ISSN 2277-3061
Volume 16 Number 7
International journal of Computers and Technology

# A Survey on Design Methods for Secure Software Development

Amjad Hudaib[1], Mohammad AlShraideh[1], Ola Surakhi[1], Mohammad Khanafseh[1]

[1]University of Jordan, King Abdullah II School for Information Technology, Computer Science Department, Amman -Jordan.

## Abstract

Software provide services that may come with some vulnerabilities or risks. Attackers perform actions that break security of system through threats and cause a failure. To avoid security vulnerability, there are many security-specific concepts that should be determined as requirements during software development life cycle in order to deliver a strong and secure software. This paper first, survey a number of existing processes, life cycle and methodologies needed for developing secure software based on the related published works. It starts by presenting the most relevant Secure Software Development Lifecycles, a comparison between the main security features for each process is proposed. The results of the comparison will give the software developer with a guideline which will help on selecting the best secure process. Second, the paper list a set of the most widely used specification languages with the advantages and disadvantages for each.

## Keywords:

Software, Security, Software Development Life Cycle

## 1. Introduction

A secure software is the software where unauthorized person cannot access it, modify it, or attack it. The degree of such security is measured by the existing number of security vulnerabilities. The software with no vulnerabilities is a high secure software, where the software with at least one vulnerability is insecure software [1].

In order to include security in the software engineering, the security aspects should be included from the beginning of the software development life cycle. The secure software engineering is the process of designing, building, and testing software so that it becomes secure, this includes secure software development life cycle (SSDLC) processes and secure software development (SSD) methods. A SSDLC process considers security aspects of the software during the development life cycle by using SSD methods. SSD methods include, among others, security specification languages, security requirements engineering processes, and software security assurance methods [24].

The importance of including security on the software developing comes from the high cost of removing system errors which may cause a security vulnerability after developing.

This paper survey a set of secure software development life cycle processes, identify the characteristics for each one, and presents a comparison between them based on the security activities that are included in the development lifecycle, this will be useful for the software developers by helping him choosing the correct process.

## 1.1 Software Security Goals

There are three principal dimensions to achieve security in the computer system, confidentiality, integrity and availability. These three aspects also are referred as CIA.

**Confidentiality** means to disclose information to people or programs that are authorized to have access to that information.

**Integrity**, assures that a system performs its intended function in an unimpaired manner, free from deliberate or inadvertent unauthorized manipulation of the system.

**Availability**, assures that systems work promptly, and service is not denied to authorized users.

Threats to the confidentiality of the system can disclose information to people or programs that are not authorized to have access to that information. Threats to the integrity of the system and its data can damage or corrupt the software or its data. Threats to the availability of the system and its data can restrict access to the software or its data for authorized users.

Over the years, different security mechanism was used to achieve these goals like authentication and authorization. But the rate of attacks to computer system is increasing, and the situation may be critical especially for large systems. Because of that, many researchers pay attention on the software security field in order to produce a high secure system.

## 1.2 Software Security Basic Concepts

This section will explain some of security terms that are used in this paper.

1. *Asset:* Is anything that has value to the organization, its business operations and their continuity, including information resources that support the organization's mission.


http://cirworld.com/ DOI: 10.24297/ijct.v16i7.6467

2. *Vulnerability:* A weakness in the design, operation, implementation or any process in the system which expose the system to a threat. [10] defined it as a weakness of an asset or group of assets that can be exploited by one or more attacker.
3. *Threat:* A possible danger that may result in harm of systems and organization.
4. *Attack:* An actual event done by a person; attacker to harm as asset of the software through exploiting a vulnerability.
5. *Risk:* a potential for loss, damage, or destruction of an asset as a result of a threat exploiting a vulnerability.
6. *Software Security Requirement:* is a non-functional requirement that elicit a control, constraint, safeguard, or countermeasure to avoid or remove security vulnerabilities from requirements, design or code [24-26].
7. *Confidentiality:* means to disclose information to people or programs that are authorized to have access to that information.
8. *Integrity:* assures that a system performs its intended function in an unimpaired manner, free from deliberate or inadvertent unauthorized manipulation of the system.
9. *Availability:* assures that systems work promptly, and service is not denied to authorized users.
10. *Process:* is an instance of a computer program that is being executed.
11. *Secure software process:* is a set of activities used to develop and deliver a secure software solution.

## 2. Secure Software Development Life Cycle processes

The stages of the software development are:

1. Requirements definition and specification
2. Design document containing system abstraction and their relationships
3. Coding, Programming components of the system
4. System testing
5. Implementation and Maintenance

Adding the security activities to software development life cycle will in general include the following in Figure 1.

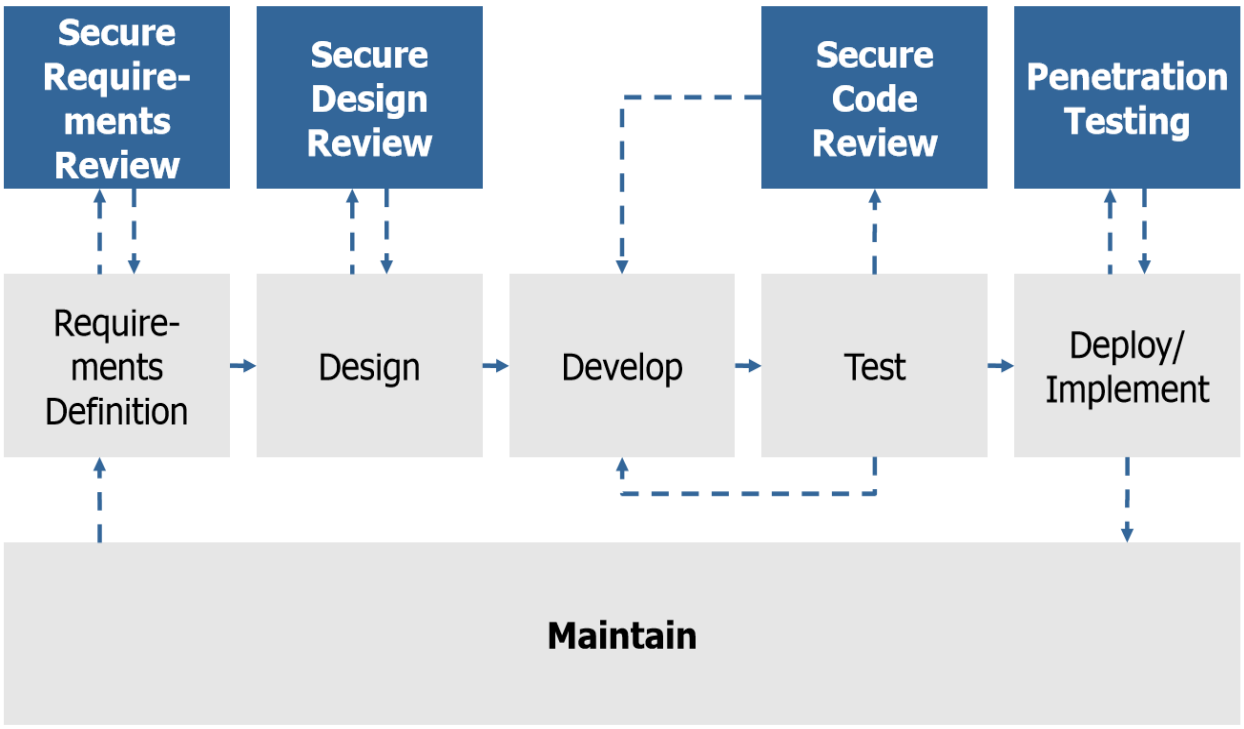


Figure 1: Security in the SCLC

There are a set of secure software development methods that had been proposed to add security concept to the software. The software development life cycle is used to develop software and the security activities are added in different phases according to the specific method. This section summarizes a literature review for the most popular secure software development process.


