworld.com/security/380406/how-your-compiler-may-be-compromising-application-security (accessed June 18, 2014).
14. Antunes, N.; Vieira, M. Security testing in SOAs: Techniques and tools. In *Innovative Technologies for Dependable OTS-Based Critical Systems.* Springer: Milan, 2013; 159–174.
15. Stuttard, D.; Pinto, M. *The Web Application Hacker's Handbook: Fighting and Exploring Security Flaws*, 2nd ed.; Wiley: Indianapolis, 2011.
16. Shahriar, H.; Zulkernine, M. Mitigating program security vulnerabilities: Approaches and challenges. ACM Comput. Surv. **2008**, *44* (3). AQ3
17. Howard, M. A process for performing security code reviews. IEEE Secur. Priv. **2006**, *4* (4), 74–79.
18. McGraw, G. Automated code review tools for security. IEEE Comput. **2008**, *41* (12), 108–111.
19. Arkin, B.; Stender, S.; McGraw, G. Software penetration testing. IEEE Secur. Priv. **2005**, *3* (1), 84–87.
20. Palmer, S. *Web Application Vulnerabilities: Detect, Exploit, Prevent*. Syngress: Rockland, 2007.
21. La, T. Secure software development and code analysis tools. SANS Inst. InfoSec Read. Room, 2002; 1–51.
22. Veracode Enterprise-Grade Security. *Veracode*, 2014. [Online], http://www.veracode.com/products/enterprise-grade-security (accessed March 19, 2016).
23. KPMG. *KPMG Vulnerability Management System*, 2011. [Online], http://www.kpmg.com/FI/fi/Ajankohtaista/Uutisia-ja-julkaisuja/Neuvontapalvelut/Documents/KPMG-Enhanced-Vulnerability-Management.pdf (accessed June 11, 2014).
24. Symantec DeepSight. *Semantec*, 2014. [Online], http://www.symantec.com/deepsight-products (accessed June 05, 2014).
25. Wolfgang Kandek. *Vulnerability Management For Dummies*, 2nd Ed.; John Wiley & Sons, Ltd.: Chichester, West Sussex, England, 2015.
26. GFI Software. GFI Languard Features. [Online], http://www.gfi.com/products-and-solutions/network-security-solutions/gfi-languard/specifications (accessed March 19, 2016).
27. Rapid7. *Nexpose*. https://www.rapid7.com/products/nexpose/ (accessed March 19, 2016).
28. Core Security. CORE Impact Pro Product Overview. http://www.coresecurity.com/system/files/attachments/Core-Impact-Pro-data-sheet.pdf (accessed March 19, 2016).
29. Flexera Software. Secunia Vulnerability Intelligence Manager. http://www.flexerasoftware.com/enterprise/products/software-vulnerability-management/vulnerability-intelligence-manager/ (accessed March 19, 2016).
30. Tenable Network Security. Nessus Enterprise. [Online], http://www.tenable.com/products/nessus/nessus-enterprise (accessed July 11, 2014).
31. IBM. *IBM Security AppScan*. [Online], http://www-03.ibm.com/software/products/en/appscan/ (accessed July 11, 2014).
32. Chess, B. HP Fortify. [Online], http://h71028.www7.hp.com/enterprise/downloads/software/Assessing_Application_Vulnerabilities_Fortify_360.pdf (accessed June 05, 2014).
33. Fuggetta, A. A classification of CASE technology. Computer **1993**, *12*, 25–38.
34. Graff, M. Secure coding: The state of the practice. Para-Prot. Serv. **2001**, *1*, 1–62.
35. Panko, R.R.; Panko, J.L. *Business Data Networks and Security*, 9th ed.; Pearson: Boston, 2012.
36. Boyle, R.; Panko, R. *Corporate Computer Security*, 3rd ed.; Pearson: Boston, 2012.
37. Jones, R.L.; Rastogi, A. Secure coding: Building security into the software development life cycle. Inf. Syst. Secur. **2004**, *13* (5), 29–39.

38. Tapp, C. Security and C language: fix it, don't nix it. Eng. Technol. **October 2009**, *4* (18), 58–59.
39. Seacord, R. CERT secure coding standards. Presented at the Secure Coding Initiative, Carnegie Mellon University, Pittsburgh, PA, June 28, 2006.
40. Jones, R.L. Rastogi, A. Secure coding: Building security into the software development life cycle. Inf. Syst. Secur. **2004**, *13* (5), 29–39..
41. Fowler, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed.; Addison-Wesley Professional: Boston, 2003.
42. BrightTALK. Uncontrolled Privilege Creep – The Danger Within. March 19, 2016.
43. Buyens, K.; Win, B.D.; Joosen, W. Identifying and resolving least privilege violations in software architectures. Int. Conf. Availab. Reliab. Secur. **2009**, *4*, 232–239.