electronics

MDPI

Article

The Application of a New Secure Software Development Life Cycle (S-SDLC) with Agile Methodologies

Juan de Vicente Mohino, Javier Bermejo Higuera, Juan Ramón Bermejo Higuera and Juan Antonio Sicilia Montalvo *

Escuela Superior de Ingeniería y Tecnología, Universidad Internacional de La Rioja, Avda. de La Paz 137, 26006 Logroño, La Rioja, Spain; juandevicentemohino@gmail.com (J.d.V.M.); javier.bermejo@unir.net (J.B.H.); juanramon.bermejo@unir.net (J.R.B.H.)

* Correspondence: juanantonio.sicilia@unir.net

Received: 17 September 2019; Accepted: 21 October 2019; Published: 24 October 2019

Abstract: The software development environment is focused on reaching functional products in the shortest period by making use of the least amount of resources possible. In this scenario, crucial elements such as software quality or software security are not considered at all, and in most cases, the high value offered to the projects is not taken into account. Nowadays, agile models are booming. They are defined by the way they achieve the interaction and integration of everyone involved in the software life cycle, the advantages of the quick reaction to change, and the implementation of artifacts or deliverables which display the level of progress reached at any time. In this context, it seems clearly necessary to define a new software development model, which prioritizes security aspects at any phase of the software life cycle and takes advantage of the benefits of the agile models. The proposed methodology shows that if security is considered from the beginning, vulnerabilities are easily detected and solved during the time planned for the project, with no extra time nor costs for the client and it increases the possibilities of reaching success in terms of not only functionality but also quality.

Keywords: software development; methodology; Secure Software Development Life Cycle (S-SDLC); agile; software security

1. Introduction

Traditionally, software development projects have been aimed just at getting functionality in time. Quality or security have never been seen as necessary nor relevant. Over time, quality has been taken into consideration, but security remains far away considering the importance of cybersecurity nowadays. In traditional software development methodologies, security is usually introduced in the final phases and, therefore, in the most complex ones for its implementation increasing the possibilities of generating additional costs, raising development times, and damaging business reputation.

Those are just some reasons which highlight the importance of implementing a proper Secure Software Development Life Cycle (S-SDLC) [1–9]. It seems imperative that the concept of security is extended and that it is understood that security needs to be present in any development.

On the other hand, it is increasingly necessary to immerse in agile methodologies to take advantage of the flexible response to functional requirement changes and reach a high degree of quality in software development projects. It is also important to emphasize that these kinds of methodologies do not usually take practices for secure software development into account. For this reason, it seems clearly

necessary to have a methodology that can determine the activities to be performed during analysis, design, implementation, testing, and deployment to ensure the development of secure software.

Introducing security in software development projects is not only important for reaching robustness, but also its importance goes beyond quantifying the benefits it has for both customers and users as well as for developers and providers of software solutions, cost reduction, and development times dedicated to solving errors, repetition of security audits, etc.

2. Background

2.1. Introduction

A Software Development Life Cycle (SDLC) is a framework that defines the process used by organizations to develop an application from its origin to the end of its life cycle [1–9]. There are many software development methodologies and generally, all of them contemplate, from a high-level point of view, the following set of activities:

- Identification of requirements
- Architecture and design
- Codification
- Testing
- Production and maintenance of the application

The development of secure and reliable software requires the adoption of a systematic process or discipline that addresses security in each of the phases of its life cycle. Two types of security activities must be integrated in the same stage: the first following secure design principles (minimum privilege, etc.) and the second including a series of security practices (specification of security requirements, cases of abuse, risk analysis, code analysis, dynamic penetration tests, etc.). This new life cycle with included security practices included is called S-SDLC. Among others, it is possible to mention some advantages of adopting an S-SDLC such as error identification or coding and design weaknesses in the early stages of development, which imply significant cost savings.

On the other hand, agile methodologies [10–14] represent a philosophy of using organizational models based on collaboration between people and shared values that seek to achieve customer satisfaction through short and fast iterations from which deliverables can be obtained through which you can have a view of the progress of developments from the first phase.

The very nature and principles defined in the Agile Manifesto [15,16] establish the main objective of achieving functional software as quickly as possible, which generally implies not considering security activities during the life cycle such as security impact analysis or verification and validation tests, in addition to not taking into account the support and supervision by an expert or security team.

The purpose of this section is to collect and present overview information about existing processes, standards, life cycle models, frameworks, and methodologies that support or could support secure software development and agile methodologies.

2.2. Secure Software Development Life Cycle (S-SDLC)

According to a Carnegie Mellon University technical note [17] relative to S-SDLC, a number of existing processes, models, and other standards identify the following four focus areas for secure software development:

- Security engineering activities
- Security assurance activities
- Security organizational and project management activities
- Security risk identification and management activities

A novel example of security engineering activities is the work of Buinevich, Izrailov, and Vladyko [8], in which they proposed a scheme of the life cycle of a vulnerability, synthesized on the basis of the analysis of possible representations of typical SW (Software) in telecommunication devices, although applicable to any type of software:

A. Representation No. 1: Main Concept
B. Representation No. 2: Conceptual Model
C. Representation No. 3: Architecture
D. Representation No. 4: Algorithms (of the Source Code)
E. Representation No. 4.1: Algorithms (of the Assembler Code)
F. Representation No. 4.2: Algorithms (of the Assembler Code) - Modified
G. Representation No. 5: Source Code (Original)
H. Representation No. 5.1: Source Code (Pseudo)
I. Representation No. 6: Assembler Code
J. Representation No. 7: Machine Code
K. Representation No. 8: Image File

It even develops a tool that allows the realization of search of vulnerabilities according to the representations previously indicated in the developed software.

Trujillo and Chávez [18] justify that the S-SDLC establishes the phases or steps that a software must follow in order to strengthen itself, complying in turn with the requirements of the end user and allowing the generation of quality software.

A S-SDLC [1] process "is either a SDLC process augmented with various security practices or activities like a security specification language, security requirements engineering process, secure design specification language, set of secure design guidelines, secure design pattern, secure coding standard, and software security assurance method (e.g., penetration testing, static analysis for security, and code reviews for security)."

It has always been customary to perform security tasks only during the testing stage. The great disadvantage of this approach is that many of the problems were discovered when the development was very advanced or finished [2]. This represents a serious problem, not only in terms of error correction but also in terms of cost. The S-SDLC model focuses on the incorporation of security in any type of SDLC, which entails a series of benefits, among which can be mentioned [19]:

- Software that is more robust and trustworthy
- Early detection of faults, coding errors, design weaknesses, security deficiencies, etc.
- Cost reduction derived from an early detection and resolution of the problems listed in the previous point
- Lower risk to the organization

Also, existing Capability Maturity Models (CMM) provide a reference model of mature practices for a specified engineering discipline. An organization can compare their practices to the model to identify potential areas for improvement [3]. The CMM provides goal-level definitions and key attributes of specific processes (software engineering, systems engineering, security engineering), but does not generally provide operational guidance for performing the work.

This work analyzed several approximations of the S-SDLC and CMM that were reported in several articles. These include McGraw's S-SDLC process [19], Trustworthy Computing Security Development Life Cycle or Microsoft Software Development Life Cycle (MS SDL) [20], Comprehensive Lightweight Application Security Process (CLASP) [21], Team Software Process for Secure Software Development (TSP-Secure) [22], Rational Unified Process Secure (RUPSec) [23], Building Security in Maturity Model (BSIMM) [24], Software Assurance Maturity Model (OPEN SAMM) [25], Appropriate and Effective Guidance for Information Security (AEGIS) [26], Secure Software Development Model

(SSDM) [27], Writing Secure Code [28], Waterfall-based software security engineering process model [29], Secure Software Development Model Cátedra Viewnext Extremadura University [30], Secure Software Development Model (SecSDM) [31], Software Development Process Model (S2D-ProM) [32], Correctness by Construction (CbyC) [33], and Security Quality Requirements Engineering (SQUARE) methodology [34].

Table 1 summarizes and compares these S-SDLC and/or CMM processes. The four main phases of the secure development life cycle have been considered: identification of requirements, design, implementation, and verification and validation.

Table 1. Comparative analysis of the Secure Software Development Life Cycle (S-SDLC) at the level of security activities proposed in each phase.