DOI: 10.24297/ijct.v16i7.6467

## 1. McGraw's Secure Software Development Life Cycle Process [2,3]

To build a secure software, McGraw suggests 7 touchpoint activity areas, connecting to software development artefacts. In lifecycle order:

- Abuse cases (in requirements), describe the desired behavior of the system under different kinds of abuse
- Security requirements (in requirements), identify functional security requirements
- Architectural Risk analysis (in design), considers security during design
- Risk-based security tests (in test planning), test security functionality using a standard method
- Code review and repair (in coding), eliminate problems
- Penetration testing (in testing and deployment), find real problems
- Security operations (during deployment), managing the security of the deployed software

Figure 1 shows the life cycle security touch points.

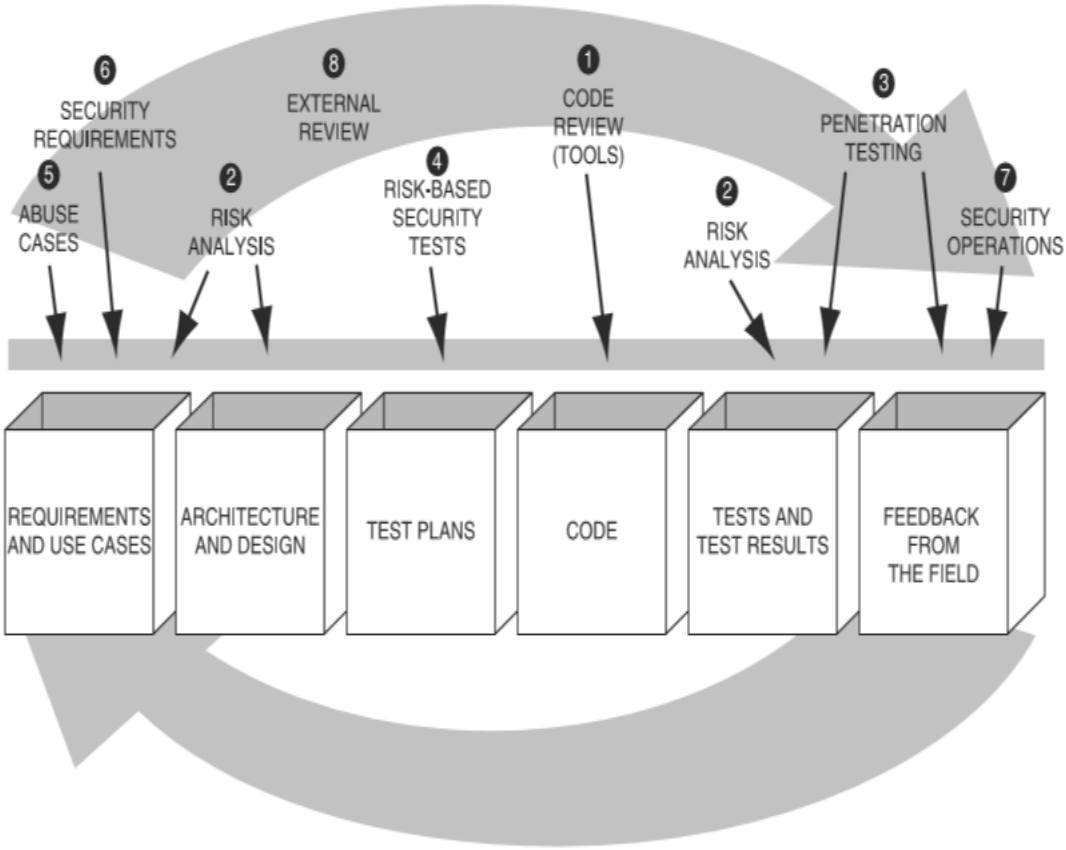


Figure 1: McGraw's Secure Software Development Life Cycle Process

## 2. Microsoft's Trustworthy Computing Security Development Lifecycle (MS SDL)

Microsoft's Trustworthy Computing Security Development Lifecycle (SDL) is a process developed be Microsoft to add a series of secure activity to each phase of Microsoft's software development process as follows: during requirement phase, the security feature requirements are defined based on the customer demands, in the design phase the MS SDL suggests a set of activities to be performed such as threat modeling for security risk identification, identifying components that are critical to security or needs special attention during testing. In the implementation phase, use of static analysis code-scanning tools and code reviews. After completing implementation, the complete software is tested focusing on the security critical components of the software during the testing phase. A final code review of new as well as legacy code is used during verification phase. And finally, during the release phase, a Final Security Review is conducted by the Central Microsoft Security team [4,12,13]. The overall process is shown in Figure 2.


DOI: 10.24297/ijct.v16i7.6467

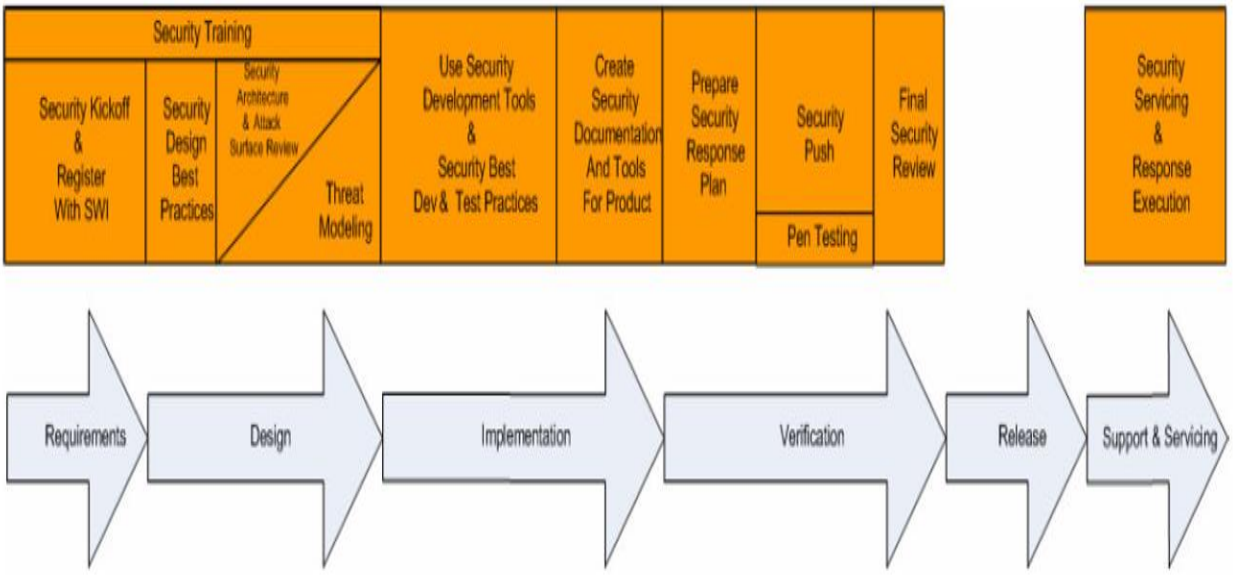


Figure 2: Microsoft's Trustworthy Computing Security Development Lifecycle

## 3. Team Software Process for Secure Software Development (TSP Secure)

TCP process is specifically designed for software teams, to build a high-performance team and help them in plan their work to achieve the best performance. The TSP Secure focuses directly on the security of software in three ways: planning, development and management, and training for developers about security related aspects and other team members. In the initial phase; planning, the team identify security risks, security requirements, secure design, code review, use of static analysis tools, unit tests, and Fuzz testing, and produce a detailed plan to be used in the development phase during a series of meetings. Next, the plan is executed, and the team ensure that all the security activities are taken place [14]. TSP security strategy has multiple defect removal point during SDLC, which enables problems to be easily removed. Figure 3 shows defect removal activity in the software development life cycle.

