IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 31, 2020, accepted April 9, 2020, date of publication April 21, 2020, date of current version May 8, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2989113

A Preventive Secure Software Development Model for a Software Factory: A Case Study

JOSÉ CARLOS SANCHO NÚÑEZ, ANDRÉS CARO LINDO, AND PABLO GARCÍA RODRÍGUEZ
Department of Computer and Telematics Systems Engineering, University of Extremadura, ES-10003 Caceres, Spain

Corresponding author: José Carlos Sancho Núñez (jcsanchon@unex.es)

This work was supported by the Junta de Extremadura (European Regional Development Fund), Consejería de Economía e Infraestructuras, under Project GR18138 and Project IB16089.

ABSTRACT The number of cyberattacks has greatly increased in in the last years, as well as their sophistication and impact. For this reason, new emerging software development models are demanded, which help in developing *secure by default* software. To achieve this, the analysis and comparison in depth of the current models of secure software development is especially important. In this paper, a review of the most popular secure software models is presented, and a new secure software methodology is proposed, adapted to all current environments. A practical experiment in a software development company is tested, as a case study, considering data from real software projects. The results are presented and compared in two development scenarios: a classic one with a reactive security approach, and another one, emerging and preventive, that applies security by default in all phases of the software life cycle. In the case study, the total amount of vulnerabilities is reduced by 68,42%, decreasing their criticality and the temporal impact of their resolutions. In this way, software security and quality are methodologically improved with the proposed model, proving that the new emerging approach provides a more secure software.

INDEX TERMS Commercial experiment, preventive model, secure software development, vulnerability reduction.

I. INTRODUCTION

Multitude of cyberattacks occur every day. The exposure of vulnerable systems facilitates the proliferation of computer attacks with serious consequences [1], [2], especially when attacks are driven to critical infrastructures, industrial processes, or IoT (*Internet of Things*) devices. As a consequence, security has become a major challenge for software development companies.

Classic software development models adopt a reactive approach, where security tasks are relegated to the final stages of software life cycle. Nevertheless, software development models should integrate functionality and preventive responses to security problems.

In general, development companies generate software according to functional requirements. At the testing phase of the development life cycle, the quality of the software is determined, and the security problems are identified and solved. This methodology of development involves several problems, and, besides, the estimation of effort to develop secure software has a great significance for software companies. In this regard, the National Institute of Standards and Technology (NIST) affirm that solving a vulnerability during the post-production phase is up to 30 times more expensive than solving it during the requirement gathering stage [3].

The costs are exponentially increased when vulnerability detection tasks are relegated to the testing phase. Reactive measures can help avoiding vulnerabilities, but they do not minimize the costs of its resolution, and do not properly protect systems. This is because the identification of vulnerabilities –without knowing their origin– could cause a redesign, and excessive changes in software implementation.

The consequences of software security failures have an impact on companies not only at economic levels, but on corporate reputation and even on the legal breach of the Data Protection of customers.

To tackle all these issues, this paper presents a new Preventive Secure Software Development Model, called *Viewnext-UEx* model, performing an experimental approach in an industrial environment. All data of the experiments

The associate editor coordinating the review of this manuscript and approving it for publication was Luis Javier Garcia Villalba.

correspond to real software projects of Viewnext, an IBM subsidiary company, being an interesting standpoint since few works present results based on the analysis of the real data in a real world case scenario. For this reason, the study case presented in the paper is a significant issue of this paper. The main features of the *Viewnext-UEx* model are detailed, discussing some of the strengths of the approach, such as the development of a tool for training, the process of traceability and monitoring of risks and security requirements, and the establishment of good practices for secure coding.

The main contribution of this paper can be summarize as follows: i) A new emerging Secure Software Development Model, preventive and flexible, is proposed; ii) Real software projects have been used to validate the proposal; iii) Secure software development models have been analyzed and compared in depth; and iv) Software security is methodologically improved with the application of the proposed model.

II. BACKGROUND AND RELATED WORK

Recent researches in the detection and correction of vulnerabilities include reactive security approaches. Hence, Tran *et al.* [4] propose an early vulnerabilities detection model to identify and prevent zero-day attacks. Murtaza *et al.* [5] analyze tendencies and patterns of vulnerabilities in software. In depth, Abaimov and Bianchi [6] propose and design CODDLE, a new approach for code injection detection. However, these works are focused only in reactive approaches to fix security failures.

On the contrary, new emerging models manage the security in the software development process from the initial stages. In this way, Apvrille and Pourzandi [7] reveal the need of considering security in all tasks of the software life cycle. Others researchers [8] try to reduce vulnerabilities from the initial stages of development, by minimizing malicious attacks, from the security requirements engineering process.

The estimation of costs to perform a secure software development is essential. Yang *et al.* [9] establish a model to estimate the effort of secure software development of operating systems in China. However, the evaluation of the specific causes of software failures is also complicated. Some other problems of generating secure software are presented by Sodiya *et al.* in [10]. Development of applications that meet security standards is an arduous process. Even when artificial intelligence techniques are applied, as Rehman and Saba propose [11].

According to Solinas *et al.* [12], security for software should be no longer optional, but mandatory. In this regard, most studies focused on secure development aim to introduce checks and measurements in Software Development Life Cycle (SDLC). Jones argues that security should be included in all the process of systems development life cycle [13]. On the other hand, Karim *et al.* [14] design a security extension to the SDLC model.

Other studies try to ensure the agile process. In this way, Kaur *et al.* [15] propose a spiral model with security. Othmane *et al.* [16] integrate security activities into the agile software development process.

In this paper, several frameworks that integrate security by default are studied:

- Microsoft Security Development Lifecycle (Microsoft SDL) [17]
- Agile Security Development Lifecycle [18] in its agile version
- Oracle Software Security Assurance (OSSA) [19]
- Comprehensive Lightweight Application Security Process (CLASP) of the Open Web Application Security Project (OWASP) [20]
- Team Software Process Secure (TSP-Secure) [21]
- Software Assurance Maturity Model (OpenSAMM) [22] from OWASP
- Building Security In Maturity Model Framework (BSIMM) [23]

These methodologies consider security activities that cover the whole software development process.

Fig. 1 shows similarities and differences of the models, through a Venn Euler diagram, comparing easily all the models.

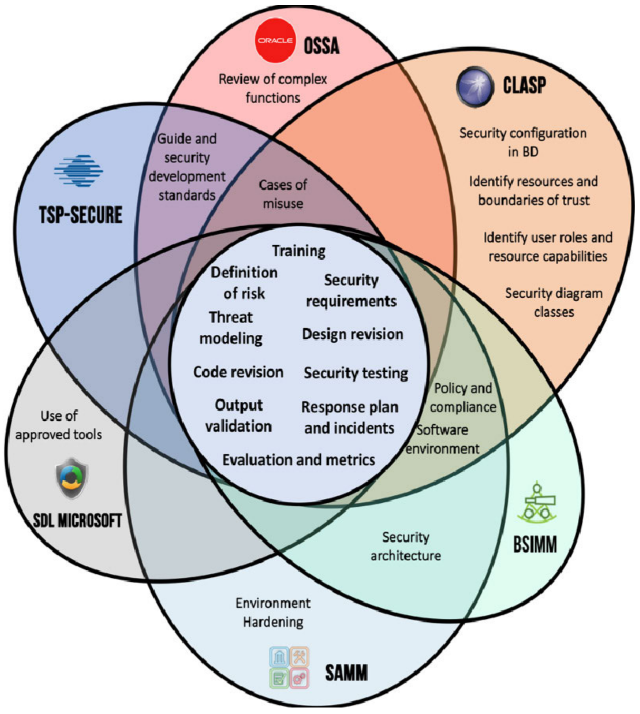


FIGURE 1. Comparison of the studied models.