Model	Engineering Requirements	Design	Implementation	Verification
McGraw [19]	Identification of security requirements; specification of abuse cases	Risk analysis	Code review	Penetration and risk-based testing
Microsoft SDL [20]	Identification of objectives, interfaces, and security requirements; definition of output criteria	Identification of critical security components, attack surface, secure design techniques or guides, threat modeling, risk analysis, and definition of secure architectures	Follow-up of coding standards, code review, use of testing tools, and static code analysis	Code reviews and security testing
CLASP [21]	Risk analysis, threat modeling, identification of attackers and attack surface, specification of abuse cases, and mitigation measures	Follow-up of design guides, use of class annotation diagrams, threat modeling, and risk analysis	Tracking secure codification policies	Code review, use of static code analysis, and testing tools
TSP [22]	Threat modeling, abuse case specification, and risk analysis	Pattern design, status machine design, and verification	Follow-up of guidelines and standards of safe development	Fuzz testing, penetration testing, static code analysis tools, and code review
RUPSec [23]	Unambiguous definition of requirements	Abuse cases, threat definition, use cases, event flow, and threat modeling	Portability and precision in implementation	Evaluation of security aspects
BSIMM [24]	Training, policies, and metrics	Attack models, definition of secure design properties, and standards	Code review, security testing, and architectural analysis	Penetration testing, vulnerability and configuration management, and software environment
OPEN SAMM [25]	Policy, strategy and metrics, education, and guidance	Security requirements, threat assessment, and security architecture	Vulnerability management and operational assessment	Design review, security testing, and code review
AEGIS [26]	Identification of assets, risk analysis, abuse cases, and establishment of security requirements	Design activities based on identified requirements	-	-
SSDM [27]	Threat modeling and security policy specification	Follow-up of security policies	-	Penetration testing
Writing Secure Code [28]	Security questions and previous security training	Threat modeling	Security equipment review, security documentation, tool preparation, best practice and security programming guides, fault analysis, external review, fuzzing tests, minimum privileges testing, and revision flaws	Compliance and maintenance of the security level
Waterfall [29]	Definition of security controls	Solutions relating to algorithms, architecture, conceptual and design model, logic diagrams, and interfaces	Secure code, static analysis, and TDD (test driven development)	Penetration testing, dynamic analysis, error correction, monitoring, and security testing
Viewnext [30]	Definition of objectives, training, knowledge of risks	Threat modeling, secure design	Configuration analysis, secure testing	Continuous assessment, validation of compliance, and continuous learning

Table 1. *Cont.*

Model	Engineering Requirements	Design	Implementation	Verification
SecSDM [31]	Risk analysis, identification of security requirements, identification of assets, threats, degree of vulnerability, and comprehensive risk management (impact, probability, assessment, priority, and security level)	Security services and mechanisms; consolidation of security mechanisms	Security mechanisms and components, specification of security methods, functions, and testing	Validation of security mechanisms, review, and correction of errors, training, and document verification
S2D-ProM [32]	Following security standards, risk analysis, error identification, and specification of security functionalities	Definition of security patterns using modeling languages, risk analysis, design review, and model verification	Monitoring of secure coding standards; use of secure programming languages	Risk analysis, code reviews, and use of static code analysis tools
CbyC [33]	Identification of security requirements due to a formal and clearly defined specification	Incremental development	SPARK (Spade Ada Kernel)	Debugging and formal verification methods
SQUARE [34]	Security requirements prioritization			Use of static and dynamic code analysis tools
Oracle SSA (Software Security Assurance) [35]	Identification of objectives and principles	Vulnerability management	Vulnerability management	Vulnerability management

In addition, Table 2 analyzes the use of resources, artifacts, whether they have agile properties, and the degree of use in the software industry today.

Table 2. Comparative analysis of S-SDLCs at the level of resources, artifacts, agile properties, and use in the software industry.

Model	Resources	Artifacts	Agile	Use in Industry
McGraw [19]	Secure design guides	Abuse cases for obtaining test cases	No	Reported
Microsoft SDL [20]	Secure design guides	-	Yes [1]	Reported
CLASP [21]	Security design and implementation guides, vulnerability lists, and their possible mitigations	Testing based on the requirements	No	Reported
TSP [22]	-	-	No	Not reported
RUPSec [23]	Security expert	-	No	Not reported
BSIMM [24]	Security models, standards, and repeatable processes	Security domains	Yes	Reported
OPEN SAMM [25]	Security coding and measuring tools	-	No	Reported
AEGIS [26]	-	Abuse cases for obtaining test cases	No	Not reported
SSDM [27]	-	-	No	Not reported
Writing Secure Code [28]	Best practice guides and secure programming	Key elements	No	Not Reported
Waterfall [29]	Best practices	-	No	Reported
Viewnext [30]	Policies	Security levels	Yes [2]	Not reported
SecSDM [31]	ISO (International Organization for Standardization) standards, NIST (National Institute of Standards and Technology) guidelines	-	No	Not reported
S2D-ProM [32]	-	-	No	Not reported
CbyC [33]	Formal methods	-	No	Not reported
SQUARE [34]	-	-	No	Not reported
Oracle SSA [35]	Secure coding best practices	Policies	No	Reported

[1] Microsoft includes an agile version of its methodology. [2] The model proposed by Andrés Caro can be applied following agile properties or traditional methodologies.

2.3. Agile Methodologies

Agile methodologies represent a philosophy based on collaboration between people and shared values that seek to achieve customer satisfaction through short and fast iterations from which deliverables are obtained and there is a general view of the progress from early stages.

The scope established by the agile methodologies is summarized in four values and twelve principles applied in a set of practices. The four values that define the Agile Manifesto are [15,16]:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

From these four values, the twelve agile principles were declared. The Table 3 shows the principles and the implications they have regarding compliance with software security.

Table 3. Agile Manifesto principles [15,16].

Principle	Resolution	Justification
Customer satisfaction through early and continuous software delivery	Obstructive	Security tests are usually ignored, or organizations do not allocate enough resources for them
Accommodate changing requirements throughout the development process	Obstructive	This could not be obstructive if the customer is willing to spend time to evaluate the security impact for each new requirement
Frequent delivery of working software	Obstructive	This could not be obstructive if the customer gives more priority to security than to delivery
Collaboration between business stakeholders and developers throughout the project	Neutral	It could be contributory if the development team had security experts and the client team included security as one of their priorities
Support, trust, and motivate the people involved	Neutral	It could be obstructive if the team ignores security priorities.
Enable face-to-face interactions	Neutral	It is important to keep in mind that there must be experts outside the team who evaluate security
Working software is the primary measure of progress	Obstructive	If the focus is only on functionality, security will never be considered
Agile processes to support a consistent development pace	Neutral	The consistent development pace must be applied both functionally and at the security level
Attention to technical detail and design enhances agility	Contributory	It is contributory, especially when "technical excellence and good design" reflect a strong experience in the commitment to ensure software development
Simplicity	Contributory	If simplicity is a feature it will be easier to take security into consideration
Self-organizing teams encourage great architectures, requirements, and designs	Neutral	The work team must have at least the figure of a security expert that oversees security in the software development process
Regular reflections on how to become more effective	Contributory	If security is not taken into account, the exploitation of vulnerabilities will affect the progress

Some of the benefits associated to agile methodologies are [10–14]:

1. Reduction in the time necessary for installation and maintenance
2. Cost reduction
3. Visibility from the beginning
4. Improved software quality
5. Better priority management
6. Improved software maintenance
7. Better alignment between IT (Information Technology) and the business
8. Reduction in risks
9. Improved team morale
10. Increased productivity

11. Simplified development process
12. Improved engineering discipline

The Table 4 shows a comparative analysis of the different agile methodologies that currently exist and have been studied. The analysis carried out includes a section that lists the main principles that define the methodology, then the practices, methods or artifacts each one defines and, finally, security-related aspects are covered.

Table 4. Agile methodologies.

Model	Principles	Artifacts	Security
SCRUM [36–39]	Early achievements, accommodate changing requirements, reaction to change, continuous communication, small and incremental deliveries, and maximizing customer value	Sprints, meetings (sprint planning, daily planning, SCRUM review, backlog grooming, release planning); product and sprint backlog, roles (product owner, SCRUM master, development team), cards, and boards	Not covered
XP (eXtreme Programming) [38,40]	Simplicity, communication, feedback, respect, and courage	Small deliveries, continuous integration, use of standards, pair programming, test driven development (TDD), sustainable pace, full team participation and ownership, customer tests, refactoring, and simple design	Focus on testing
Kanban [39,41]	Start applying at any time, encourage incremental changes, preserve current roles and responsibilities, and leadership by the whole team	Workflow visualization, limit work in progress, policies, use of models, use of cards and boards, early problem detection, and collaboration	Security factor
Lean [39,42]	Eliminate unnecessary elements, continuous learning, as much information as possible, fast deliveries, training and motivation, integrity, and general overview	-	Focus on integrity
FDD (Feature Driven Development) [43]	Iterative, incremental, and customer-oriented development	Features, global model, list of functionalities, plan, design, and build by function	Model SFDD (secure feature driven development)
SFDD (Secure Feature Driven Development) [44]	Security in all phases, planning, risk analysis, testing, validate software security, and security role	-	Security oriented
Microsoft SDL [20]	Identification of security requirements, performance of security activities in all phases, and according to scope, culture, and security related training of team members, related vulnerabilities, use of tools for security activities, threat modeling, exception handling, and final security review (FSR)	Every-sprint requirements, bucket requirements, and one-time requirements	Security oriented
SECDEVOPS (Secure Development Operations) [45]	A dynamic and flexible model. Allows users to get involved during software development and get customers' input in a strategic way to define security requirements. Improves the security of the software considering the necessary effort	This life cycle allows the integration of developers, operators, and security managers from the beginning, thus avoiding security that falls behind at the end of the process. Recommends maintaining frequent and short development cycles, integrating security measures with minimal impact on operations. Proposes the automation of security checks	Security oriented

3. Relative Work

Generally, security is not considered in any of the phases of the software development life cycle. At best, it is covered through the definition of non-functional requirements; for this reason it is either not taken into consideration or is done at the end of the project. Security is seen as an element that increases the development and delivery times of the project, something which goes against agile principles [46–48].

There is no concrete definition of security activities, techniques, and methods that can be applied in software development projects.

The project team usually has little knowledge and experience in security. Similarly, there is a lack of security awareness.