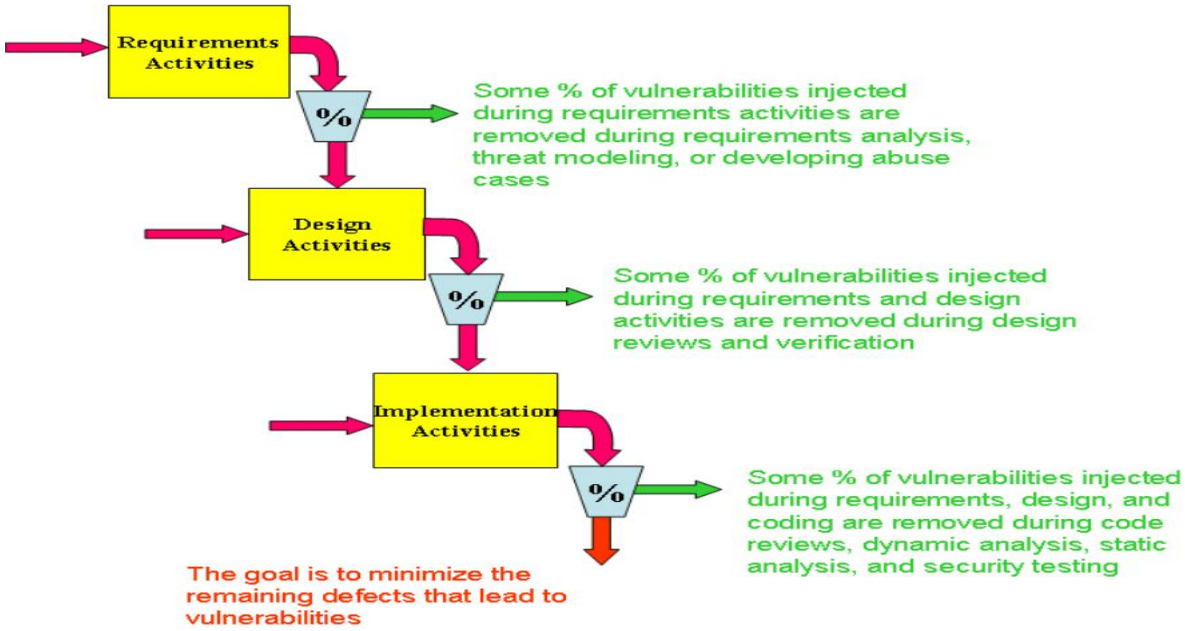


Figure 3: Vulnerability Removal Filters

Every time defects are remove, they are measured. The measurement points help the team to know where they stand against their goals, helps them decide whether to move to the next step or to stop and take corrective action, and indicates where to fix their process to meet their goals.

The team considers the following questions when managing defects:

- What type of defects lead to security vulnerabilities?
- Where in the software development life cycle should defects be measured?
- What work products should be examined for defects?
- What tools and methods should be used to measure the defects?
- How many defects can be removed at each step?
- How many estimated defects remain after each removal step?

TSP-Secure includes training for developers, managers, and other team members [33].

4. Agile Methods

Agile is a new family in the software engineering which include extreme programming, scrum, Crystal Methodologies, Lean Software Development, Feature Driven Development, and Dynamic Systems Development Methodology. All of these methods that fall under the umbrella of "Agile", are different in their methodology, but they are all have some common principles, such as short development iterations, minimal design upfront, emergent design and architecture, collective code ownership and ability for anyone to change any part of the code, direct communication and minimal or no documentation (the code is the documentation), and gradual building of test cases [15]. In order to add security to the agile method, the security requirements must be adaptable and simple. [16] listed in his article "Towards Agile Security Assurance," a table that shows the compatibility of common security assurance activities with Agile methods. It shows that almost 50% of traditional security assurance activities are not compatible with Agile methods (12 out of 26), less than 10% are natural fits (2 out of 26), about 30% are independent of development method, and slightly more than 10% (4 out of 26) could be semi-automated and thus integrated more easily into the Agile methods.

5. Secure Software Development Process Model (S2D-ProM)

Is a strategy oriented process model that provide the developers and software engineers from beginners to experts with a guidance support that cover most resources and knowledge need to develop a secure software. The process provides two level of guidance, the first one is strategic which helps the developers choosing one among several strategies. The second level guidance is tactical helping developers achieving their selection for producing secure software [17].

It gives the developers with more than one option while advancing from one phase to another in order to achieve flexibility and extensibility in the secure software development process [17]. The security development process here is intention oriented. At any moment, the engineer has an intention, a goal in mind that he wants to fulfill.

6. Comprehensive, Lightweight Application Security Process (CLASP)

Is a lightweight process, consists of 24 top-level security related activities that can be fully or partially incorporated into software that is being constructed during software development life cycle [18]. Each CLASP Activity is divided into discrete process components and linked to one or more specific project roles, this provides a guidance to project participant and results in incremental improvements for security in the software life cycle. For example, threat modeling and risk analysis are performed during requirement and design phase [19]. In the design and implementation phase, it suggests secure design guidelines and secure coding standards [20-23]. inspections, static code analysis, and security testing are performed in the assurance phase. Figure 4 shows the 24 activities of CLASP.

| | |
|---|---|
| Project Manager | 1. Institute security awareness program<br>2. Monitor security metrics<br>3. Manage security issue disclosure process |
| Requirements Specifier | 1. Specify operational environment<br>2. Identify global security policy<br>3. Document security-relevant requirements<br>4. Detail misuse cases |
| Architect | 1. Identify resources and trust boundaries<br>2. Identify user roles and resource capabilities<br>3. Integrate security analysis into source management process (Integrator)<br>4. Perform code signing (Integrator) |
| Designer | 1. Identify attack surface<br>2. Apply security principles to design<br>3. Research and assess security posture of technology solutions<br>4. Annotate class designs with security properties<br>5. Specify database security configuration<br>6. Address reported security issues |
| Implementer | 1. Implement interface contracts<br>2. Implement and elaborate resource policies and security technologies<br>3. Build operational security guide |
| Test Analyst | 1. Identify, implement and perform security tests<br>2. Verify security attributes of resources |
| Security Auditor | 1. Perform security analysis of system requirements and design (threat modeling)<br>2. Perform source-level security review |


DOI: 10.24297/ijct.v16i7.6467

Figure 4: CLASP security activity role mapping

7. Correctness by Construction [12]

The Correctness by Construction methodology developed by Praxis Critical Systems is a process for developing high-integrity software [33]. It has been used to develop safety-critical and security-critical systems with a great degree of success [34]. Correctness by Construction is based on the following tenets: do not introduce errors in the first place and remove any errors as close as possible to the point that they are introduced.