The similarities should be integrated in any secure software development model, in a mandatory way. These activities are represented in the central part of the diagram.

The comparison shows there are models [19]–[21] sharing security activities, such as abuse cases. Similarly, the frameworks [19] and [22] converge on the activities related to policies and securitization of the development environment.

TABLE 1. Security activities of the models based on the life cycle.

Phase/Model	Microsoft SDL/ASDL	OSSA	CLASP	TSP-Secure	SAMM	BSIMM	Viewnext-UEx
Policies			• Identify global security policy • Identify resources and boundaries of trust • Identify user roles and resource capabilities • Specify operational environment		• Policy & Compliance	• Compliance & Policy	• Strategy and orientation • Definition of risk
Training	• Core Security Training	• Training	• Institute security awareness program	• Training	• Education & Guidance	• Training	• Training
Requirements	• Security and Privacy Risk Assessment • Establish Security Requirements	• Definition of risk • Security Requirements	• Identify attack surface • Document security-relevant requirements	• Risk analysis • Security Requirements	• Security Requirements • Secure Architecture	• Standards & Requirements • Architecture Analysis	• Requirements validation • Threat modeling
Design	• Threat Modeling • Analyze Attack Surface • Establish Design Requirements	• Threat Modeling • Design Revision • Cases of misuse	• Threat Modeling • Apply security principles to design • Security diagram classes • Cases of misuse	• Threat Modeling • Security Design • Cases of misuse	• Threat Assessment • Design Review	• Attack Models • Security Features & Design	• Design revision
Implementation	• Use Approved Tools • Deprecate Unsafe Functions	• Guide and safe development standards • Review of complex functions • Code Revision	• Security configuration in BD • Integrate security analysis into source management process	• Guide and security development standards • Code Revision	• Code Review	• Code Review	• Development revision
Testing	• Dynamic Analysis • Fuzz testing	• Security Testing	• Security Testing	• Security Testing • Fuzz testing	• Security Testing	• Penetration Testing	• Security testing
Pre-Release	• Final Security Review • Release Archive	• Security release checklists	• Verify security attributes of resources				• Output validation
Post-Release	• Execute Incident Response Plan	• Vulnerability Correction	• Manage security issue disclosure process	• Vulnerability Management	• Vulnerability Management • Environment Hardening • Operational Enablement	• Configuration Management & Vulnerability Management • Software Environment	• Security observatory • Vulnerability repository • Response plan and incidents
Metrics	• Quality Gates and Bug Bars	• Product security acquisition checklists	• Monitor security metrics	• Security metrics	• Strategy & Metrics	• Strategy & Metrics	• Evaluations and metrics • Project status
Industrial use	X	X	X	X	X	X	X

Few studies analyze in depth the methodologies of *secure-by-default* development [24], [25]. Grégoire *et al.* [26] compare the similarities of Microsoft SDL [17] and CLASP-OWASP [20] in a theoretical way. In the same way, De Win *et al.* [27] compare Microsoft SDL [17], CLASP-OWASP [20], and Touchpoints [28] by grouping the similarities of the processes according to the phase of the traditional life cycle in which they are performed. However, these studies do not analyze known secure development models. In this way, Microsoft SDL Agile, TSP-Secure, OpenSAMM and BSIMM frameworks are not considered in relevant studies.

In addition, none of the papers proposes new security activities, nor the creation of new models adapted to current needs. All of them simply shows the similarities drawn from their studies.

Previous works are only based in classifying and/or showing similarities. In [29] main models of secure software development ([17]–[23]) are studied, including both agile and traditional methodologies. As a starting point, a comparative of all aforementioned works is done, based on the most recent versions of each framework. Table 1 shows security activities from these models based on their life cycle. The *Viewnext-UEx* model is also included in this comparison. As can be seen, security activities are included (or not) and implemented in the models in different ways.

III. THE VIEWNEXT-UEX EMERGING MODEL

In this scenario, is essential to change the software development process. New paradigms are required, where the final software product was developed as a combination of functionality and security, presenting preventive responses that anticipate vulnerabilities.

In this way, Hamid and Weber propose in [30] a model driven engineering (MDE) methodological approach focus on patterns to support the development of secure software systems. Diaz *et al.* [31] use DevSecOps techniques to study the versioned configuration of a cybersecurity monitoring infrastructure.

This paper presents the *Viewnext-UEx* model, an emerging approach based on a preventive security perspective. This proposal produces software as functional as that one obtained by means of classic and reactive models, and more secure and effective for the productivity of a software factory.

The *Viewnext-UEx* model classifies activities in development areas, similar to business tasks, following the organization of SAMM and BSIMM models. Microsoft SDL and its agile ASDL version, organize security procedures in a different way, according to their execution in the traditional software life cycle.

Fig. 2 shows the *Viewnext-UEx* emerging model, which is organized in four areas of development and fourteen security activities.

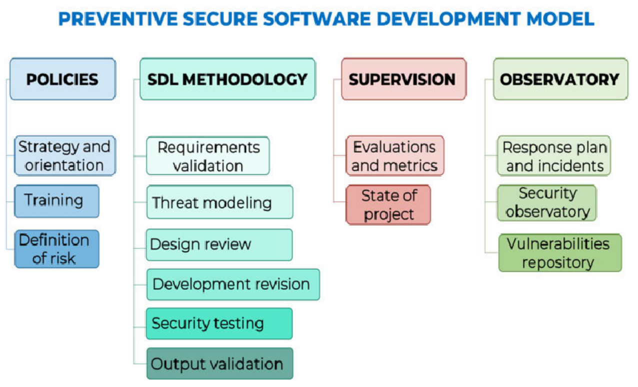


FIGURE 2. Viewnext-UEx preventive secure software development model.

The **four development areas** are: *Policies*, *Secure Development Methodology*, *Supervision*, and *Observatory*.

The *Policies* area creates and unifies the security strategy to obtain secure software. It focuses on defining the global guidelines and security objectives for software projects in specific sectors. The activities in this area require the active participation of all the groups involved in the process of software construction.

The *SDL Methodology* area is specifically oriented to develop secure software by default.

The *Supervision* area controls software security indicators and performs a final security evaluation of the delivered software.

The *Observatory* area performs a continuous surveillance to find unknown vulnerabilities. New research lines (R&D&I) could be proposed to study and discover unknown cyberattack techniques.

In relation to the **fourteen security activities**, Table 2 shows them and the corresponding development areas.

Ten of the security activities are directly integrated from the common tasks identified in the central part of Fig. 1. Another activity (*Strategy and orientation*), although it is not included in all the models analyzed, it is considered in the proposed approach, since it is essential for the application of the model. The other three activities, identified by (*) in Table 2, are new and proposed in the *Viewnext-UEx* model. All these security activities are systematically organized and planned, as indicated in [32].

TABLE 2. Security activities and their development areas and phases of the life cycle.

Development areas	Security Activity	Phase
Policies	Strategy and orientation	All life cycle
	Training	All life cycle
	Definition of risk	Requirements
SDL Methodology	Requirements validation	Requirements
	Threat modeling	Requirements Design
	Design review	Design
	Development revision	Implementation
	Security testing	Implementation Testing
	Output validation	Pre-Release
Supervision	Evaluations and metrics	All life cycle
	(*) State of the project	All life cycle
Observatory	Response plan and incidents	Post-Release
	(*) Security observatory	All life cycle
	(*) Vulnerabilities repository	All life cycle

A. POLICIES

Three practices are involved in the *Policies* area:

Strategy and orientation, a transversal practice, establishes a unified strategic plan to ensure software security. The *Orientation* covers the ten most exploited vulnerabilities [33]: Injection, Broken Authentication, Sensitive Data Exposure, XXE, Broken Access Control, Security Misconfiguration, XSS, etc. As a result, the security objectives of each project are defined and measured.