However, some of the secure software development life cycles are starting to consider some of the agile properties:

- Microsoft SDL: Microsoft's model tries to include non-functional activities such as those related to security, creating a backlog of non-functional tasks or requirements. For them, it defines three phases in which the following activities are included [20]:
    - Every-sprint requirements
    - Bucket requirements
    - One-time requirements

In addition to these phases, the agile version of Microsoft's SDL refers to additional tasks like culture and security training by team members, considers vulnerabilities according to the scope of the software, remains up to date on secure cryptographic algorithms, uses support tools for the execution of security activities, realizes in-depth threat modeling as well as treatment of exceptions and final security review (FSR) (see Figure 1).

- BSIMM: since this S-SDLC arose from the collaboration of multiple organizations for the proposal of good practices, agile properties are addressed as part of a process of adaptation to the trend that marks the market and adaptation to the processes that are followed today in software development projects. It also introduces the concept of agile trainer to help teams adopt security practices in agile environments. Emphasis is also placed on the use of agile practices to identify metrics in early stages of development [19,24].
- Viewnext: the model proposed by [30] refers to an agile adaptation of the proposed S-SDLC. No specific details about its implementation are specified, but the AS-SDLC concept is referred to as Agile Secure Software Development Life Cycle.

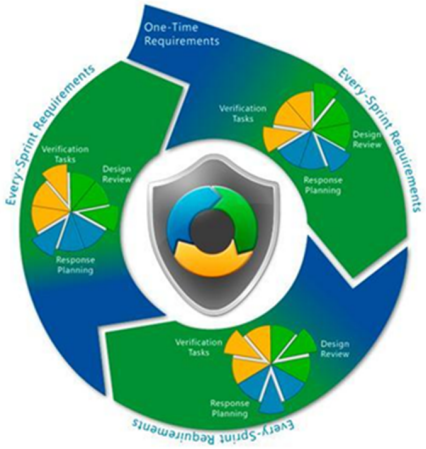

Figure 1. Agile SDL [49].

4. Definition of New S-SDLC Based on Agile Methodologies

4.1. Introduction

The proposed S-SDLC introduces security as a crucial element in software development environments and at the same time takes advantage of agile properties.

Following this line of taking advantage of agile methodologies, the design of the new S-SDLC focuses on taking the most representative elements and artifacts from them and selecting those that are most useful for software development projects. In this way, and analogously to SCRUM, the unit of time used in the development of the project is the sprints or the definition of cards for the

classification of the different activities is done through the box concept, taking Kanban as a reference. Other representative features would be pair programming of eXtreme programming or the approach based on decision-making with the most information available as Lean does.

The new S-SDLC has been developed around three pillars that stand as the foundations and the essence of the methodology and serve as the basis of the 23 principles proposed.

Agility and security represent the logical and main pillars; flexibility has been added as the ability to adapt to the changing needs of software development projects.

As for the principles, they establish the guidelines to be followed in order to take advantage of the pillar's intrinsic properties. Among all the principles, it is important to note that more than principles, they are recommendations whose purpose is to provide guidelines that will help to achieve the project's objectives. The second most relevant principle is that the implementation of the S-SDLC must provide a general and personal value to each member of the team in a way that its level of involvement in the project makes him or her a better team member.

The principles are as follows:

1. First of all, there are no principles, only recommendations.
2. Choose a model considering the needs of the project.
3. The backlog of activities identifies all the activities that will cover the requirements collected. It must be prioritized and updated according to the client's needs and the key milestones identified by the team.
4. The security requirements must be included in the backlog as an integral part of the functionalities to be covered with the development of the project.
5. It is recommended to classify security activities according to their importance and the way in which they must be repeated considering at least the following three levels: activities essential for security that must be carried out for each release, activities that must be carried out at the end of each phase or for the most important releases, and activities that can be developed in any phase of the project and only need to be done once.
6. Milestones have to be defined. They are used to make deliveries to customers on a regular basis and with incremental value. In these intervals of time or sprints the deliverables or releases are generated. The releases must provide an incremental value. Between each sprint it is recommended to hold an internal meeting in which the work done in the finished sprint is evaluated, reviewing the problems found, identifying improvements, planning the activities for the next one, and defining the scope of the next deliverable. These meetings are called sprint start meetings. It is recommended that the release generated at the end of each sprint be evaluated in three aspects: functionality, safety, and quality.
7. Periodic communications with client. Similarly, the client must be informed of the scope and content of the releases that will be developed in each sprint. It is recommended to carry out extraordinary communications with the client in those cases in which important changes are identified in the requirements or events that seriously affect the planning of the project.
8. At the end of each sprint a client meeting must be held to present the release developed in the sprint. These meetings are called release meetings.
9. It is important to have an expert in security, known as the security master. It is also recommended that the development team includes one or more roles with relevant security knowledge, known as the security gurus.
10. It is necessary to have a project manager, called the project master, who is responsible for ensuring compliance with the requirements and objectives of the project, plays the role of interlocutor with the client, manages internal and external problems that may affect the development of the project, and organize internal meetings and meetings with clients.
11. Each member of the team is considered a leader. The project and security master advises and manages in his or her respective areas and is responsible for the final decision in case of conflict,

but each and every one of the members must be considered as leaders or gurus in their areas of responsibility. Leadership should be used as a way of motivation and cohesion of team members.

12. It is recommended to use the pair programming technique in order to improve the design and coding of the software by taking advantage of the evaluation from different points of view.
13. It is recommended to continue training in security and development technologies. The promotion and sharing of knowledge among all team members is also important. It is proposed to implement a knowledge database or Wikibase that collects the technical, managerial or documentary knowledge that was generated during the project.
14. The methodology must be applicable at any time and in any phase of the project.
15. Get the most information possible before making any decision in the project. All members must provide their vision according to their role, as well as their particular vision according to their experience.
16. It is recommended to deal with the uncertainty in an agile way and as early as possible regarding any aspect of the project from its detection and making it known to all team members and the client if necessary.
17. It is recommended to apply the principle of code refactoring in order to facilitate the detection of errors and improve the quality of the software.
18. It is recommended to follow the principle of non-reinvention. In the world of software development there are usually studies and/or technical solutions for most of the problems that need to be addressed. Therefore, a periodic review of technical guides and good practices, forums, publications, etc., is also recommended.
19. Contextualize the development by identifying the most relevant attack scenarios and vulnerabilities that may affect the software.
20. Measurement should be a tool for the improvement and feedback of a project. At this point it is necessary to adopt the Deming cycle guidelines corresponding to the "Check" and "Act" phases.
21. Safety testing must be promoted. It is recommended to define and perform at least one safety test for each functional test that is defined.
22. Documentation should be promoted. The development of a clear and detailed documentation facilitates the deployment and maintenance of the software.
23. Finally, the project and participation in it must provide general and particular value to each member of the team, such as the acquisition of new technical or management skills, the accumulation of experience, the improvement of communication capabilities, etc.

Implementing ethical hacking-related activities is a good practice; although they are undoubtedly necessary, the world of software development still needs more and it is of crucial importance to perform them from the beginning to the end of the project. At the moment, when a bug or problem is detected, it is usually too late, and attackers have enough time to detect it or the modification of the application requires a complete re-codification.

There are some key processes to attend when considering security in software development projects. In addition to focusing on security from software requirements specification, gathering as much information as possible is crucial for the purpose of decision-making.

Security requirements must be included in the backlog as an integral part of the functionalities to be covered during the development of the project. It is not recommended to make independent lists of security requirements. If they separate there is a risk of prioritizing functional requirements to the detriment of security.

It is also very important to consider pair programming in order to improve the design and make coding easy while taking advantage of the evaluation from different points of view.

We will also need to deal with the uncertainty related to any aspect of the project in an agile and early way since its detection and make it known to all team members and the client if necessary. It has to be addressed by following the principles of as much information as possible.

Other relevant principles are code refactoring in order to facilitate error detection and to improve the quality of the software and to contextualize the project, considering most relevant attack scenarios and vulnerabilities may be affected depending on the software developed and the context in which it will be used. In addition to this, it is interesting to follow the principle of non-reinvention, which establishes that most of the problems in software development have already been solved so, it is not really necessary to reinvent solutions. There are numerous technical solutions, guides or publications to follow.

These are the most important principles, but not the only ones. In total, 23 principles have been defined based on the three pillars that make up the S-SDLC. All of them allow lay the foundation of the S-SDLC and establish the necessary guidelines to apply the methodology.

4.2. Roles

In accordance with the principles defined for the S-SDLC, six different roles have been identified so far by taking the main agile methodologies and the traditional structure of software development teams into consideration. The roles defined and their responsibilities have been meticulously selected to meet the properties and nature of an agile environment and cover security principles established by S-SDLC.

In SCRUM for example, there is a SCRUM master, who leads the team and guides them to comply with the rules and processes of the methodology, as well as a product owner, who focuses on the business part and is responsible for the ROI (return on investment) of the project and the development team, the ones with the necessary technical knowledge to develop the project.

Consonantly with this approach, the project master and software gurus roles are responsible for the implementation of project management and software functionalities. These roles represent traditional actors in software development environments. On the other hand, the security master and security gurus roles introduce a security perspective by covering different security fields and offering expertise and relevant experience.

- The project master is in charge of managing internal and external problems, controlling project deviations, organizing meetings, and playing the role of interlocutor with clients and final users.
- Software gurus make up the development team as specialists from different areas of development, analysis, design, testing, quality, and deployment.
- The security master is recommended to be an expert in logical security. He or she is the final one responsible talking about security-related decisions.
- Security gurus stand like software gurus but considering the security point of view. The areas of specialization in this case cover a wide range of alternatives depending on the specific requirements, but should encompass secure coding, authorization and authentication, network security, cryptography, deployment or DevOps (development operations) or SecDevOps and security in new and emerging scenarios.
- The client is the figure responsible for defining the scope, priorities, and project requirements.
- Users are the subjects for using the final software developed. Depending on the project, client and user roles can be the same.