The process is based on the strong belief that each step should serve a clear purpose and be carried out using the most rigorous techniques available to address that particular problem. Specifically, the process almost always uses formal methods to specify behavioral, security, and safety properties of the software. The belief is that only by using formality can the necessary precision be achieved.

The seven key principles of Correctness by Construction are:

1. Expect requirements to change. Changing requirements are managed by adopting an incremental approach and paying increased attention to design to accommodate change. Apply more rigor, rather than less, to avoid costly and unnecessary rework.
2. Know why you're testing. Recognize that there are two distinct forms of testing, one to build correct software (debugging) and another to show that the software built is correct (verification). These two forms of testing require two very different approaches.
3. Eliminate errors before testing. Better yet, deploy techniques that make it difficult to introduce errors in the first place. Testing is the second most expensive way of finding errors. The most expensive is to let your customers find them for you.
4. Write software that is easy to verify. If you don't, verification and validation (including testing) can take up to 60% of the total effort. Coding typically takes only 10%. Even doubling the effort on coding will be worthwhile, if it reduces the burden of verification by as little as 20%.
5. Develop incrementally. Make very small changes, incrementally. After each change, verify that the updated system behaves according to its updated specification. Making small changes makes the software much easier to verify.
6. Some aspects of software development are just plain hard. There is no silver bullet. Don't expect any tool or method to make everything easy. The best tools and methods take care of the easy problems, allowing you to focus on the difficult problems.
7. Software is not useful by itself. The executable software is only part of the picture. It is of no use without user manuals, business processes, design documentation, well commented source code and test cases.

   These should be produced as an intrinsic part of the development, not added at the end. In particular, recognize that design documentation serves two distinct purposes:

- To allow the developers to get from a set of requirements to an implementation. Much of this type of documentation outlives its usefulness after implementation.
- To allow the maintainers to understand how the implementation satisfies the requirements. A document aimed at maintainers is much shorter, cheaper to produce and more useful than a traditional design document.

8. Appropriate and Effective Guidance for Information Security (AEGIS)

This process based on the spiral model, it integrates into normal software engineering lifecycles, as can be seen in its application to the Spiral model of software development as shown in Figure 5.


DOI: 10.24297/ijct.v16i7.6467

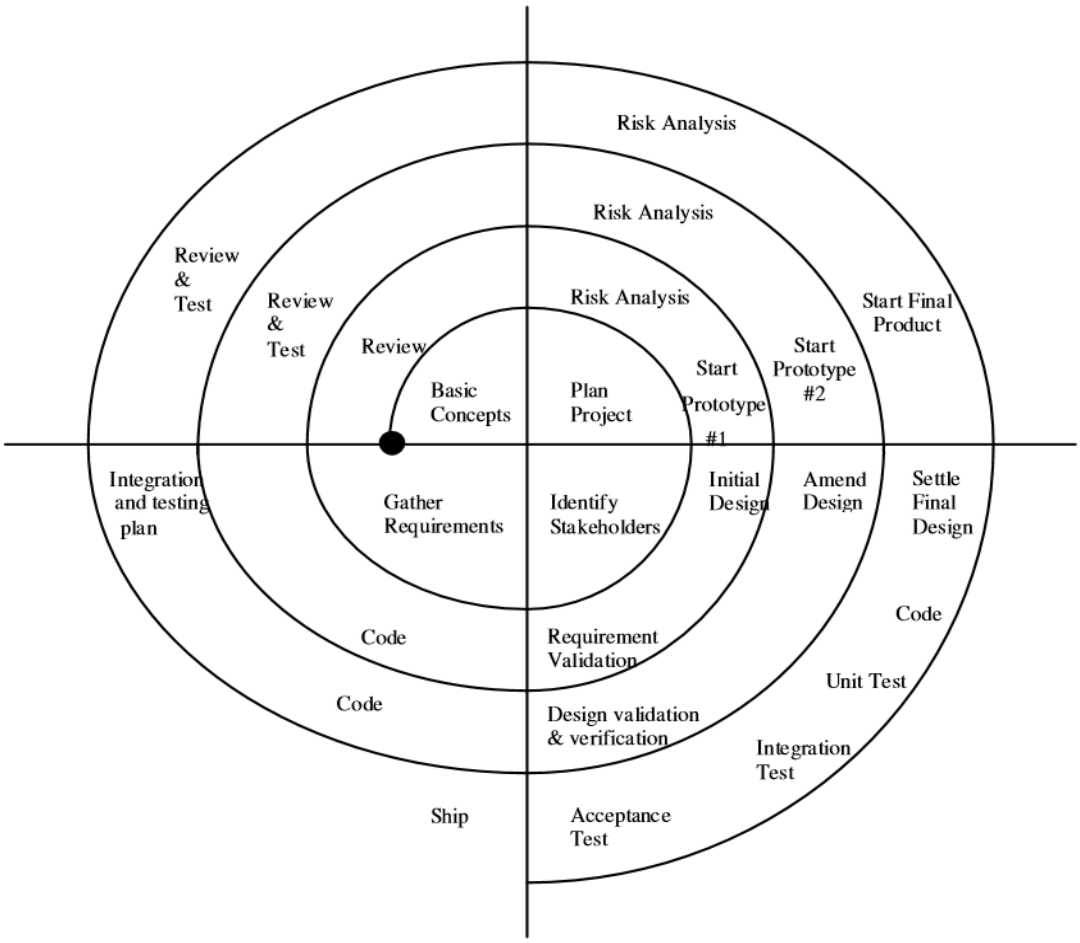


Figure 5: AEGIS Spiral Model of Software Development

The process identifies security requirements first by determining system assets and their relationships, then risk analysis in which vulnerabilities, threats and risks are identified [5, 31].

9. The Secure Software Development Model (SSDM)

This model incorporates various security activities into a waterfall software development life cycle model, as shown in Figure 6. The security training provides adequate security education to stakeholders in software development. Threat model identifies attackers and their capabilities and is performed during requirements phase. After that, security specification must be defined by stating the guidelines on how security will be achieved. Penetration Testing is the only SSD activity for the security assurance phase where Capabilities of the software in preventing attacks are tested here [6, 30],