Training is divided into training in secure design, risk model, and secure coding. This activity is intended for personnel involved in the entire software life cycle. More details can be found in the SECURITY TRAINING section.

Definition of risk obtains a list of security risks related to business indicators. Business risks are different in a software project for banking, for the electricity sector, for commercial sale.

B. SDL METHODOLOGY

In relation to *SDL Methodology* area, six practices are considered:

Requirements validation obtains a list of security requirements classified based on business functionality, according to the risks known in the previous activity.

Threat modeling focuses on the design of possible threats to detect the risk of global development security, reducing the surface of attacks. The entry points of the application and the assets to be protected are studied.

Design review evaluates the design and architecture of the software based on OWASP Application Security Verification Standard [34] in order to detect security-related problems. This activity is performed before starting the development, avoiding potential costs of solving security problems.

Development revision searches for basic vulnerabilities at the code level through static analysis. A list of vulnerabilities classified according to type and criticality is obtained.

Security testing is performed by a team of experts who are not part of the development team. A mixed methodology is applied, by using automated tools and manual tests based on the *OWASP Top 10* vulnerability standards [33]. Again, a list of vulnerabilities classified according to type and criticality is obtained.

Output validation checks the status of the delivered code, validating the security objectives established with the client. A definitive version of the software is obtained without errors or vulnerabilities.

C. SUPERVISION

In the *Supervision* area, two activities are considered:

Evaluations and metrics, a transversal task throughout the development of each software project, focuses on the security and the cost of implementing the development phases.

State of the project is the first new security practice proposed in the Viewnext-UEx model. The security management of several software projects is a complex task. Mainly due to the parallel progress of the different phases of each project. This new activity is proposed to avoid the loss of current security perspective of the project. The objective of this activity is to verify the compliance of the security guidelines. These guidelines are established in other activity of the model. This procedure assess the project in relation to security. This will allow adapting the resources to solve critical security situations. In addition, this activity improves the resolution of incidents that imply compliance with the software quality standards of the Capability Maturity Model Integration (CMMI).

D. OBSERVATORY

In the *Observatory* area three practices are included:

Response plan and incidents, a reactive activity, defines a planned action policy to identify, evaluate and resolve an incident, event or vulnerability. The objective is the resolution of incidents in an effective way.

Security observatory is the second new activity proposed in the Viewnext-UEx model. The main objective of this activity is avoiding software insecurity as little time as possible. That minimizes exposure time and risk factors. Early detection provides time to find solutions or security patches. Secure developments of today may not provide the secure software of tomorrow. Therefore, searching for new unknown vulnerabilities that emerge every day has become an essential task. Information from reputable sources in the field of computer security, where vulnerabilities and recent attack techniques are published, becomes very useful. This allows the automation of the work to generate secure software and its validation. Likewise, this positively affects the reputation of the development teams and the trust of the clients.

Vulnerabilities repository, the third new security practice proposed in the Viewnext-UEx model, is included to progress and improve from experience. Although some TPS-Secure, SAMM and BSIMM models list and manage vulnerabilities, they could be improved. This activity transforms reactive measures into preventive ones, in order to adopt these improvements in the initial phases of software development. In this way, it is possible to learn from the security failures of advanced phases. Therefore, failures and errors are included in the knowledge base to train the development team. All the gathered information is used to mitigate future errors. This activity should be considered as a dynamic practice within the life cycle of secure software, allowing a capable resolution of vulnerabilities.

IV. STRENGTHS AND WEAKNESSES OF THE VIEWNEXT-UEX MODEL

Although classic models ([17]–[23]) have proven their validity, they present some deficiencies, mainly caused by the reactive standpoint, where the objective is to fix security problems instead of preventing them. These are some of the reasons why these models are becoming outdated. Thus, the new activities included in the *Viewnext-UEx* model allow avoiding vulnerabilities, performing an empirical feedback, and monitoring the security state of the software during the development process.

As a result, considering the four development areas and their fourteen security activities, the *Viewnext-UEx* model is performed as an integrated, preventive and flexible approach, allowing feedback to improve the performance. Table 3 summarize the main features of this model.

TABLE 3. Main features of the emerging Viewnext-UEx model.

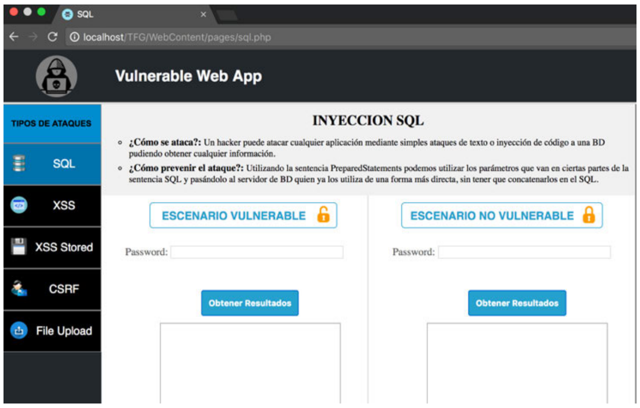
Features	Description
Integrated	• Correctly planned activities • Systematic Execution • Usual tasks performed with security policies
Preventive	• Protected life cycle phases • Traceability among security risks and security requirements • Good code practices.
Flexible	• Customizable to any methodology (agile or traditional) • Flexible tasks
Feedback	• Initial phases feedback with global learning • Avoid error reproduction • Known vulnerabilities prevention • Continuous security improvement

Two remarkable features differentiate the proposed model from others. The first one is the training of developers and auditors in security matters. The second one is the process of traceability to identify security risks, to obtain security requirements or records, and to define good coding practices. Both features are implemented in a preventive approach.

A. SECURITY TRAINING

A customized environment was built for specialized training in software security [35]. This allows the reproduction of the most exploited vulnerabilities, and also offers a double educational and learning vision to auditors and developers. The tool provides the user with skills to avoid the most common vulnerabilities, their attack/defense vectors and the exploitation flow. The final goal is the prevention of these vulnerabilities, as well as the secure codification. A differentiating feature with respect to other security learning tools such as [36]–[38]

is the possibility of simultaneously considering two specific approaches (see Fig.3): one vulnerable and the other one protected.



INYECCION SQL

- **¿Cómo se ataca?:** Un hacker puede atacar cualquier aplicación mediante simples ataques de texto o inyección de código a una BD pudiendo obtener cualquier información.
- **¿Cómo prevenir el ataque?:** Utilizando la sentencia PreparedStatements podemos utilizar los parámetros que van en ciertas partes de la sentencia SQL y pasándolo al servidor de BD quien ya los utiliza de una forma más directa, sin tener que concatenarlos en el SQL.

FIGURE 3. Interface of the training security tool.

The vulnerable approach allows accessing to the reserved information when the application is attacked. It lets viewing the vulnerable code.

The protected approach avoids the attacks. The scenario is implemented securely. The environment prevents both attacks and access to protected information, allowing auditors and developers to view the secure source code, to learn secure implementation techniques.

B. TRACEABILITY IN THE SDL METHODOLOGY

The systematic inclusion of security practices in the development life cycle requires a process of traceability. The Viewnext-UEx model prevents design and implementation of software from being only oriented to functionality [39]. By means of automated questionnaires, intrinsic aspects of security are extracted from the functional requirements. Siiskonen *et al.* propose in [40] similar ways to obtain security user stories in a generic way. Thus, the possible security risks are identified (definition of risks) and transformed into requirements or security stories (requirements validation). And threats related to the obtained security requirements are modeled (threat modeling). This finally allows the production of a white book of good practices, to perform secure coding.

After the traceability process, activities are checked and reviewed: design revision, development revision and security testing. The last two ones use automated tools to measure the security and quality of the code.

C. WEAKNESSES AND THREAT TO VALIDITY

One real case study is evaluated in this paper. Being a real case, the comparison of several projects is not practicable. For a software development company, the economic cost of developing a project for months is certainly high. However, the validation of the model in a real case study is feasible and affordable. Perhaps this is the reason why few works present experiments based on real projects. As future works, new experiments could be proposed.