4.3. Meetings

The definition of the different meetings a methodology should cover is an important process which should be thoroughly evaluated. Meetings should be as effective as possible, covering the required attendants, carrying out to cover the specific objective they were designed for, and being managed to be effective rather than having meetings for the sake of having them.

SCRUM introduces the concept of ceremonies with four types of meetings, providing a framework for empowering the whole team to collaborate and get the work done in a structured manner.

This concept and the principle of simplicity lay down the foundations for the meetings of the proposed S-SDLC.

Seven meetings have been defined for which their scope, objectives, and assistants have been specified. Four of the proposed meetings have a cyclical nature that allow periodic monitoring of the development of the project and adapt to changes following the principle of adaptive response to the change of agile methodologies. These meetings are the sprint start meeting, release meeting, review meeting, and feedback meeting; the first two are mandatory.

- The sprint start meeting aims to establish the planning of activities for the next sprint and to define the scope of the next release. In addition, the activities carried out, problems encountered, and lessons learned, are reviewed and the internal assessment of the generated release is done. Releases are typically inspected in release meetings.
- During release meetings the developed product is presented for client evaluation. This means that software is measured in terms of not only functionality, but also quality and security.
- The review meeting is an internal ceremony like the sprint start meeting, but in this case, it is an extraordinary session whose aim is to solve problems or possible deviations not foreseen in the development of a sprint.
- The feedback meeting is an extraordinary meeting requested by the client due to a change of special relevance in scope of the project, prioritization of requirements or any other situation in which the achievement of the project's objectives may be affected. They are similar to review meetings, but in this case, they remain with the client.

Additionally, three additional meetings have been defined to establish the basis for the beginning and the end of the project.

- Kick-off meetings represent the launch meeting of the project in which the formal communication of the start is made. The meeting identifies the scope and context of the project; final users can provide detailed information on the requirements and any additional information that may affect it, such as time and/or quality requirements, client availability, etc.
- The closing meeting is the end-of-project meeting in which formal communication is carried out to finalize it for all those involved in the project. It includes a final assessment of the results obtained.
- The SRS (software requirement specification) meeting is the requirement's identification meeting. The objective is this case is to obtain a first draft of the list of requirements from which the backlog of activities is made. It is recommended to keep it with the end users to carry out an in-depth analysis of the functionalities and security risks that the development can include. SRS meetings can be carried out as necessary until a solid list of requirements has been prepared and validated by the client.

4.4. Artifacts

In addition to the artifacts previously mentioned, for the proposed S-SDLC, new elements have been introduced to make the implementation of the methodology easy.

Software requirement specification, or SRS, encompasses the complete set of requirements identified in the SRS meeting. The list is subject to updates in any phase prior to the deployment of the software. However, any change must be previously approved.

The backlog of activities includes the complete list of activities planned for compliance with the identified requirements (see Figure 2). It is subject to changes that are produced at sprint or release meetings.

Project					
% completed	Phase	Id.	Activities	Boxes	
100%	Requirements identification	PR-RM-01	Kick-off meeting		
75%	Requirements identification	PR-RM-02	SRS meetting		
50%	Requirements identification	PR-RM-03	SRS	Requirements	
25%	Analysis	PR-RM-04	Functional analysis	Security use cases	
0%	Analysis	PR-RM-05	Security analysis	Abuse cases	Threat modeling

Figure 2. Backlog of activities.

With respect to the boxes, there are functional boxes and security boxes. Both provide a solution to one or more requirements of the SRS, but while functional boxes constitute functional activities that make up the backlog, security boxes focus on security tasks.

Finally, Wikibases are knowledge databases generated from the lessons learned, technical guides, training, etc. acquired in the development of a project.

4.5. Process

The methodology is applied following an iterative process of four steps (see Figure 3):

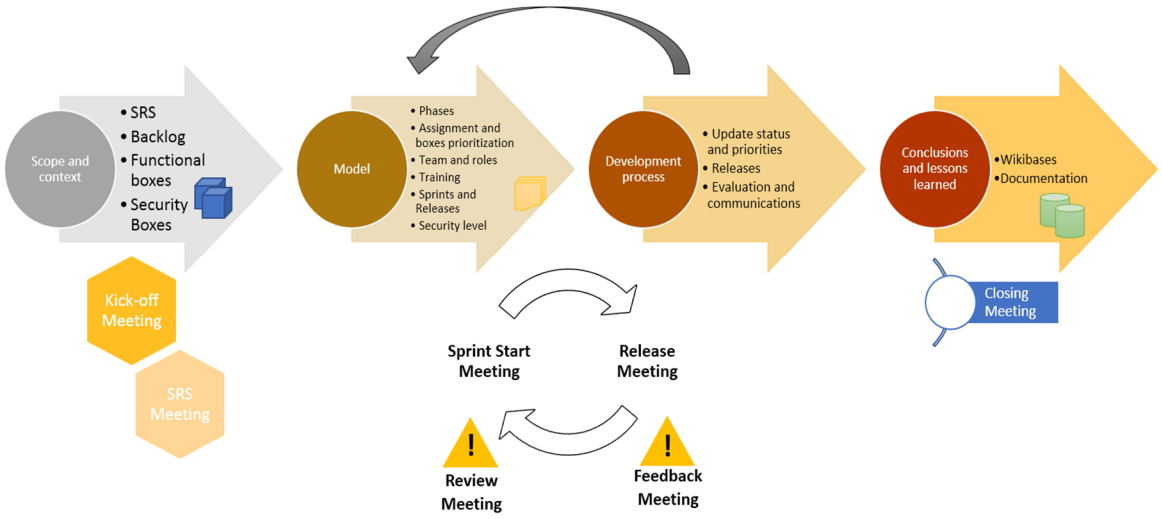

Figure 3. Process.

First of all, and on the basis of the requirements identified in the SRS meeting, the scope and context of the project are defined. During this phase, SRS, the backlog of activities, and functional and security boxes are specified. It is important to identify all the dependencies between the different outputs, for example, to understand what activities are critical and need to be completed before other work can be attempted.

Secondly the model is determined: specifying the phases or steps of the software development process; assigning functional and security boxes to backlog activities; prioritizing and identifying dependencies between boxes and activities; establishing the work team and the assigned roles and training required; pointing out details of sprints and releases; selecting the level of security to apply depending on the security context (development technologies, common vulnerabilities, risks, etc.);

determining the tools and mechanisms for evaluating progress and functional, security, and quality levels for each release; and setting up meetings, client communications, information exchange, etc.

At this stage, the team has selected the phases to follow and has built up the model which should cover at least analysis, design, development, verification and validation, deployment, and maintenance. This is an iterative step which may lead to steps one and two when it could be necessary to update the SRS, backlog of activities or to set out the content of a release. The main activities covered are the update of the current state, priority and dependencies of the SRS and the backlog of activities, the update of the current state, priority and dependencies of the functional and security boxes and the update of the current state and duration of sprints and evaluation of the releases. It also includes an update of the current state of the model, including phases, work team, training, communications, security level, measurement, and evaluation tools, which keep up with possible deviations from the original planning. Communications with client and documentation are also carried out along with documentation.

4.6. Security Activities

In addition to the roles and artifacts, security is articulated through the definition of security activities to be carried out in each of the phases of the life cycle of the proposed software.

The activities fall under a model of minimum phases that every software development project should contemplate. These phases include requirements engineering, analysis and design, coding, testing, verification and validation, deployment, and maintenance.

The activities have been classified according to the level of relevance they have for software security (see Table 5).

Table 5. Security activities.

Activity	Low Relevance (*)	General Interest (**)	High Relevance (***)
Risk analysis and identification of assets		X	
Vulnerability scanning			X
Threat modeling			X
Defining security metrics	X		
Security training and awareness	X		
Security requirements		X	
Cost-benefit analysis of security mechanisms	X		
Security architecture and configuration analysis	X		
Establishment of security design principles	X		
Definition of security use cases		X	
Definition of abuse cases		X	
Use of secure coding guides		X	
Analysis of attacks	X		
Static code analysis		X	
Dynamic code analysis		X	
Code analysis			X
Fuzz testing		X	
Security tests		X	
White/black/gray testing			X
Pentesting			X
Iterative and final security review Vulnerability correlation		X	
Execution of a response plan	X		
Communication of security failures	X		
Strengthening	X		
External analysis		X	
Monitoring of policies, regulations, procedures, and safety guidelines	X		

* For activities of low relevance for software security. ** For activities of general interest for software security. *** For activities relevant to software security.

To ensure compliance with the minimum level of security, it is recommended to carry out at least the following activities:

- Vulnerability analysis: in the phases of requirements identification, analysis, design, and implementation.
- Threat modeling: in the phases of requirements identification, analysis, design, and implementation.
- Pentesting: in the development, testing, and validation phases.
- Code analysis: during development and before deployment.
- White/black/gray testing: during the testing and validation phase.

4.7. Security Levels

For the implementation of security, the methodology presents a scenario based on levels of security covering the following:

- Non-existent (0)
- Very low (1)
- Low (2)
- Medium (3)
- Elevated (4)
- Very high (5)

The minimum-security level is established based on the five security activities indicated in the previous section. If all the activities are carried out in each of the recommended phases, the medium level (3) would be reached.