DOI: 10.24297/ijct.v16i7.6467

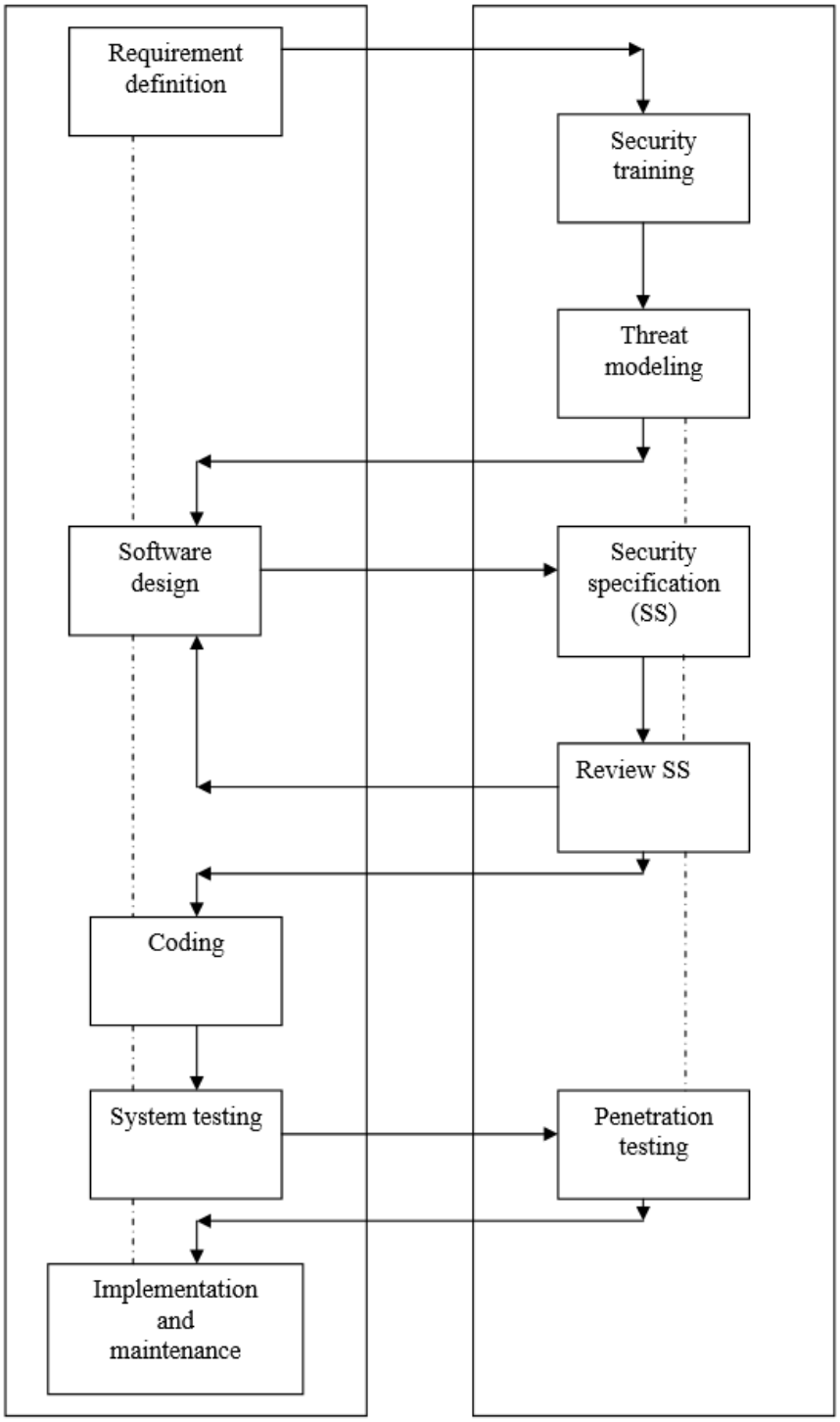


Figure 6: Secure Software Development Model (SSDM) [6]

10. Security Quality Requirements Engineering (SQUARE) methodology

This model initially focuses on building security concepts at the early stages of the software development cycle. It provides a means for eliciting, categorizing, and prioritizing security requirements for information technology systems and applications. The final output of SQUARE is a security requirements document that is designed to satisfy the security goals of the organization [7]. Then, it has been enhanced into SQUARE+R [8] to incorporate risk analysis in all phases of the development to ensure security. It uses a static and dynamic code analysis tools for the security assurance phase.

## 3. Comparison of existing secure software development life cycle process SSDLC

This section presents the strengths and weakness point for each SSDLC with a comparison between them according to the activities that should be added during development life cycle to develop a secure software. The secure activities that can be added to each phase in the software development life cycle are listed below in table 1.

Table 1: security activities for each phase in the software development life cycle

| requirement engineering | Design | Implementation | Security Assurance | Resources |
|---|---|---|---|---|
| 1. Threat modeling activity<br>2. Requirement specification review activity<br>3. security requirement specification language<br>4. risk analysis | 1. design guideline and secure design pattern<br>2. identify possible software threats<br>3. review activity<br>4. errors and threats identification<br>5. risk analysis | 1. Secure programming language selection | 1. security static analysis<br>2. security code review | 1. Use secure resources<br>2. configuration review<br>3. network configuration review |

The details of each activity are:

1. Secure software development activities for requirement engineering

A Software Requirement Specification or SRS is a document which records expected behavior of the system or software that needs to be developed.

Each SSDLC should specify the use of SSD activities for each (i.e., requirements engineering, design, implementation, and security assurance), in order to develop a secure software, the recommended activities should be added to each phase in the software development life cycle phases. For requirement engineering phase different secure activity must be found as follow

1) Threat modeling activity which refers to identify possible threats that can attack the software, so that appropriate security feature must take into consideration through requirement specification phase.
2) Requirement specification review activity, through this activity security error was specified.
3) Use of security requirement specification language to specify security requirement which can help to remove identified language.
4) Evaluation for security state of requirement specification using risk analysis to prioritize the identified security requirements.

All upper activities must be used through first phase of software development life cycle to achieve secure requirement specification, some of current models used some of these activities and some other model does not use them, and many processes recommend a similar set of SSD methods to be used during requirements engineering. Almost all the SSDLC processes (except SSAI [27] and MS SDL [4, 13]) perform threat modeling and/or risk analysis during the requirements specification phase. Moreover, all processes except SSAI and SSDM recommend defining security features to mitigate identified threats.

2- secure software development through design phase.

Second phase of software development life cycle refer to design phase, through this phase different activities must be take into account to be found through this phase to achieve secure software development, these activities are as follow:

1) Through entail design development the secure design guideline and secure design pattern must be followed, and secure design decision should be specified using secure design specification language.
2) Must identify possible software threats by performing threat modeling on the initial design phase, where this activity was different from that which found on requirement phase because the data or information through this phase was more than that on previous requirement phase.
3) Design review activity which refers to review design phase to make sure that no security error was found, where error that can be found here on this phase was caused by requirement specification phase error, for that all error on previous phase must be fixed.
4) Errors and threats identifications must be used as a basis to security requirement specification for design phase.
5) To prioritize security requirements for design phase the risk analysis should be performed [2, 26].

Some processes take into consideration these activities during its design phase such as MS SDL [29] and CALSP [18], both provide comprehensive set of SSD methods for design phase.

3- Security activity in implementation phase

Secure programming language selection is the important step to achieve secure implementation phase, through following a secure coding standard and standard guideline, some model suggest that implementation using secure language to achieve

secure implementation, like Apvrille and Pourzandi [11], and S2D-ProM [17], where some other suggest to use secure coding standard guideline to achieve secure implementation like MS SDL [4, 13], SecSDM [6, 30], McGraw's SSDLC process [2], S2D-ProM, CLASP [18], MS SDL goes further and suggests that certain security assurance activities should be performed during implementation.

4- SSD Activities for Security Assurance and Testing

Some activities must be taken into account through testing and security assurance process, vulnerability assessment and fuzzing must be implemented on testing phase also security static analysis using code scanning tools and security code review after finishing the coding or implementation phase. We must note that all these activities do not guarantee security of software but identify errors and vulnerability. Different models of SDLC models use these activities for testing phase such as the exception of AEGIS [5, 31], SSDM [6], and SecSDM [30], all the processes recommend using multiple security assurance methods such as security testing, code reviews, static code analysis.