Some of the threats to validity are related to human resources, due to their differences in abilities, skills, knowledge and experience, in different environments. Another threat could be related to the methodology, although the model adapts to any type of possible scenario as indicated in Table 3 in the ''Flexible'' feature.

V. COMMERCIAL EXPERIMENT

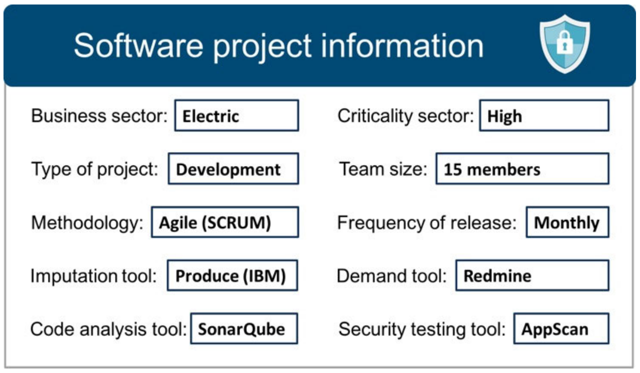
In this section, the real context of application and the evaluation methodology of the new security model is described. The main differences at security and productivity in development software at two scenarios are presented.

A. DESCRIPTION OF THE REAL CONTEXT

The case study described was achieved in Viewnext, an Information Technology Services company. This software factory is currently formed by a team of more than 4,500 professionals and is specialized in software development. The company is decentralized and distributed in several offices and technological innovation centers. The development centers are located in Spain and Portugal. The company is divided into practices, which provide remote development and maintenance services from any of the innovation centers. The ''ADM Desktop/Web practice'' performs all the activities in the SDL Methodology area, except the security testing, which are accomplished by the ''Quality and Testing practice''.

The experimental project was developed within the electric industry sector. The risks of cyberattacks turn the electricity sector into a high criticality sector. The agile methodology and the frequency of deliveries to the client determined the planning of security activities within the life cycle. The security status of the project was verified by means of several tools.

The commercial experiment presented in this paper analyzed the development of two modules (M1 and M2) of the same software project. Two scenarios were considered in this experiment: classic for the development of module M1 and emerging for module M2. Fig.4 shows the main information about this project.



Software project information

Business sector:	Electric	Criticality sector:	High
Type of project:	Development	Team size:	15 members
Methodology:	Agile (SCRUM)	Frequency of release:	Monthly
Imputation tool:	Produce (IBM)	Demand tool:	Redmine
Code analysis tool:	SonarQube	Security testing tool:	AppScan

FIGURE 4. Description of a software project.

Table 4 shows some indicators of the two modules (M1 and M2), for the software project developed (Fig.4). The same team developed both modules, with low security

TABLE 4. **Specific features of both scenarios.**

Features	Classic (M1)	Emerging (M2)
Software project	Same	Same
Team	15 members (same)	15 members (same)
Security skill	Low	High
Methodology	Agile (Scrum)	Agile (Scrum)
Approach	Reactive	Preventive
Phases apply security	Security testing (only)	All software lifecycle

skills in the early stages of development, and high level at the later ones.

Fig. 5 shows the implementation phases of the Viewnext-UEx model, starting with a classic scenario where a software module, M1, was developed (**phase 1**) based on a typical life cycle (agile, cascade…). Security skills of the development team was low, and no security tasks were included in the development of M1. The hours dedicated to development were computed according to the phases of the life cycle.

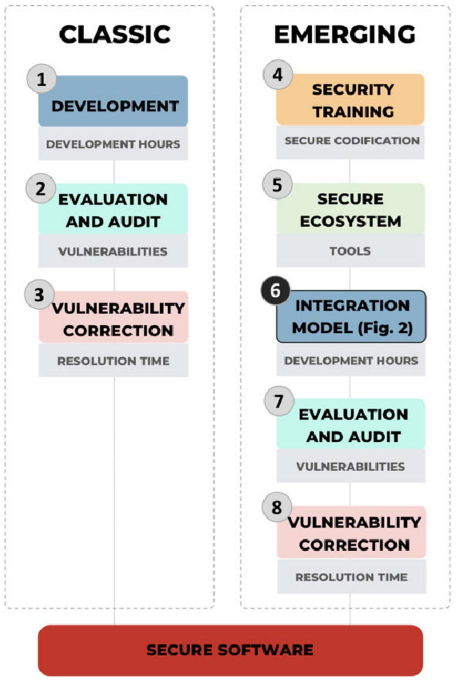


FIGURE 5. **Phases of the experiment that differentiate classic from emerging scenario.**

Afterward, the software was evaluated (**phase 2**) by means of reactive tasks, such as code analysis and security audits. The "Quality and Testing practice" of Viewnext performed the security audit process, which means the development team did not achieve these tasks (phase 2). They developed code but the team not evaluated it.

In **phase 3**, the development team analyzed and fixed the vulnerabilities found, at the end of the development process, when the module was finished, reaching this correction in a reactive way (phase 3) by means of testing tasks.

After finishing the module M1 (classic scenario), the same development team, developed a second module (M2) with the same architectural framework, functional and security complexity, but in an emerging scenario.

In contrast to classic scenario, in the preventive scenario, the vulnerability detection is anticipated. The emerging scenario found software vulnerabilities in advance during the development process, without delaying the security tasks to the latest stages. This scenario followed a preventive approach, being suitable to implement the proposed model according to the procedure presented in [41].

First, the development team was trained and formed in secure codification (**phase 4**). The security skills of the team were increased to a high level. Specialized training was designed to instruct designers and developers, and another specific training to prepare software and business analysts. The training security application shown in Fig. 3 was one of the tools used in this training process.

In **phase 5**, a secure ecosystem of tools were prepared. Check and control tools of software security are essential, and the selection of these tools are important too. In addition, it is necessary to know fundamental information of the software project to perform an effective implementation. Table 5 shows the contact questionnaire used to obtain the most relevant information of the project. Some questions have a customized answer, depending on the software project, and others must be selected. The options corresponding to M2 are highlighted in bold type.

The set of tools (secure ecosystem) to improve software security can be configured by knowing the information of each project. Table 6 shows the tools integrated into the proposed new model [42].

In the **phase 6**, the *Viewnext-UEx* model was used in the development stage of the emerging scenario, as Fig. 5 indicates, instead of using standard life cycle as agile, cascade.… Security activities of this new model were integrated from the earliest phases of the software life cycle in the software development processes.

After the software development process, a security assessment was performed through an audit (**phase 7**). The evaluation and audit tasks were similar to the implemented in the classic scenario (phase 2).

Finally, in **phase 8**, the vulnerabilities found were fixed. Again, the correction of vulnerabilities was performed as in the classic scenario (phase 3).

Fig. 6 shows the architecture of the experimental application for the emerging scenario.

Eventually, both scenarios were similar and then could be compared. The main difference between the reactive and preventive scenarios was related to the software development methodology (phase 1 and phase 6, as is specified in the Fig. 5).

B. SECURITY AND PRODUCTIVITY INDICATORS

The effectiveness of the proposed emerging model was estimated through several indicators related to software security and productivity performance.