4.8. Security Boxes

Most agile methodologies are based on the use of cards or boxes that define the activities to be carried out for each of the project phases.

In general, no classification system has been established, which affects the clarity in the scope, progress, and monitoring of the project. Therefore, it is proposed to follow a classification system that allows identification of the cards belonging to each phase contemplated.

For the identification of security activities, it is proposed to create a new type of cards called security boxes that indicate whether a certain security activity is planned, in progress, or completed. These boxes will have additional classifications that indicate the phase or phases of the project in which they will be carried out; for example, an activity such as the modeling of threats will typically be carried out in the design phase while an activity such as static code analysis will be in the development or testing phases.

The security boxes, therefore, can be executed in different phases and have a single state. A box is considered to be open if it is in a planned or developing state and closed when the activity has been completed. A box can be reopened whenever it is considered necessary to repeat an activity again.

Like security boxes, functional boxes can be defined for the other activities of the application. The operation would be similar.

4.9. Information Gathering

As a support for identifying the security activities that are most relevant to the project a simple questionnaire composed of different domains is proposed.

The domains cover programming languages and technologies, authorization and authentication, users and roles, network and connectivity, traceability, recovery time, and data processing.

- Among other aspects, the questionnaire includes questions such as:
- Session time
- Password strength
- Typology of users

- Typology of defined roles
- Connection with Internet, intranet or both
- Implementation of specific rules in the firewall
- Existence of logs in the systems in which they will be deployed
- Generation and exploitation of logs
- Criticality of the system
- Type of information stored
- Personal data

On the other hand, it is important to mention all the security activity-related documentation, such as risks analysis, vulnerabilities assessments reports, threats modeling reports, and testing that must be taken into account due to the importance of this kind of documentation for a correct security implementation for the project.

5. Testing and Validation of the New S-SDLC

Given the complexity of the S-SDLC validation process, a double validation model has been created, constituted by the definition of a checklist which includes all the activities that have to be carried out and validated for an adequate implementation of the S-SDLC (see Table 6), and secondly by the simulation of a typical software development project scenario.

Table 6. FASS (Flexible Agile Secure Software) checklist.

FASS	**Validation Checklist FASS**	**FASS-CHK-19/000 Rev. 000**
Scope and Context (AC)		
SRS		
Id.	**Item**	
AC01	Has the kick-off meeting been held?	
AC02	Has the SRS meeting been held?	
AC03	If affirmative AC02, did it have enough time and did the required personnel participate?	
AC04	If affirmative AC02, have all the requirements been identified and is there enough information to prepare the SRS?	
AC05	Has the SRS been made?	
Backlog		
AC06	Is there enough information to elaborate the backlog of activities?	
AC07	Has the backlog of activities been made?	
AC08	Is the backlog of activities reviewed periodically, especially after possible changes that take place during the development of the project?	
AC09	Is the backlog of activities kept up-to-date and prioritized according to the client and project requirements?	
Boxes		
AC10	Is there enough information available to define the functional boxes?	
AC11	Have the functional boxes been defined?	
AC12	Is the definition of the functional boxes revised periodically, especially after possible changes that take place during the development of the project?	
AC13	Is there enough information available to define the security boxes?	
AC14	Have the security boxes been defined?	
AC15	Is the definition of the security boxes revised periodically, especially after possible changes that take place during the development of the project?	

Table 6. *Cont.*

FASS	Validation Checklist FASS	FASS-CHK-19/000 Rev. 000
Model Selection (EM)		
Phases		
EM01	Have the phases of the project been identified and defined?	
EM02	Are the defined phases appropriate according to the scope, objective, and nature of the project?	
Boxes Assignment and Prioritization		
EM03	Have the boxes been prioritized according to the client and project requirements?	
EM04	Have the boxes been assigned among the different members of the work team?	
EM05	Is the scope of each box adequate?	
Roles		
EM06	Has the project team been defined?	
EM07	Is the project sizing adequate, based on its scope, objective, and nature?	
EM08	Does the project master possess the knowledge and skills necessary to perform his/her functions?	
EM09	Does the project master know his/her tasks and responsibilities?	
EM10	Does the security master possess the knowledge and skills necessary to perform his/her functions?	
EM11	Does the security master know his/her tasks and responsibilities?	
EM12	Do the security gurus have the knowledge and skills necessary to perform their functions?	
EM13	Do the security gurus know their tasks and responsibilities?	
EM14	Do the software surus possess the knowledge and skills necessary to perform their functions?	
EM15	Are the software gurus aware of their tasks and responsibilities?	
Training		
EM16	Is general or specific training required for the execution of the project?	
EM17	If affirmative EM16, has the training covered the necessary knowledge to be able to execute the project?	
Sprints and Releases		
EM18	Have the delivery milestones of each release been defined?	
EM19	Is there an estimate of the scope and content of each release?	
EM20	Has the duration of the sprints been defined?	
EM21	Is the duration of the sprints adequate for the scope, objectives, and nature of the project?	
Security Levels		
EM22	Has the security level been identified and defined?	
EM23	Is the security level appropriate for the scope, objective, and nature of the project?	
EM24	Have the security boxes been identified and defined?	
EM25	Are the security boxes adequate for the scope, objective, and nature of the project?	
EM26	Have potential threats that can affect the software been identified?	
Development (DP)		
Meetings		
DP01	Has the sprint start meeting been held?	
DP02	Did all the required personnel participate in the sprint start meeting?	
DP03	Did the sprint start meeting last long enough to cover the required topics?	
DP04	Are the purpose and scope of the sprint start meeting defined in a clear and understandable way for all participants?	

Table 6. *Cont.*

FASS	Validation Checklist FASS	FASS-CHK-19/000 Rev. 000
DP05	Is the information covered in the sprint start meeting coherent and defined according to the object and scope defined for it?	
DP06	Has the release meeting been held?	
DP07	Did all the required personnel participate in the release meeting?	
DP08	Did the release meeting last long enough to cover the required topics?	
DP09	Are the purpose and scope of the release meeting defined in a clear and understandable way for all participants?	
DP10	Is the information covered in the release meeting coherent and defined according to the object and scope defined for it?	
DP11	Was it necessary to carry out the review meeting?	
DP12	Did all the required personnel participate in the review meeting?	
DP13	Did the review meeting last long enough to cover the required topics?	
DP14	Are the purpose and scope of the review meeting defined in a clear and understandable way for all participants?	
DP15	Is the information covered in the review meeting coherent and has it been defined according to the object and scope defined for it?	
DP16	Was it necessary to carry out the feedback meeting?	
DP17	Did all the required personnel participate in the feedback meeting?	
DP18	Did the sprint Feedback Meeting last long enough to cover the required topics?	
DP19	Have the purpose and scope of the feedback meeting been defined in a clear and understandable way for all participants?	
DP20	Has the information covered in the feedback meeting been coherent and defined according to the object and scope defined for it?	
DP21	Has the closing meeting been held?	
DP22	Did all the required personnel participate in the closing meeting?	
DP23	Did the closing meeting last long enough to cover the required topics?	
DP24	Has the object and scope of the closing meeting been defined in a clear and understandable manner for all participants?	
DP25	Is the information covered in the closing meeting consistent and has it been defined according to the object and scope defined for it?	
	Status and Priorities Update	
DP26	Is the backlog of activities kept updated and prioritized during the development of the project?	
DP27	Have client and project requirements been considered in each update of the backlog of activities?	
DP28	Are boxes updated and prioritized during the development of the project?	
DP29	Have client and project requirements been considered for the update of the boxes?	
	Releases	
DP30	Are the releases performed in accordance with the planification?	
DP31	Are the releases developed with the agreed functional, quality, and security requirements?	
DP32	Are the releases delivered and presented to the client within the agreed deadlines?	

Table 6. *Cont.*

FASS	Validation Checklist FASS	FASS-CHK-19/000 Rev. 000
	Evaluation and Communications	
DP33	Are regular and continuous communications with the client maintained?	
DP34	Are the content, frequency, and scope of the communications adequate for the client and for the project?	
DP35	In case of extraordinary communications, have these been adequate according to their content and scope for the client and for the project?	
	Testing	
DP36	Have functional tests been performed?	
DP37	Have security tests been performed?	
DP38	Are the results of the different types of tests carried out reviewed?	
DP39	Are problems/errors/vulnerabilities identified communicated to those responsible for their correction?	
DP40	Are problems/errors/vulnerabilities identified during the tests solved?	
DP41	Are the tests repeated once the bugs have been fixed?	
	Conclusions and Lessons Learned (CL)	
	Wikibases	
CL01	Has at least one Wikibase been generated with the information and lessons learned from the project?	
CL02	Have generated Wikibases been made known to other projects?	
	Documentation	
CL03	Are activities related to project documentation planned?	
CL04	Is software documentation generated?	
CL05	Is project documentation generated?	
CL06	Is the generated documentation updated after each functional modification?	
CL07	Is the generated documentation accessible?	
CL08	Is the generated documentation adequate according to the scope, objective, and nature of the project?	

5.1. Checklist

The checklist has been structured in four sections: scope and context, model selection, development, and conclusions and lessons learned.

5.2. Application Scenario

Below, a typical software development project scenario has been developed to test the implementation of the proposed methodology, as well as the validation checklist.