5- secure development resources.

To achieve secure software development, a secure resource should be available for development process, where the resources can be possible mitigations for each error or vulnerability, where this knowledge will be helpful to avoid errors that can lead to vulnerabilities in the first place [31]. Other important point on secure development resource phase to review different resources as do server configuration review and network configuration review, these two steps avoid vulnerability which come from the network and resources.

Figure 7 summarize the secure activities that can be added to the software development life cycle. Table 3 summarize a comparison of SSDLC processes according to security activities in each phase of the development cycle phases.


Figure 7: secure development for software development life cycle


http://cirworld.com/
DOI: 10.24297/ijct.v16i7.6467

Table 3: comparison of SSDLC processes

| | SSD activities for requirement engineering | SSD activities for Design | SSD activities for Implementation | SSD activities for Security Assurance | Resources |
|---|---|---|---|---|---|
| **McGraw's SDL** | Identify security requirements | risk analysis | None | using abuse cases and security requirements | Secure design guidelines |
| **MS SDL** | identify security objectives and required security features. The | identification of the components that are critical to security, identification of design techniques/guidelines | following the secure coding standards | security code reviews and security testing | Secure design guidelines |
| **TSP Secure** | Threat modeling. Specification of abuse cases. Risk Analysis | Design patterns. State machine design and verification | Following secure programming standards and guidelines | Fuzz testing. Penetration testing. Use of static code analysis tools. Code reviews | None |
| **Agile** | Guidelines. Specification analysis. Review. | Use secure design principles. Formal and informal validation. external review | Informal requirement traceability. High level programming and tools | Security testing. Vulnerability and penetration testing. | None |
| **S2D-ProM** | Following security standards. Risk analysis. Identification of errors. Specification of security features | Using security modeling language, security patterns. Risk analysis. Design reviews. Model checking | Following secure coding standards. Use of a secure programming language | Risk analysis. Code reviews. Use of static code analysis tools. | None |
| **CLASP** | Risk analysis. Threat Modeling. Identification of attackers and attack surface. Specification of misuse cases with their mitigations, security features. | Following design guidelines. Annotation of class diagrams with security information. Threat modeling. Risk analysis | Following secure coding guidelines | Code reviews. Use of static code analysis tools. Security testing | Secure design and implementation guidelines. A comprehensive list of more than 200 types of vulnerabilities with their possible mitigations. |


DOI: 10.24297/ijct.v16i7.6467

| | | | | | |
|---|---|---|---|---|---|
| **Correctness by Construction** | Requirements are identified using the formal specification notation Z | Develop incrementally | SPARK | Debugging and verification<br>formal. methods. | Formal methods |
| **AEGIS** | identifying assets and performing risk analysis | Design activity according to requirements | None | None | None |
| **SSDM** | Threat modeling. Specification of security policy | Following the security policy. | None | Penetration testing. | None |
| **SQUARE** | eliciting, categorizing, and prioritizing security requirements | None | None | static and dynamic code analysis tools. | None |

## 3.1 Ten ways to infuse security into your software development life cycle

Implement security measure should be a top of priority to ensure the security level of the developed software; the following ways can help on producing secure software as follow:

- Asses Land Space, by do requirement gathering and good understood what the customer need by determine scope and boundaries, and by identify the stockholder to define what they want, and other by identify the process gaps.
- Incorporate industry standard security model for software development process through requirement gathering process, based on this step secure software will be built from the beginning of development and this way will be cost effective to not repeat the test of the program more than one time if this development does not achieve security level recommended from the customer.
- Educate personal on software security; this can be done through training or deployment phase to educate the customer who will use the software for security property of software and how to avoid unsafe actions which will affect the security level for the program.
- Perform security on the phase of requirement gathering, though this step we must incorporate security on initial steps of security development phases to insure of producing secure software as performing initial risk analysis during requirement gathering phase to promote security activities in addition phases within SDLC.
- Initiate institute for a comprehensive risk management analysis, this institute will help on identifying major risk and execute a mitigation plane, this institute have responsibility like ensure proper security design, ensure effectiveness guide in SDLC execution.
- Perform architecture review and threat modeling, this way implemented through design phase of SDLC phases, this step is important to identify different threats and risk on early stage of development which will be cost effective when identify it here rather than identity it later one software was developed, different critical tool for detect design flaws as analyzing fundamental design principle and assign the attack surface, enumerating various threat agents and other identifying weakness and gaps in security controls.
- Doing code review during implementation phase, this step is complete step for using secure coding standard and static code analysis performing secure coding review as a condition for pass of development of software as secure software development.
- Execute test plane and perform penetration test through verification phase of software development life cycle phases, this step other important to make sure that the developed product perform as expected in runtime scenario.

Last step refers to Deploy Software product through deployment phase of SDLC this is essential to successful release to production through QA and acceptance testing are completed

## 4. Specification languages for software security

The specification languages are a way used to detect existing vulnerabilities by simulating different conditions of attacks and its securities. In this section, we introduce some of these specification languages.

### 4.1 Definitions related to specification languages

#### 1. Use Case Diagram

Is a graphic description that defines system's user and activities and the interaction between them.

#### 2. Activity Diagram

Is a behavioral diagram that represent the workflow of the system. It describes the flow from one activity to another.

#### 3. Class Diagram

Is a structural diagram for a particular system. It describes classes, packages, associations and interfaces. It is widely used in the object-oriented system

#### 4. Abuse case

It describes the interaction between actor and activity, where interaction is harmful.

#### 5. Misuse Case

The case that should not be happening in the system. It is used to detect system vulnerabilities.

#### 6. Attack Language

Is the language that is used to describe, recognize, react and report an attack.

#### 7. Specification Language

Describes what the system should do**.** what data to be processed. Check requirements and verify system validation.

### 4.2 Different Specification Languages

In this section, we will introduce the most widely used specification languages.

#### 1. STATL

Is an extended finite state machine-based executable attack specification language, proposed in 2002 by Eckmann et al [35], it represents an attack by using states and transitions where states represent a different characteristic of an attack and transition connects two states. Each transition must have associated event which when occurs fires the transition. And each transition may have a code to be executed when fired. STATL is a language to support misuse detection and is widely used in network-based attacks and host-based attacks [36].

#### 2. UMLSec

UMLSec is an extension of UML language, it is developed by Jurjens et al [37] in 2002. Its main purpose is to focus on specifying role-based access control policies which can be considered as security requirements to help in the development of a security-critical system. It provides 9 stereotypes to turn normal UML language into security purpose language.

#### 3. UMLIntr

UMLintr is an extension of UML, proposed in 2006 by Hussein et al., and uses use case diagrams, class diagrams, state charts, and package diagrams to specify attacks [38]. The attack scenarios are described in misuse state machine.

#### 4. Abstract State Machine Language (AsmL)

Is an extended finite state machine-based executable software specification language, and used to specify attack scenario. This attack scenarios can be automatically translated into Snort rules which can then be used with an extension of the IDS Snort. Such attack scenarios are able to capture more attacks with multiple steps using context information [41].