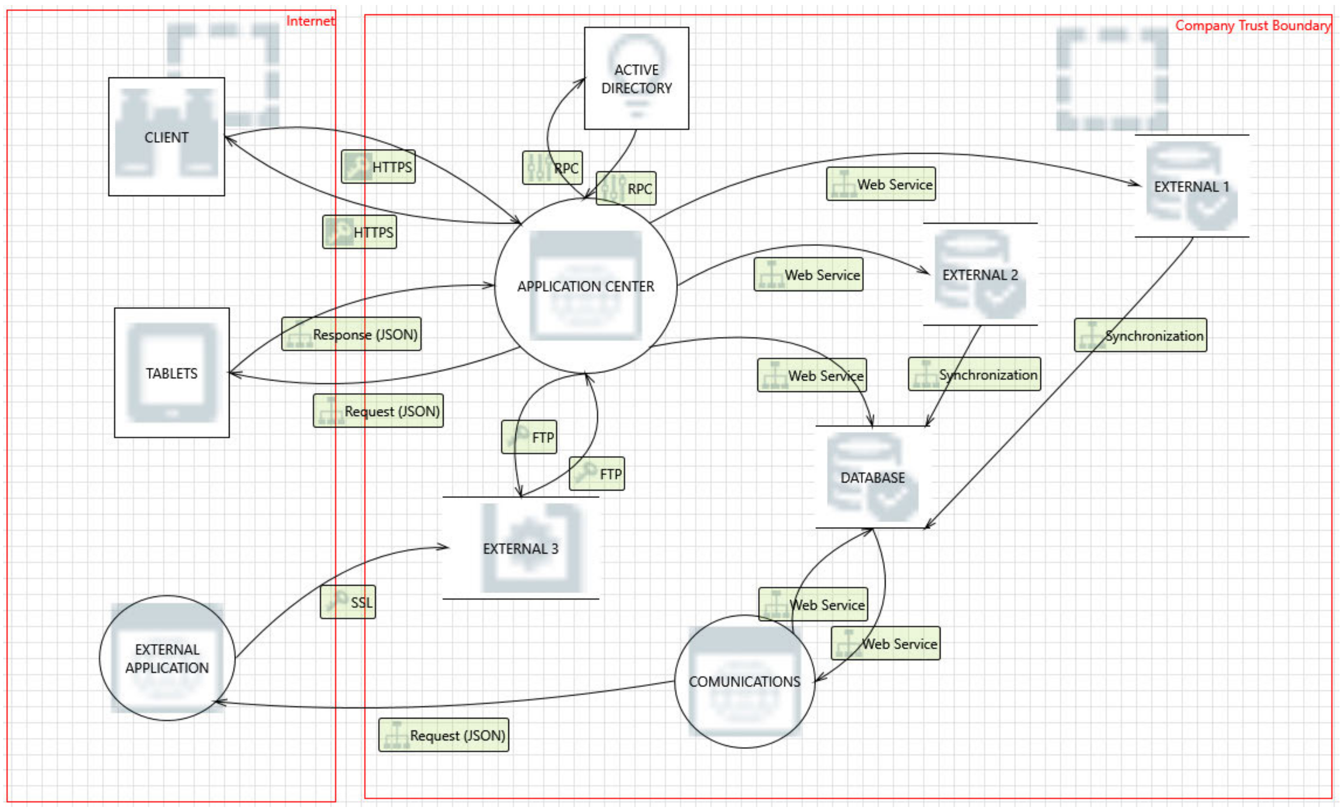


FIGURE 6. Application architecture of the emerging scenario.

Besides, to compare the productivity or effectiveness of each scenario, the development time was also estimated. It could help to decide between a classic model and the proposed *Viewnext-UEx* emerging model.

Several performance indicators were defined, in relation to the cost and security of development, as table 7 shows. The indicators were used to compare the development of module M1 (classic) in relation to module M2 (emerging).

One of the indicators could not be computed (*hours dedicated to security, per SDLC phase*), as M1 did not follow the *Viewnext-UEx* secure software development model. That indicator was the main difference between both models. Following a reactive approach for module M1, computing the time involved in security activities in development process was not possible.

The development cost was computed in hours per task, whereas the security level was determined by the number, type and criticality of the detected vulnerabilities. The number of vulnerabilities indicated the total amount of found vulnerabilities in each of the modules. The type of vulnerabilities differentiated those related to application architecture (web server, application server, database, frameworks, custom code, etc.) from the ones related to software development (injections, XSS, broken authentication, sensitive data exposure, etc.). The Common Vulnerability Score System standard [43] was used to determine the criticality. This standard classifies vulnerabilities into five categories: critical, high, medium, low and none.

One of the indicators could not be computed (*hours dedicated to security, per SDLC phase*), as M1 did not follow the *Viewnext-UEx* secure software development model. That indicator was the main difference between both models. Following a reactive approach for module M1, computing the time involved in security activities in development process was not possible.

VI. RESULTS

In the classic scenario, 2,228 hours were dedicated to the module M1. Fig. 7 show the time per phase in percentage, for this module. The hours were computed according to the phases of the life cycle.

Considering the percentage of time for the development of M1 (the four initial stages in Fig. 7), the partial percentage time added up to 88.8%, being 11.2% the percentage of time to evaluate (2.7%) and correct (8.5%) vulnerabilities.

On the other hand, in the emerging scenario, 794 hours were dedicated to develop the module M2. The time per phase in percentage for the module M2 is shown in Fig. 8.

In this preventive scenario, 95.6% of the time was devoted to the M2 development stages (the four initial ones in Fig. 8). Apparently, only 4.4% of the total time was dedicated to evaluate (3.1%) and correct (1.3%) vulnerabilities.

TABLE 5. Emerging model implementation form.

Question	Answer
Application user type	**Internal management**
Development type	Maintenance **New Development** Product
Client collaboration level	**High** Medium Low
Architecture type	Standalone **Web** Mobile SOA Microservices Mixed (web/mobile)
Access type	Internet **Intranet** Extranet
Sensible data management (medical, personal, economic, business, etc.)	**Yes** No
Bank data management (cards, accounts…)	**Yes** No
Applicable Legal regulations	**Yes** No
Applicable Sector Regulations	**Yes** No
Type Life Cycle Used	**Agile-Scrum** Agile-Kanban Cascade Other
Delivery Frequency (sprints, releases)	**Monthly**
Continuous Integration.	**Yes** No
Development Environment	**Local** VDI
Requirement Specification	**Yes** No
Tool for Dedicated Time Control	**Produce**
Tool for Demand Management	**Rational Team Concert**

TABLE 6. Selected toolset to create the secure ecosystem.

Objective	Phase	Tools
Continuous Integration	All life cycle	Jenkins, IBM Cloud, Gitlab
Threat Modeling	Requirements Design	Microsoft, IriusRisk
Static Application Security Testing	Implementation	SonarQube, PMD, Kiuwan, Checkmarx, FindBugs
Security Under Test	Implementation	Junit, xUnit.net, SoapUI
Security Integration Test	Implementation	Arquillian, SoapUI
Dynamic Application Security Testing	Testing	AppScan, BurpSuite, OWASP ZAP, Hdiv, modSecurity

The vulnerabilities identified are shown in Table 8 (for the classic scenario) and Table 9 (for the emerging scenario).

In this point, it is important to detail the time dedicated to security activities in the emerging scenario, distributed in the phases of software life cycle (Table 10), by following the *Viewnext-UEx* model.

VII. DISCUSSION

This section presents the discussion of applying the proposed evaluation methodology in both scenarios.

TABLE 7. Cost and development security indicators.