The application considered is aimed at managing the transport process made by a courier company. The application shows on Google Maps the actual location of the vehicles, as well as information related to the service. The monitoring map can be visualized on different types of devices within the intranet of the organization. The application connects every few seconds to the database and updates the location of each vehicle on the map. Likewise, it connects to other systems to obtain different information that is either processed or simply displayed.

It is composed of the following modules:

- A batch process that sends planned trip data and scheduled delivery times for each day.
- A web interface used for monitoring.

The application is developed in Java 8 and Oracle PL/SQL. A web application server will be required. The project duration is estimated for three months.

From a functional point of view, it is important to establish an adequate mechanism for updating the current situation and defining valid routes for messengers. Considering a security point of view, data protection seems to be one of the most important aspects to be taken into consideration, due to the fact that it contains information of the location of the messengers and delivery routes. Encryption mechanisms should be applied in the storage and transmission of information. Confidentiality and availability are the two most relevant security dimensions. Naturally, the range of vulnerabilities in a web-based application could be wide. Standards such as OWASP (Open Web Application Security Project) or WASC (Web Application Security Consortium) should be taken into account, but, if we stay focused on the most common vulnerabilities, Java and Oracle's possible vulnerabilities should be reviewed, as well as attacks such as SQL (Structured Query Language) injection, cross-site scripting (XSS) or cross-site request forgery (CSRF) in the web module. To deal with those kinds of threats, secure testing guides such as the OWASP Testing Guide v4, the OWASP cheat sheet implementation, and Oracle and Java secure configuration guides could be useful to identify and fix all the possible vulnerabilities. Finally, an appropriate measure to consider would be the process of hardening the servers on which the application is going to be deployed.

With the help of the validation checklist, the following shows the results that would be achieved during the development of the project, considering each particular section of the checklist:

- Scope and context (AC):
    - Taking the scope of the application into consideration, it seems more or less clear to identify all the requirements. In that way, kick-off and SRS meetings could be held for a reasonable time and in an appropriate manner, lasting long enough to prepare the SRS.
    - In regard to backlog, the activities could be prioritized according to project requirements. One of the possible difficulties could be keeping the backlog up-to-date.
    - Finally, functional boxes should be clearly defined with the available information. Regarding security boxes, the following activities would be the most appropriate: risk analysis and identification of assets, vulnerability scanning, threat modeling, definition of security use cases, definition of abuse cases, use of secure coding guides, static code analysis, dynamic code analysis, code analysis, fuzz testing, security tests, white/black/gray testing, pentesting, iterative and final security review, and external analysis.
- Model selection (EM):
    - Phases: The project should at least cover requirements engineering, analysis and design, coding, testing, verification and validation, deployment, and maintenance.
    - Boxes assignment and prioritization: This is an important step in the development of the project. For this reason, functional- and security-related activities should be distributed among the different phases selected and guarantee the involvement of each member of the project.
    - Roles: The minimum set of roles would include a project master, security master, security guru, and three software gurus.
    - Training: No specific training would be required.
    - Sprints and releases: Based on the project planning, the releases to be delivered could include requirements specification, use case diagrams, data model, design model, architecture, web application prototype, batch procedures, web application with information of positioning, web application with path information, web application with service information, web application and proven batch procedures, web application and validated batch procedures, and documentation. Regarding sprints, one-week sprints would be suitable to be planned during the requirements identification and analysis phases. From the design phase until the end of the development, the duration could be increased to two weeks.

- Security Level: At first, it is important to take the threats that can most likely affect the application into consideration. Those threats are unauthorized access to the application, unauthorized modification of information, information leaks, man in the middle (MitM) attacks, manipulation of activity records, lack of updating the standard software, dissemination of harmful software, denial of service (DoS), denial of distributed service (DDoS), and lack of awareness regarding information security.

With that information and considering the scope of the application the recommended security level would be elevated (4).

- Development (DP):
  - Meetings: During the development of the project, all team members are responsible for carrying out and participating in the different meetings established, such as sprint start meetings, release meetings, review meetings, feedback meetings, and closing meetings.
  - Status and priorities update: As we previously said regarding backlog, one of the possible difficulties could be keeping the boxes and requirements up-to-date. The whole team is accountable.
  - Releases: At this point, it is important to assure functionality, quality, and security in each release.
  - Evaluation and communications: Focus on regular and continuous communication with client.
  - Testing: For the project, it would be relevant to focus on the results of security tests, white/black/gray testing, pentesting activities, and static and dynamic code analysis. These can be useful to detect both functional and security problems and can help save time.
- Conclusions and lesson learned (CL):
  - Wikibases: It mentions some guides of how to secure a batch process, the mechanisms used to protect and at the same time, allow real time positioning of different elements, in this case, messengers, or the methods for conducting functional and security tests in a web-based environment.
  - Documentation: Finally, it is important to emphasize in the real need of generating appropriate documentation.

5.3. Expert Assessment

Finally, an evaluation of the methodology by several experts in software development and security was carried out to complete the evaluation process. The participating experts have relevant experience in different fields such as software development, cybersecurity, and computer science and have taken part in software development projects at both the business and academic level with proven experience. Some of them are university professors with research experience in the mentioned fields and other work or have worked in different companies in the industry.

The proposed methodology has been well received by experts who all agree that security is far from being well considered in software development projects and is not properly addressed in agile methodologies. The comments received are the following:

- This method is adequate to accomplish a secure development. It properly addresses security activities, roles, artifacts, security levels and boxes, meetings and communications, and activities such as reviewing testing reports or the process of vulnerabilities correlation inside the sprints of an agile development process.

- The set of security activities is very complete, and the method has taken the necessary flexibility to assess and take into consideration different security levels.
- SCRUM does not take security issues into account, so it is time to consider it as any other activity to be carried out. Therefore, it is a very good idea to present it as another step to take in agile methodologies.
- The proposed validation checklist is detailed and comprehensive and covers all the necessary activities to implement S-SDLC properly.
- The definition of this S-SDLC makes the formalization of the introduction of security in software development from the initial stage much easier. It is the key to obtain secure software, allowing vulnerabilities to be discovered before they make considerable damage in the amount of time spent in development or, sometimes, even at a project stage when they are almost impossible to solve.
- The idea of controlling the software to be developed and also the libraries, dependencies or frameworks that have been used to build the software is very interesting
- This study agrees with the creation of a Wikibase to preserve the knowledge and valuable experience obtained in the development of the project.
- This study agrees with the sentence: "It is not recommended to make independent lists of security requirements. If they separate, there is a risk of prioritizing functional requirements to the detriment of security." Keeping independent security requirement lists leads to prioritizing the development of functional requirements.
- This study agrees with pair programming. At this point, on the one hand, a change of mentality is required so that this principle can be applied, and on the other, it should be implemented in a rotary basis, so those "pairs" can constantly be changing within as the team members are acquiring new experience.
- Regarding code refactoring, it seems to be essential, and at the same level as the unit tests, which are the basis of robust, quality, and secure software (quality and security are strongly connected). The software is not built linearly but iteratively and therefore continuous improvement requires continuous software changes. Therefore, software must be prepared to adapt to changes in an agile way.
- This study liked the orientation given to how the team should work, the importance of knowledge management, information sharing, and documentation, something which many methodologies have considered incompatible with agile development.
- The documentation generated could be improved in order to optimize future code maintenance. The generation of WikiBases should remain in a pending state until the end of the project so the know-how generated during development would be as complete as possible.
- Based on our latest projects (cloud-based projects), the importance of security in software development is highlighted. It is also important to continue evolving in different ways such as the development of a DevSecOps environment to implement automated functional and security tests.
- This study believes that the issue of security could be extended, at least with one more paragraph, giving examples of essential security activities that should be carried out at each release, activities that should have be performed at the end of each phase, and activities that should be carried out just once. It is important to make clear how crucial security is in software development.
- Another relevant point to emphasize would be the importance of the users in software development. For the success of the project, I think it is essential to have highly motivated users who will really add value to the software. The client's role is also very important, but we strongly support the users' side, especially if they are really committed to the project. Knowing the user also helps to know and reduce security vulnerabilities.
- During the development of the project, delivery times were not been scrupulously maintained due to lack of time since the criterion of finalizing the security revisions for each release was prioritized.

We believe that the haste and stress generated by the start of the next project prevented us from carrying out the closing session (closing meeting). Otherwise, we had no problems applying the proposed methodology.

- An important part of agile methodologies is to manage changes in requirements and design, especially those that affect security. It is important to remember that these changes must be verified and validated.
- The use of tools for project management in software development projects which implement agile methodologies could be interesting. They could facilitate the transparency of activities and allow the measurement of the level of progress and the status of all the activities at all times.
- According to principle #22, there are people who believe that the best documentation is the code itself. Unit tests should be also taken into consideration. It is possible to say that the best documentation for a project is when unit tests link directly to the system's own requirements.

The results obtained in the survey carried out by experts on the proposed methodology are shown in Figure 4. The graphics show the different categories of the methodology, which are the same as the validation checklist. For each one, the experts have evaluated the suitability of the methodology and its relevance for reaching success in software development projects.