#### 5. AsmlSec

Is an extension of specification language Asml [39], and proposed by Raihan et al. in 2007 [40]. Some features were missing in Asml to represent different attacks. AsmlSec integrated those missing features with Asml by adding states and transitions. States are different situations of an attack and transitions are different events that trigger another state. A transition can only take place if guard or condition is satisfied. If a system reaches any state, which is an attack state then the system generate an alert [36].

#### 6. Snort Rules


DOI: 10.24297/ijct.v16i7.6467

It uses attacks scenarios specified as rules to detect attacks over the network. It specifies what action should be taken if the rule is matched to a network packet, the source and destination IP addresses and ports, the protocol of the observed network, and direction of network packet. A number of options can also be specified. These options range from logging a message to searching for a particular string in the packet [42].

## 4.3 Comparison of Existing Security Specification Languages

Table 4 summarize the advantages and disadvantages for each specification language [36].

| | Advantages | Disadvantages |
|---|---|---|
| **STATL** | 1. can represent domain independent attack scenarios<br>2. can represent a large range of network and host-based attacks<br>3. It can relate previous attacks to detect or predict new attacks | 1. its own syntax only can be understood by experts.<br>2. To work with STATL, Secure Software developer has to learn a new language. |
| **UMLSec** | 1. Fair Exchange: In e-commerce fair exchange is very important. It protects the buyer and seller from cheating. Fair exchange ensures that, if payment is made then the buyer will receive his product or he will get his money back.<br>2. Secure Information Flow: It checks whether the information is leaked. Even a small amount of leaked information can be used to regenerate full information.<br>3. Secure Communication Link: It ensures that communication between multiple users is secure against intruders.<br>4. Confidentiality: It checks that resources will be only accessed by authorized party. | 1. UMLSec mainly worked on validation and requirement, check of the system<br>2. Some stereotypes functionalities only work if the exact string is used in the action name of a UML diagram |
| **UMLIntr** | 1. UMLIntr introduced 8 tagged values. It is possible to represent skill of the attacker, security level of the victim, the severity level of the attack, the exploitation method of the attack, the action of the attacker, events needed to detect attack, gained access level of the attack and initial access level of attacker by using this tagged value.<br>2. UMLIntr added 19 stereotypes | 1. UMLIntr has a stereotype that can represent only four attacks (DoS attack, Remote to user attack, User to root attack, Probing attack) but other attacks (e.g.: distributed denial of service, side channel attack, Cloudbased attack etc.) cannot be presented<br>2. UMLIntr is only suitable for detecting Intrusion but not for other security demands like verification, validation, authorization |
| **AsmL** | 1. Can represent usage senareo<br>2. Tool support | 1. Can not Formulate Basic Security Requirements<br>2. Can not represent security mechanism |
| **AsmlSec** | 1. Use the state transition diagram to represent an attack. | 1. Can not directly read HTTP packets |


DOI: 10.24297/ijct.v16i7.6467

| | | |
|---|---|---|
| | It is easier to understand attack states and conditions of attack from a transition<br>2. AsmlSec has its own compiler which generates attack signatures from AsmlSec specification<br>3. AsmlSec tested on DARPA datasets to represent CrashIIS attack signatures, which was successful | 2. AsmlSec does not support regular expression<br>3. Documentation is hard to read |
| **Snort Rules** | 1. Can represent usage senareo<br>2. Tool support | 1. Can not Formulate Basic Security Requirements<br>2. Can not represent sequrity mechanism |

## 5. Security Software Testing

Security testing is an important to identify vulnerabilities and ensure security functionality. It is about testing of security requirements related to security properties like confidentiality, integrity, availability, authentication, authorization, and non-repudiation. To achieve that, many techniques can be used, this section will provide an overview of the recent security testing techniques.

### a. Risk Analysis

Risk analysis is done during design phase in order to review security requirements and to identify security risks. Threat modelling is a methodical process to identify threats and vulnerabilities in software. It is recommended that application have a threat model developed and documented, and created as early as possible in the SDLC [50].

### b. Model-Based Security Testing

It is an approach that validate software system requirements that are related to security properties. It combines security properties like confidentiality, integrity, availability, authentication, authorization and non-repudiation with a model of the SUT and identifies whether the specified or intended security features hold in the model. SUT is a test model that is built from informal requirements, one execution trace of such a model acts as a test case: input and expected output. It can generate an infinite number of tests, in order to reduce them a test selection criteria are applied [43].

### c. Code-Based Testing and Static Analysis

Static analysis is dealing with detecting the vulnerabilities from the program code, which is an important task to do as it helps in removing the vulnerabilities at early stage of the development [44]. Analyzing the source code can be done manually or automated. The manual approach can be done by an expert who has a good experience on the implementation and security, the results of the analysis is send back to the developers to fix the identified vulnerabilities. The automatic approach, also called Static Application Security Testing (SAST) [45], is a tool analyses the program code of a software component automatically and reports potential security problems.

### d. Vulnerabilities Scanning

A tool scans the executing application software for input and output of known patterns that are associated with known vulnerabilities. An example of it is signature-based scanning, which is used for web server, database management systems, and some operating systems, these vulnerability patterns, or ”signatures”, are comparable to the signatures searched for by virus scanners, or the ”dangerous coding constructs” searched for by automated source code scanner, making the vulnerability scanner, in essence, an automated pattern-matching tool [50].

### e. Penetration Testing and Dynamic Analysis

Penetration testing is an application or system is tested from the outside in a setup that is comparable to an actual attack from a malicious third party [43], the entity that is conducting the test has potentially only limited information about the system under test and is only able to interact with the system’s public interfaces.

The NIST Technical Guide to Information Security Testing and Assessment [46] partitions the penetration testing process in four distinct phases:

1. Planning: define and document the component of the application that are relevant to the test.

2. Discovery: discover the external interface and make a vulnerability analysis in which the applicable vulnerability classes that match the interface are identified.
3. Attack: the identified interfaces are tested through a series of attack attempts.
4. Reporting: documents all findings along with their estimated severances.

Fuzzy testing is a type pf dynamic testing, that feed the program with a random data until it crashes [47], it is an effective technique for finding vulnerabilities in software.

### f. Security Regression Testing

This technique ensure that changes made to a system do not harm its security, are therefore of high significance and the interest in such approaches has steadily increased [48]. The changes may be due to new business needs, new regulations, and new technologies. [49] classify regression testing techniques into three categories: test suite minimization, test case prioritization and test case selection.

Test suite minimization seeks to reduce the size of a test suite by eliminating redundant test cases from the test suite.

Test case prioritization aims at ordering the execution of test cases in the regression test suite based on a criterion.

Test case selection deals with the problem of selecting a subset of test cases that will be used to test changed parts of software [43].

## 6. Summery and Conclusion

Choosing the suitable secure software development process is a challenging thing for the stakeholder to make the decision which one to use without knowing the difference between them.

This paper surveys a set of secure software development life cycle processes, gives a short overview for each and compare between them according to the secure activities included at each phase. The results show that most of them do not include the security activities in all the development phases, the processes, MS SDL and CLASP cover more aspects of secure software development.

We also summarized a set of the specification languages used to identify attack in the software with a list of advantages and disadvantages for each one.