Key Performance Indicator	Classic (M1)	Emerging (M2)
Global development hours	✓	✓
Development hours, per SDLC phase	✓	✓
Hours dedicated to security, per SDLC phase	✗	✓
Hours dedicated to vulnerability resolution	✓	✓
Total number of vulnerabilities	✓	✓
Development vulnerabilities	✓	✓
Architecture vulnerabilities	✓	✓
Criticality: None	✓	✓
Criticality: Low	✓	✓
Criticality: Medium	✓	✓
Criticality: High	✓	✓
Criticality: Critical	✓	✓

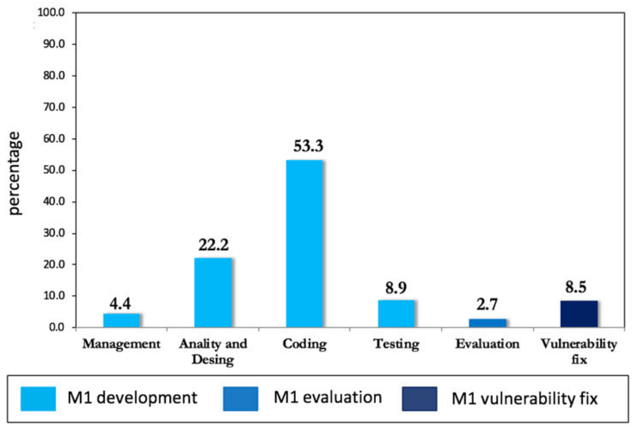


FIGURE 7. Time per phase of M1 module, corresponding to classic scenario.

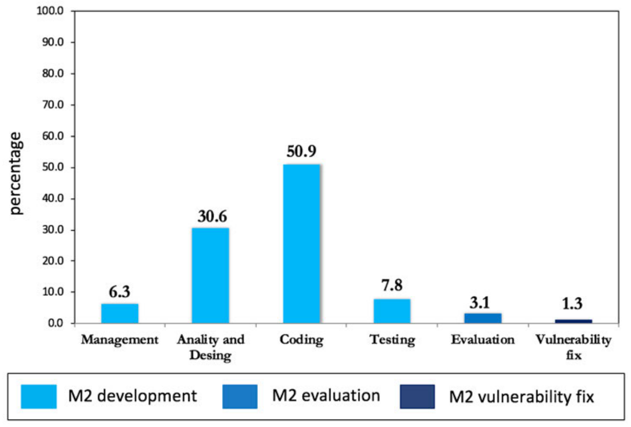


FIGURE 8. Time per phase of module M2, corresponding to scenario emerging.

Security by default was not considered in the classic scenario. As can be seen in Fig. 7, 88.8% of the time was dedicated to the development of M1, and 11.2% to evaluate and fix vulnerabilities.

TABLE 8. Vulnerabilities identified in the classic scenario.

Vulnerability	Criticality	Type
Stored Cross Site Scripting	Critical	Development
Reflected Cross Site Scripting	High	Development
Accessible Database	Low	Development
Autocomplete Attribute not enabled	Low	Development
POST change requests for GET	High	Architecture
POST directive with invalidated parameters	High	Development
Functional Privilege Escalation	High	Development
URL Privilege Escalation	High	Development
Session ID Vulnerable	High	Architecture
Logout incorrectly implemented	Low	Architecture
Sensible application information and use of vulnerable components	Medium	Architecture
Sensitive information in metadata	Low	Architecture
Sensitive information in the source code	Medium	Architecture
Default server page	Low	Architecture
HTTP application instead of HTTPS	High	Architecture
Phishing through framework	High	Architecture
Links injection	High	Architecture
TCP response timestamp	Low	Architecture
Concurrent connections from different IPs	Medium	Architecture

TABLE 9. Vulnerabilities identified in the emerging scenario.

Vulnerability	Criticality	Type
Accessible Database	Low	Development
SSL weak certificates	High	Architecture
Improperly enabled services and ports	High	Architecture
TCP response timestamp	Low	Architecture
Concurrent connections from different IPs	Medium	Architecture
SMB signature (Server Message Block) Not required	High	Architecture

For the emerging scenario, although the greatest development effort was made in the analysis/design, and coding tasks (Fig. 8), however the security tasks were distributed almost equally among analysis/design, coding and evaluation. *Security by default* approach was considered for M2, which implied not only time to evaluate and correct vulnerabilities (4.4% as mentioned in the results section) but also to *prevent* them. Thus, computing the percentage of time for development (88.3%) and for security tasks (11.7%) as Table 10 shows, these percentages are really very similar to those obtained for the module M1.

TABLE 10. Time dedicated to security compared to time dedicated to development in hours.

Applied phases	Development	Security
Management	49	1
Analysis/design	215	28
Coding	383	21
Testing	54	8
Evaluation	-	25
Vulnerability fix	-	10
TOTAL	701	93
%	88.3	11.7

In relation to indicators of rating security, nineteen vulnerabilities were found in the classic scenario: twelve related to architecture and seven related to development issues (as Table 8 shows). Regarding the criticality of the vulnerabilities, ten out of nineteen were classified as critical or high, while three were of medium criticality and six were of low criticality.

In contrast, much less vulnerabilities were detected in the emerging scenario. Particularly, six vulnerabilities (Table 9), being five of them of architecture and the other one of development. In addition, three out of six were of high criticality, one of medium and two of low criticality.

Tables 8 and 9 are summarize in Fig. 9, which presents a graphical comparative of both experimental scenarios, showing the number of vulnerabilities found and their level of criticality.

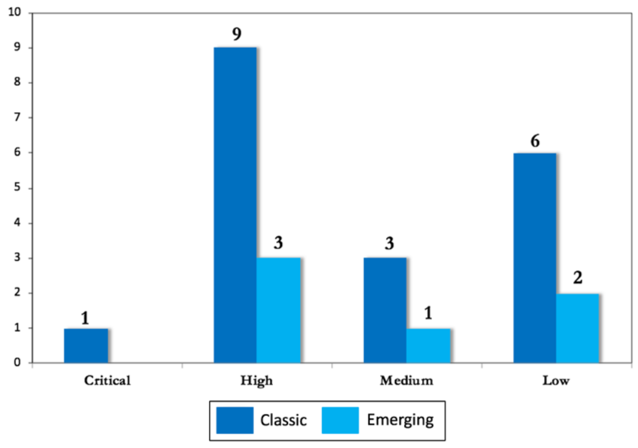


FIGURE 9. Comparative of vulnerabilities and their criticality identified in each scenario.

A considerable reduction in the number of vulnerabilities can be seen in Fig. 9 (nineteen at the classic scenario versus six at the emerging). This implies a reduction of 68.42% in the quantity of vulnerabilities that affected software development.

The study about the impact of those vulnerabilities is also essential. In the classic scenario, the 73.68% of the vulnerabilities found were classified as medium/high/critical. This is a really worrying percentage. On the contrary, in the emerging scenario, although the 50% of the vulnerabilities were of high criticality, none of them was critical.

The Viewnext-UEx preventive model not only reduced the number of vulnerabilities, but also their criticality and impact. Consequently, this new proposal developed software by default more secure than classic models.

The vulnerabilities found for both scenarios are detailed in table 11, according to their type (architecture or development) and criticality (based on the Common Vulnerability Scoring System standard).

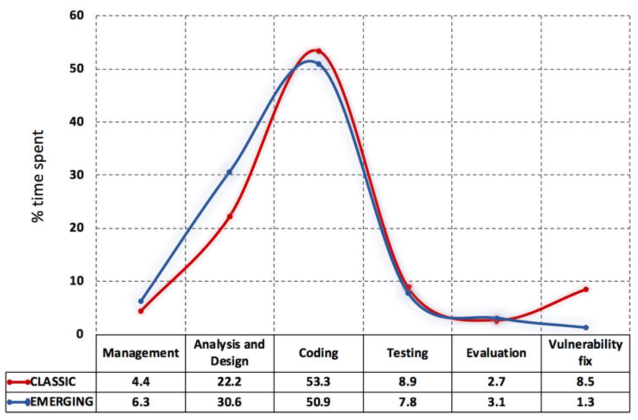
Table 11 shows the vulnerabilities found in the experiment, level of criticality and type of vulnerability (architecture or development). As Fig 9, this table reveals the great number

TABLE 11. Comparative of vulnerabilities identified in each scenario. Criticality classified by CVSS version 3.