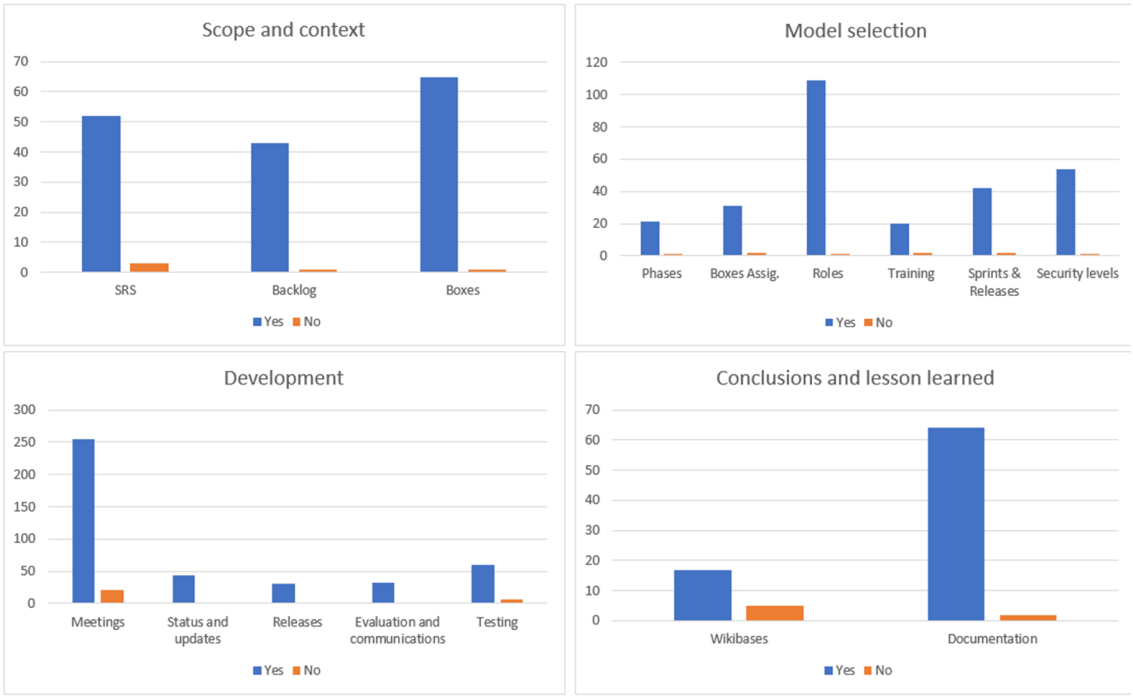

Figure 4. Survey results.

First, the scope and context section show that experts consider that both kick-off and SRS meetings are adequate. Additionally, in the case of the SRS meeting, experts conclude that it is crucial that it lasts long enough time and that all the required personnel take part. The main difficulties encountered in this case lie in the requirements' identification. Experts believe that obtaining an SRS with at least enough information from the beginning of the project is a complex task to commit. The sections corresponding to the backlog and the boxes follow a similar trend: backlog definition of the different activities (functional and security) is suitable and allows a more or less reliable idea of all the necessary tasks to be performed.

As for model selection, the experts consider that the activities proposed for selecting the artifacts that will be necessary during the development of the project is well-suited, especially, the phases, training, sprints and releases, security levels, and boxes assignments. Finding relevant training,

complying with the planned duration of the sprints, and identification of the most appropriate level of security for the project were possible difficulties. Experts highlight the identification of roles, which adequately cover both functionality and security and the definition of security levels.

If we focus on development, the results show a similar trend to the previous categories. The meetings section is shown as the subcategory in which experts find it more difficult, for both suitability and for conducting all the required meetings. In particular, two of the experts agree that problems may arise concerning the closing meeting. This is due to the lack of time and resources needed in the final phases of the project. Another sees some possible barriers in the attendance of all the necessary members to the different meetings. On the contrary, the development of the releases, the update of activities, and the communications with clients are driven in a suitable manner and reinforce client satisfaction. Regarding testing activities, some experts agree that, despite their importance, the results of the security tests are not properly reviewed.

Finally, the conclusions and lessons learned section shows mixed results. On the one hand, both Wikibases and documentation are seen as the main elements of any software development project. However, on the other hand, it may be difficult for the team to find time to generate Wikibases once the development is completely finished. As for the documentation, some of the experts conclude that the code itself represents the key of documentation. They add that in some cases, the documentation may not be detailed enough.

The survey has also provided an evaluation of the completeness and consistency of the items of the validation checklist. The experts conclude that its scope, covers all artifacts and elements developed in the methodology and the proposed questions help to implement it effectively. As a final remark, the conclusions and lessons learned section could include some additional questions, especially for Wikibases, which is a really innovative and useful artifact and which could be expanded for further research.

5.4. Validation Conclusions and Suitability of the Proposed Methodology

The results of the evaluation process show that that the application of the methodology has some important and positive effects in the development of a software project.

First of all, security from the beginning implies considering the likely threats which could affect these kinds of applications, such as unauthorized access or modification of data or denial of service, and which are usually materialized in SQL injection or XSS attacks, and are detected and solved during the time planned for the project, with no extra time nor costs for the client. Fixing SQL injection or XSS vulnerabilities may take weeks for the whole team. Extrapolating this to the scenario, a team composed of a project master, security master, security guru, and three software gurus would require additional costs for the extra time required which could be estimated between 60 and 90 euros for the project and security masters and 30–60 euros for the security and software gurus for each additional hour required for the project. Other vulnerabilities such as Oracle or Java patches application, could imply recompiling source code or adjusting settings of the servers. It seems clear that solving errors when the application is already deployed could require stopping current services, getting the team back together to find proper solutions to the problems detected, retaking the tests, rewriting documentation, etc.

Another benefit is the guarantee of reaching a high level of software quality and functionality. This is because agile properties focus on functionality, but also because of the security activities added to the project and the communication and the involvement of the client. Let us have a look at the security boxes selected for the scenario: pentesting, white/black/gray testing, and especially the process of correlating vulnerabilities, assure that those possible threats will be identified and addressed before the release of the software. Moreover, security tests improve functional ones helping to detect functional related problems.

As for the internal benefits, the generation of Wikibases could serve for future projects, for example, when conducting security tests or securing batch processes and servers. The clear definition of the meetings, phases or roles from the beginning, the use and update of a backlog of activities available for

all team members, or the process of looking for specific training if necessary, help to ensure the success of the project.

Finally, principles such as pair programming, code refactoring, or the promotion of documentation stand as key elements in a software development environment.

6. Conclusions

As has been shown and explained, considering security from the first stage not only for the development, but also for each project phase, guarantees quality, functionality, and obviously, security, in addition to reducing the probabilities of increasing the costs of fixing errors in the final stages or carrying out additional security audits.

The process of cost quantification that can be incurred if security is not taken into account is a complex task that depends on multiple factors but is also very representative in order to show the importance of security.

This scenario has intended to show the most common vulnerabilities that software development projects have to deal with, but naturally, the range of threats is wide, and the specific scope and technologies used in each software development project makes it difficult to address the issue.

Let us suppose an additional scenario of a software development project in which six vulnerabilities have been identified after the end of the development phase. These vulnerabilities are based on the following OWASP categories: injection, broken authentication, sensitive data exposure, and security misconfiguration. The additional costs in which it is incurred should consider the resolution of each particular vulnerability, the execution of functional tests, and the repetition of the security activities or audit. All this implies additional hours for the project involving almost the whole team. SQL injection vulnerabilities, for example, may involve modifying the totality of the queries by parameterizing commands or by using stored procedures. The vulnerabilities related to security misconfiguration can be due to cookies configuration, cached responses of http requests, or to the process of updating the version of a framework implemented. Broken authentication usually refers to problems related to the process of authentication and authorization of the software. Depending on the scope of the application such as the existence of multiple web services or the access to various modules with different permissions, the resolution costs may be higher or lower. Finally, sensitive data exposure could refer to key exposure of information or not implementing code obfuscation techniques. In this case, it seems crucial to incorporate encryption and protection mechanisms.

In the above-mentioned case the resolution of the vulnerabilities considering a standard scenario could take between three and four weeks. To these must be added the execution time of at least one additional security audit that validates the implemented solutions. It must be borne in mind that the identification of vulnerabilities in the final stages of a project involve mobilizing the resources, designing new security solutions, developing new pieces of code, and testing and validating them, so it is necessary to add new development hours to the project, plus the cost of conducting additional security activities if an S-SDLC has not been followed. Therefore, the project can be extended to two more months until the software has been validated and deployed in the production environment. Moreover, the additional time dedicated to the project implies the non-participation or partial participation of the team in other projects.

As we have seen, the application of the proposed methodology, like any other S-SDLC does not guarantee security 100%, but minimizes the possibilities of vulnerability detection, especially after software release, increments the possibilities of reaching success in terms of not only functionality but also quality and security, and maximizes the added value to the software for the benefits of developing robust software and involving each member of the team in the process.

Unfortunately, nowadays it is still usual to hear from software vulnerabilities in applications of all types, and this leads to a bigger problem taking into account the increasingly level of dependency we have on software. New scenarios such as cloud environments or DevOps and new kinds of problems

which will need to be faced, will be better addressed from a security point of view, and at that point, the proposed methodology seeks to be a useful tool to use.