Category	Vulnerability	Criticality	Type	Classic	Emerging
Input Validations	Stored Cross Site Scripting	Critical [9-10]	Development	X	
	Reflected Cross Site Scripting	High [7-8.9]	Development	X	
	Accessible Database	Low [0.1-3.9]	Development	X	X
	Autocomplete Attribute not enabled	Low [0.1-3.9]	Development	X	
	POST change requests for GET	High [7-8.9]	Architecture	X	
	POST directive with invalidated parameters	High [7-8.9]	Development	X	
Authorization	Functional Privilege Escalation	High [7-8.9]	Development	X	
	URL Privilege Escalation	High [7-8.9]	Development	X	
Sessions Management	Session ID Vulnerable	High [7-8.9]	Architecture	X	
	Logout incorrectly implemented	Low [4.0-6.9]	Architecture	X	
Error and log handling	Sensible application information and use of vulnerable components	Medium [4.0-6.9]	Architecture	X	
	Sensitive information in metadata	Low [0.1-3.9]	Architecture	X	
	Sensitive information in the source code	Medium [4.0-6.9]	Architecture	X	
Configuration Management	Default server page	Low [0.1-3.9]	Architecture	X	
	HTTP application instead of HTTPS	High [7-8.9]	Architecture	X	
	Phishing through framework	High [7-8.9]	Architecture	X	
	Links injection	High [7-8.9]	Architecture	X	
	SSL weak certificates	High [7-8.9]	Architecture		X
	Improperly enabled services and ports	High [7-8.9]	Architecture		X
	TCP response timestamp	Low [0.1-3.9]	Architecture	X	X
	Concurrent connections from different IPs	Medium [4.0-6.9]	Architecture	X	X
	SMB signature (Server Message Block) Not required	High [7-8.9]	Architecture		X

of vulnerabilities in the classic scenario compared to the emerging one. The preventive emerging scenario avoided most of the vulnerabilities of the classic approach. On the other hand, some of the vulnerabilities were only found in the emerging scenario, which could mean they were not detected in the classic development models.

Fig. 10 presents the percentages of time dedicated on every of the phases of software development life cycle. It exposes the emerging model dedicated more percentage time in the earliest phases, mainly because security issues were considered in these initial stages of the development.



	Management	Analysis and Design	Coding	Testing	Evaluation	Vulnerability fix
CLASSIC	4.4	22.2	53.3	8.9	2.7	8.5
EMERGING	6.3	30.6	50.9	7.8	3.1	1.3

FIGURE 10. Comparison of time spent for each phase of the development process of the classic and emerging scenarios.

All the results are presented in percentages of time mainly because the total amount of development time is different in each scenario. Time invested was recovered in the final stages of the development process.

In this regard, a qualitative fact detected by the Scrum Manager of the project was the time spent on analysis and design phases compared to the coding stage. The emerging scenario shown that greater time dedication to the initial phases of analysis and design reduced the coding time.

In relation to the vulnerability resolution phase, there is a big difference between the two scenarios (a reduction of 7.2%). The classic scenario was affected by a great impact in this phase, due to the reactive approach. However, as mentioned above, the emerging preventive scenario distributed the temporal cost related to security activities among all phases of software development. In this way, the vulnerability resolution phase had a minimal cost.

Indeed, the final stage of software development is often tense and stressful. In addition to possible delays, last-minute issues can arise, making it desirable to reduce the number of vulnerabilities that must be fixed as much as possible. Furthermore, as it has been demonstrated, the final developed software was not only functionally correct, but also more secure.

This new proposal in the field of emerging software models is the first research that presents comparative results of the application of a classic and reactive model versus an emerging and preventive one. Few studies have been found to contrast the results presented in this paper. Mainly due to the big difficulty of evaluating real software projects, which prove the novelty of this research.

In addition, direct transfer of knowledge from a research center to a company like Viewnext is another remarkable aspect. The company has adopted this new secure software development methodology in its production system, and is currently marketing it, which in some way proves the validity of the proposal.

VIII. CONCLUSION

This paper presents the Viewnext-UEx model, a new, preventive and flexible approach to develop secure software. The best-known models in secure software development have been studied and compared, identifying their best practices, and some of their deficiencies. The new model includes the better security activities of these well-known models, besides other security tasks, correcting the weaknesses of the proposed models and following a preventive approach.

The Viewnext-UEx model is tested with real data. The case study shows that the number of detected vulnerabilities is reduced by 66%. The criticality of vulnerabilities is also significantly reduced. All this produces an evident reduction in costs and times, much more prominent in the final stages of development. The security and quality of software is increased, as well as the productivity of development.

The availability of real data together with the comparison of the best-known models to identify best practices and the inclusion of specific activities suggest that the proposed model could be customized to other business environments.

ACKNOWLEDGMENT

The authors would like to thank Viewnext, an Information Technology Services company of IBM Spain Group, for the help provided with the data transfer.

REFERENCES

[1] *Cyberthreats and Tendencies Executive Summary 2018*, Nat. Cryptol. Center Comput. Emergency Team Response, Madrid, Spain, 2018.

[2] *Cyber Threats and Cyber Security 2019*, Nat. Cryptologic Center Comput. Emergency Team Response, Madrid, Spain, 2019.

[3] National Institute of Standards and Technology. (2002). *The Economic Impacts of Inadequate Infrastructure for Software Testing*. Accessed: Jun. 18, 2017. [Online]. Available: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf

[4] H. Tran, E. Campos-Nanez, P. Fomin, and J. Wasek, "Cyber resilience recovery model to combat zero-day malware attacks," *Comput. Secur.*, vol. 61, pp. 19–31, Aug. 2016.

[5] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and A. B. Bener, "Mining trends and patterns of software vulnerabilities," *J. Syst. Softw.*, vol. 117, pp. 218–228, Jul. 2016.

[6] S. Abaimov and G. Bianchi, "CODDLE: Code-injection detection with deep learning," *IEEE Access*, vol. 7, pp. 128617–128627, 2019.

[7] A. Apvrille and M. Pourzandi, "Secure software development by example," *IEEE Secur. Privacy Mag.*, vol. 3, no. 4, pp. 10–17, Jul. 2005.

[8] D. Mellado, E. Fernandez-Medina, and M. Piattini, "A security requirements engineering process in practice," *IEEE Latin Amer. Trans.*, vol. 5, no. 4, pp. 211–217, Jul. 2007.

[9] Y. Yang, J. Du, and Q. Wang, "Shaping the effort of developing secure software," *Procedia Comput. Sci.*, vol. 44, pp. 609–618, Jan. 2015.

[10] A. S. Sodiya, S. A. Onashoga, and O. B. Ajayi, "Towards building secure software systems," *Issues Informing Sci. Inf. Technol.*, vol. 3, pp. 635–646, Jan. 2006.

[11] A. Rehman and T. Saba, "Evaluation of artificial intelligent techniques to secure information in enterprises," *Artif. Intell. Rev.*, vol. 42, no. 4, pp. 1029–1044, Dec. 2014.

[12] M. Solinas, L. Antonelli, and E. Fernandez, "Software secure building aspects in computer engineering," *IEEE Latin Amer. Trans.*, vol. 11, no. 1, pp. 353–358, Feb. 2013.

[13] R. L. Jones and A. Rastogi, "Secure coding: Building security into the software development life cycle," *Inf. Syst. Secur.*, vol. 13, no. 5, pp. 29–39, Nov. 2004.

[14] N. S. A. Karim, A. Albuolayan, T. Saba, and A. Rehman, "The practice of secure software development in SDLC: An investigation through existing model and a case study," *Secur. Commun. Netw.*, vol. 9, no. 18, pp. 5333–5345, Dec. 2016.

[15] P. Kaur, D. Kaur, and H. Singh, "Secure spiral: A secure software development model," *J. Softw. Eng.*, vol. 6, no. 1, pp. 10–15, Jan. 2012.

[16] L. B. Othmane, P. Angin, H. Weffers, and B. Bhargava, "Extending the agile development process to develop acceptably secure software," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 6, pp. 497–509, Nov. 2014.

[17] S. Lipner, "The trustworthy computing security development lifecycle," in *Proc. 20th Annu. Comput. Secur. Appl. Conf.*, 2004, pp. 2–13.

[18] Microsoft Corporation. (2010). *Agile Development Using Microsoft Security Development Lifecycle*. Accessed: Jun. 19, 2017. [Online]. Available: https://www.microsoft.com/en-us/SDL/Discover/sdlagile.aspx

[19] COC Redwood Shores. (2011). *Oracle Software Security Assurance*. [Online]. Available: https://www.oracle.com/support/assurance/index.html

[20] OWASP Project. *Comprehensive, Lightweight Application Security Process*. Accessed: Mar. 15, 2020. [Online]. Available: https://www.us-cert.gov/bsi/articles/best-practices/requirements-engineering/introduction-to-the-clasp-process

[21] N. Davis, P. L. Miller, W. R. Nichols, and R. C. Seacord, "TSP-secure," in *Proc. 4th Annu. TSP Symp.*, 2009, pp. 3–8. [Online]. Available: http://resources.sei.cmu.edu/asset_files/ConferencePaper/2009_021_001_298907.pdf

[22] OWASP Project. (2009). *Software Assurance Maturity Model*. [Online]. Available: http://www.opensamm.org/downloads/SAMM-1.0.pdf

[23] S. Migues, J. Steven, and M. Ware. (2019). *Building Security in Maturity Model*. [Online]. Available: https://www.bsimm.com/content/dam/bsimm/reports/bsimm10.pdf

[24] D. V. Mohino, B. Higuera, B. Higuera, and S. Montalvo, "The application of a new secure software development life cycle (S-SDLC) with agile methodologies," *Electronics*, vol. 8, no. 11, p. 1218, 2019.

[25] L. Williams, G. McGraw, and S. Migues, "Engineering security vulnerability prevention, detection, and response," *IEEE Softw.*, vol. 35, no. 5, pp. 76–80, Sep. 2018.

[26] J. Grégoire, K. Buyens, B. D. Win, R. Scandariato, and W. Joosen, "On the secure software development process: CLASP and SDL compared," in *Proc. 3rd Int. Workshop Softw. Eng. Secure Syst. (SESS: ICSE Workshops)*, May 2007, p. 1.

[27] B. De Win, R. Scandariato, K. Buyens, J. Grégoire, and W. Joosen, "On the secure software development process: CLASP, SDL and touchpoints compared," *Inf. Softw. Technol.*, vol. 51, no. 7, pp. 1152–1171, Jul. 2009.

[28] G. McGraw, "Software security: Building security in," in *Proc. 17th Int. Symp. Softw. Rel. Eng.*, Nov. 2006, p. 1.

[29] J. C. S. Núñez, A. C. Lindo, and P. G. Rodríguez, "Análisis de metodologías de Desarrollo de Software Seguro," in *Proc. Jornadas Nacionales Investigación Ciberseguridad (JNIC)*, 2016, pp. 42–47.

[30] B. Hamid and D. Weber, "Engineering secure systems: Models, patterns and empirical validation," *Comput. Secur.*, vol. 77, pp. 315–348, Aug. 2018.

[31] J. Diaz, J. E. Perez, M. A. Lopez-Pena, G. A. Mena, and A. Yague, "Self-service cybersecurity monitoring as enabler for DevSecOps," *IEEE Access*, vol. 7, pp. 100283–100295, 2019.

[32] J. C. S. Núñez, A. C. Lindo, P. G. Rodríguez, and Á. Quesada, "Categorización de Actividades de Seguridad en el Desarrollo de Software," in *Proc. Jornadas Ingeniería Softw. Bases de Datos*, 2016, pp. 565–568.

[33] *OWASP Top 10—The Ten Most Critical Web Application Security Risks*, OWASP, Bel Air, MD, USA, 2017.

[34] *Application Security Verification Standard (2014)*, OWASP, Bel Air, MD, USA, Oct. 2014, p. 47.

[35] J. C. S. Núñez, M. L. Castaño, A. C. Lindo, J. A. Félix, G. Rodríguez, and A. B. Gómez, "Herramienta de entrenamiento para el desarrollo de software seguro," in *Proc. Actas las 24th Jornadas Ingeniería Software Bases Datos (JISBD)*, 2019, pp. 1–4.

[36] OWASP. *OWASP WebGoat Project*. Accessed: Mar. 7, 2020. [Online]. Available: https://owasp.org/www-project-webgoat/

[37] J. Druin, "InfoSec reading room introduction to the OWASP mutillidae II Web," SANS Inst. InfoSec Reading Room, 2013. [Online]. Available: https://www.sans.org/reading-room/whitepapers/infosec/introduction-owasp-mutillidae-ii-web-pen-test-training-environment-34380

[38] *Damn Vulnerable Web Application (DVWA)*. Accessed: Mar. 10, 2020. [Online]. Available: http://www.dvwa.co.uk/

[39] J. C. S. Núñez, A. C. Lindo, L. Fondón, and J. A. F. de Sande, "Herramienta para la identificación de requisitos de seguridad en un Modelo de Desarrollo Seguro," in *Proc. Reunión Española Sobre Criptología Seguridad la Información (RECSI)*, 2018, pp. 92–95.

[40] P. Pietikäinen, J. Röning, T. Siiskonen, and V. Ylimannela, *Handbook of The Secure Agile Software Development Life Cycle*. Oulu, Finland: Univ. of Oulu, 2014.

[41] J. C. S. Núñez, A. C. Lindo, P. G. Rodríguez, and J. A. F. de Sande, "Metodología de Implantación Empresarial de un Modelo de Desarrollo de Software Seguro," in *Proc. Jornadas Nacionales investigación en Ciberseguridad (JNIC)*, 2017, pp. 128–133.

[42] J. A. F. de Sande, J. C. S. Núñez, and A. C. Lindo, "Evaluación y selección de un ecosistema de herramientas para un enfoque preventivo y continuo en modelos de desarrollo seguro de software," in *Proc. Jornadas Nacionales Investigación en Ciberseguridad (JNIC)*, 2018, pp. 87–94.

[43] *Common Vulnerability Scoring System V3.0: Specification Document*. Forum Incident Response Secur. Teams, Morrisville, NC, USA, 2015, pp. 1–21.

JOSÉ CARLOS SANCHO NÚÑEZ received the degree in computer science from the University of Extremadura, in 2015. He is currently a Substitute Professor with the Department of Computer and Telematic Systems Engineering, University of Extremadura, and a member of the Media Engineering Group (GIM). He has participated in several national congresses and workshops and has made a research stay at the Complutense University of Madrid, Spain. His research interest includes audit and security software development.

ANDRÉS CARO LINDO received the B.Sc. and M.Sc. degrees in computer science, in 1993 and 1998, respectively, and the Ph.D. degree in computer science from the University of Extremadura, Spain, in 2006. He has been an Associate Professor with the Department of Computer Science, University of Extremadura, since 1999. He is also the Lab Head of the Media Engineering Group, University of Extremadura. He has participated in several Research and Development projects. He is a coauthor of numerous research SCI journal articles. His research interests include cybersecurity, big data and machine learning, and pattern recognition.

PABLO GARCÍA RODRÍGUEZ received the Ph.D. degree in computer science, in 2000. He was the Director of the School of Technology, Cáceres, from 2017 to 2019, where engineering studies are teaching in civil, building, computing, and telecommunications. He is currently the General Director of the Digital Agenda of the Government of the Autonomous Community of Extremadura, Spain. His teaching was mainly centered on subjects of programming and information systems in computer engineering. His research interests include the Internet of Things (IoT), bigdata and pattern recognition, and image analysis.