Author Contributions: Conceptualization, J.d.V.M., J.B.H., J.R.B.H., and J.A.S.M.; methodology, J.d.V.M., J.B.H., and J.R.B.H.; validation, J.d.V.M., J.B.H., J.R.B.H., and J.A.S.M.; formal analysis, J.d.V.M. and J.B.H.; investigation, J.d.V.M., J.B.H., and J.R.B.H.; resources, J.d.V.M., J.B.H., J.R.B.H., and J.A.S.M.; data curation, J.d.V.M., J.B.H., J.R.B.H., and J.A.S.M.; writing—original draft preparation, J.d.V.M., J.R.B.H., J.R.B.H., and J.A.S.M.; writing—review and editing, J.d.V.M., J.R.B.H., J.R.B.H., and J.A.S.M.; visualization, J.d.V.M., J.B.H., J.R.B.H., and J.A.S.M.; supervision, J.d.V.M., J.B.H., J.R.B.H., and J.A.S.M.; project administration, J.d.V.M., J.B.H., J.R.B.H., and J.A.S.M.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Khan, M.U.N.; Zulkernine, M. On selecting appropriate development processes and requirements engineering methods for secure software. In Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference, Seattle, WA, USA, 20–24 July 2009; pp. 353–358.
2. Brian, C.; Jacob, W. *Secure Programming with Static Analysis*; Pearson Education: London, UK, 2007.
3. Rea-Guaman, A.; Sánchez-García, I.; San Feliu, T.; Calvo-Manzano, J. Maturity models in cybersecurity: A systematic. In Proceedings of the 12th Iberian Conference on Information Systems and Technologies, Lisbon, Portugal, 21–24 June 2017.
4. Lipner, S. The trustworthy computing security development lifecycle. In Proceedings of the 20th Annual Computer Security Applications Conference, Tucson, AZ, USA, 6–10 December 2004; pp. 2–13. [CrossRef]
5. Secure Software Development Life Cycle. Available online: https://www.owasp.org/images/9/9d/OWASP-LATAMTour-Patagonia-2016-rvfigueroa.pdf (accessed on 11 September 2019).
6. Beiter, M. Steps in a Secure Software Development Lifecycle Model. Available online: https://www.michael.beiter.org/2013/11/29/steps-in-a-securesoftware-development-lifecycle-model-1/ (accessed on 11 September 2019).
7. Dadu, M.I. Secure software development model: A guide for secure software life cycle. In Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS), Hong Kong, China, 17–19 March 2010; Volume 1.
8. Buinevich, M.; Izrailov, K.; Vladyko, A. The life cycle of vulnerabilities in the representations of software for telecommunication devices. In Proceedings of the 18th International Conference on Advanced Communication Technology (ICACT), Pyeongchang, Korea, 31 January–3 February 2016; pp. 430–435.
9. Hudaib, A.; AlShraideh, M.; Surakhi, O.; Khanafseh, M. A survey on design methods for secure software development. *Int. J. Comput. Technol.* **2017**, *16*, 7047–7064.
10. Characteristics of Agile Development Success. Available online: https://resources.collab.net/agile-101/agile-development-success (accessed on 11 September 2019).
11. Martin, R. *Clean Code: A Handbook of Agile Software Craftsmanship*; Pearson Education: London, UK, 2008.
12. Martin, R. *Agile Software Development, Principles, Patterns, and Practices*; Prentice Hall: Upper Saddle River, NJ, USA, 2002.
13. Cohn, M. *User Stories Applied: For Agile Software Development*; Addison-Wesley Professional: Boston, MA, USA, 2004.
14. What is Agile Project Management? Available online: https://www.apm.org.uk/resources/find-a-resource/agile-project-management/ (accessed on 11 September 2019).
15. Agile Manifesto. Available online: http://agilemanifesto.org/iso/en/manifesto.html (accessed on 11 September 2019).
16. Agile Software Development. Available online: https://en.wikipedia.org/wiki/Agile_software_development#The_Agile_Manifesto (accessed on 11 September 2019).
17. Noopur, D. *Secure Software Development Life Cycle Processes: A Technology Scouting Report*; Software Engineering Institute: Pittsburgh, PA, USA, 2005.
18. Trujillo, D.; Chávez, A. *Sistema de Control. de Versiones para el Desarrollo de Software Seguro*; Fundación Universitaria los Libertadores: Bogotá, Colombia, 2016.

19. Security Software Building Security in Seven Touchpoints for Software Security by Gary McGraw. Available online: http://www.swsec.com/resources/touchpoints/ (accessed on 11 September 2019).
20. Microsoft SDL—Microsoft Talks about SDL and How it Changed the Security Landscape in the Software Industry. Available online: https://mspoweruser.com/microsoft-talks-about-sdl-and-how-it-changed-the-security-landscape-in-the-software-industry/ (accessed on 11 September 2019).
21. CLASP Concepts. Available online: https://www.owasp.org/index.php/CLASP_Concepts (accessed on 11 September 2019).
22. Humphrey, W. *The Team Software Process (TSP)*; Software Engineering Institute: Pittsburgh, PA, USA, 2000.
23. Shirazi, M.R.A.; Jaferian, P.; Elahi, G.; Baghi, H.; Sadeghian, B. *RUPSec: An Extension on RUP for Developing Secure Systems—Requirements Discipline*; Amirkabir University of Technology: Tehran, Iran, 2007; Volume 1, pp. 1–5.
24. BSIMM. Available online: https://www.bsimm.com/ (accessed on 11 September 2019).
25. Chandra, P. Software Assurance Maturity Model. 2011. Available online: https://opensamm.org/downloads/SAMM-1.0-en_US.pdf (accessed on 11 September 2019).
26. Fléchais, I. *Designing Secure and Usable Systems*; University College of London: London, UK, 2005; pp. 57–59.
27. Mahendran, F. *Secure Software Development Model (SSDM)*; Singapore Computer Society: Singapore, Singapore, 2012.
28. Howard, M.; LeBlanc, D. *Writing Secure Code*, 2nd ed.; Microsoft Press: Hoboken, NJ, USA, 2003.
29. Bassil, Y. A simulation model for the waterfall software development life cycle. *arXiv* **2012**, arXiv:12056904.
30. Lindo, A.C. Modelos de Desarrollo Seguro del Software. Available online: http://web.fdi.ucm.es/posgrado/conferencias/AndresCaroLindo-slides.pdf (accessed on 11 September 2019).
31. Futcher, L.; von Solms, R. SecSDM: A model for integrating security into the software development life cycle. In *Fifth World Conference on Information Security Education*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 41–48.
32. Essafi, M.; Jilani, L.; Ghezala, H.B. S2D-prom: A strategy oriented process model for secure software development. In Proceedings of the International Conference on Software Engineering Advances (ICSEA 2007), Cap Esterel, France, 25–31 August 2007; p. 24.
33. Hall, A.; Chapman, R. Correctness by construction: Developing a commercial secure system. *IEEE Softw.* **2002**, *19*, 18–25. [CrossRef]
34. Mead, N.; Stehney, T. *Security Quality Requirements Engineering (SQUARE) Methodology*; Software Engineering Institute; Carnegie Mellon University: Pittsburgh, PA, USA, 2006; pp. 1–7.
35. Oracle Software Security Assurance. Available online: https://www.oracle.com/assets/software-security-assurance-2293569.pdf (accessed on 23 October 2019).
36. SCRUM. Available online: http://www.scrum.org/ (accessed on 11 September 2019).
37. What is Scrum? Available online: https://www.atlassian.com/agile/scrum (accessed on 11 September 2019).
38. Fuentes, J.R.L. *Desarrollo de Software Ágil: Extreme Programming y Scrum*; IT Campus Academy: Vigo, España, 2014; ISBN 978-1502952226.
39. Lasa Gómez, C.; Álvarez García, A.; de las Heras del Dedo, R. *Métodos Ágiles. Scrum, Kanban, Lean*; Anaya Multimedia: Madrid, Spain, 2017; ISBN 9788441538887.
40. Extreme Programming: A Gentle Introduction. Available online: http://www.extremeprogramming.org/ (accessed on 11 September 2019).
41. Kanban Encyclopedia: Concepts and Terms. Available online: https://kanbanize.com/kanban-resources/getting-started/kanban-encyclopedia/ (accessed on 11 September 2019).
42. Lean Software Development. Available online: http://www.javiergarzas.com/2012/10/lean-software-development-2.html (accessed on 11 September 2019).
43. Feature Driven Development (FDD) and Agile Modeling. Available online: http://agilemodeling.com/essays/fdd.htm (accessed on 23 October 2019).
44. Firdaus, A.; Ghani, I.; Jeong, S.R. Secure feature driven development (SFDD) model for secure software development. *Procedia-Soc. Behav. Sci.* **2014**, *129*, 546–553. [CrossRef]
45. Kosten, S. Cypress Data Defense. Available online: https://www.cypressdatadefense.com/technical/matching-sdlc-model-to-the-bestsecurity-processes/ (accessed on 8 September 2019).

46. Sass, R. How to Balance between Security and Agile Development the Right Way. Available online: https://resources.whitesourcesoftware.com/blog-whitesource/how-to-balance-between-security-and-agile-development-the-right-way (accessed on 11 September 2019).
47. Komlo, C.; Gómez, M. Incorporating Security Best Practices into Agile Teams. 2016. Available online: https://www.thoughtworks.com/insights/blog/incorporating-security-best-practices-agile-teams (accessed on 11 September 2019).
48. Ghani, I.; Azham, Z.; Jeong, S.R. Security backlog in Scrum security practices. In Proceedings of the Malaysian Conference in Software Engineering (MySEC), Johor Bahru, Malaysia, 13–14 December 2011. [CrossRef]
49. SDL-Agile Requirements. Available online: https://msdn.microsoft.com/es-es/library/windows/desktop/ee790620.aspx (accessed on 11 September 2019).