## References

[1] Schneider, T., “Secure Software Engineering Processes: Improving the Software Development Life Cycle to Combat Vulnerability”, SQP VOL. 9, NO. 1, 2006, http://www.asq.org

[2] McGraw, G., Software Security: Building Security In, Addison Wesley, 2006

[3] Verdon, D. and McGraw, G., “Risk Analysis in Software Design,” IEEE Security and Privacy, IEEE CS Press, 2004, volume 2, number 4, pages 79-84.

[4] Lipner, S., “The Trustworthy Computing Security Development Lifecycle,” In Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC ‘04), Tucson, Arizona, USA, 2004, IEEE CS Press, pages 2-13.

[5] Flechais, I., Mascolo, C., and Sasse, M. A., “Integrating Security and Usability into the Requirements and Design Process,” International Journal of Electronic Security and Digital Forensics, Inderscience Publishers, Geneva, Switzerland, 2007, volume 1, number 1, pages 12-26.

[6] Sodiya, A. S., Onashoga, S. A., and Ajayi, O. B., “Towards Building Secure Software Systems,” Issues in Informing Science and Information Technology, Informing Science Institute, California, USA, 2006, volume 3, pages 635-646.

[7] Mead, N. R., Hough, E., and Stehney, T. “Security Quality Requirements Engineering (SQUARE) Methodology,” Technical Report CMU/SEI-2005-TR-009, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 2005.

[8] Yu,W. D. and Le, K., “Towards a Secure Software Development Lifecycle with SQUARE+R,” In Proceedings of the 36th International Conference on Computer Sofwtare and Applications Workshops, Izmir, Turkey, 2012, pages 565-570

[9] Jain, S.and Ingle, M., “Techno-Management View of Secured Software Development,” In Proceedings of the 6th International Conference on Software Engineering (CONSEG), Indore, India, 2012, pages 1-6.

[10] British Standard Institute, Information technology -- Security techniques -- Management of information and communications technology security -- Part 1: Concepts and models for information and communications technology security management BS ISO/IEC 13335-1-2004

[11] A. Apvrille and M. Pourzandi, “Secure Software Development by Example,” IEEE Security and Privacy, IEEE CS Press, 2005, vol. 3, no. 4, pp. 10-17.

[12] Noopur Davis, “Secure Software Development Life Cycle Processes: A Technology Scouting Report”, December 2005, Software Engineering Process Management

[13] Lipner, Steve & Howard, Michael. The Trustworthy Computing Security Development Lifecycle. http://msdn.microsoft.com/security/default.aspx?pull=/library /en-us/dnsecure/html/sdl.asp (2005).

[14] Sanjai Gupta, Md Faisal, Mohammed Hussain," SECURE SOFTWARE DEVELOPMENT PROCESS FOR EMBEDDED SYSTEMS CONTROL", International Journal of Engineering Sciences & Emerging Technologies, Dec. 2012., ISSN: 2231 – 6604, Volume 4, Issue 1, pp: 133-143 ©IJESET

[15] Abrahamsson, P., Warsta, J., Siponen, M.T. & Ronkainen, J., (2003), New directions on agile methods: A comparative analysis. International Conference on Software Engineering.

[16] Beznosov, Konstantin. eXtreme Security Engineering: On Employing XP Practices to Achieve 'Good Enough Security' without Defining It. http://konstantin.beznosov.net/professional/papers /eXtreme_Security_Engineering.html (2003).

[17] Mehrez Essafi, Lamia Labed, and Henda Ben Ghezala, "S2D-ProM: A Strategy Oriented Process Model for Secure Software Development", In Proc. of the 2nd International Conference on Software Engineering Advances (ICSEA'07), Cap Esterel, French Riviera, France, 2007, p. 24.

[18] OWASP Foundation, "OWASP CLASP v1.2 Comprehensive, Lightweight Application Security Process", OWASP. November 9, 2007.

[19] Li, W. and Chiueh, T., "Automated Format String Attack Prevention for Win32/X86 Binaries," In Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC'07), Miami, Florida, USA, Dec 2007, pages 398409

[20] Peine, H., "Rules of Thumb for Developing Secure Software: Analyzing and Consolidating Two Proposed Sets of Rules," In Proceedings of the 3rd International Conference on Availability, Reliability and Security (ARES'08), Barcelona, Spain, 2008, IEEE CS Press, pages 1204-1209.

[21] Saltzer, J. H., and Schroeder, M. D., "The Protection of Information in Computer Systems," Proceedings of the IEEE, IEEE Press, 1975, volume 63, number 9, pages 1278-1308.

[22] Viega, J. and McGraw, G., Building Secure Software, Addison Wesley, 2002.

[23] Howard, M. and LeBlanc, D., Writing Secure Code 2nd Edition, Microsoft Press, 2003.

[24] Khan, M. U. and Zulkernine, M., "On Selecting Appropriate Development Processes and Requirement Engineering Methods for Secure Software," In Proceedings of the 4th IEEE International Workshop on Security, Trust, and Privacy for Software Applications (STPSA 2009), Seattle, Washington, USA, 2009, IEEE CS Press, volume 2, pages 353-358.

[25] Khan, M. U. and Zulkernine, M., "Activity and Artifact Views of a Secure Software Development Process," In Proceedings of the International Workshop on Software Security Process (SSP'09), Vancouver, Canada, 2009, IEEE CS Press, volume 3, pages 399-404.

[26] Khan, M. U. and Zulkernine, M., "Quantifying Security in Secure Software Development Phases," In Proceedings of the 2nd IEEE International Workshop on Secure Software Engineering (IWSSE'08), Turku, Finland, 2008, IEEE CS Press, pages 955-960.

[27] D.P. Gilliam, T.L. Wolfe, J.S. Sherif, and M. Bishop, "Software Security Checklist for the Software Life Cycle," In Proc. of the 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03), Linz, Austria, IEEE CS Press, 2003, pp. 243-248.

[28] D. Gilliam, J. Powell, E. Haugh, and M. Bishop, "Addressing Software Security Risk and Mitigations in the Life Cycle," In Proc. of the 28th Annual NASA Goddard Software Engineering Workshop (SEW'03), Greenbelt, Maryland, USA, 2003, pp. 201-206.

[29] G. McGraw, "Testing for Security During Development: Why we should Scrap Penetrate-and-Patch," IEEE Aerospace and Electronic Systems, IEEE CS Press, 1998, vol. 13, no. 4, pp. 13-15.

[30] L. Futcher and R.v. Solms, "SecSDM: A Model for Integrating Security into the Software Development Life Cycle," In IFIP International Federation for Information Processing, Volume 237, Proc. of the 5th World Conference on Information Security Education, Springer, 2007, pp. 41-48

[31] I. Flechais, M.A. Sasse, and S.M.V. Hales, "Bringing Security Home: A Process for Developing Secure and Usable Systems," In Proc. of the New Security Paradigms Workshop (NSPW'07), Ascona, Switzerland, ACM Press, 2003, pp. 49-57.

[32] J. Gregoire, K. Buyens, B. De Win, R. Scandariato, and W. Joosen, "On the Secure Software Development Process: CLASP and SDL Compared," In Proc. of the 3rd International Workshop on Software Engineering for Secure Systems (SESS'07), Minneapolis, Minnesota, USA, IEEE CS Press, 2007, pp. 1-1.

[33] Hall, Anthony & Chapman, Roderick. "Correctness by Construction: Developing a Commercial Secure System." IEEE Software 19, 1 (January/February 2002): 18–25.

[34] Ross, Philip E. "The Exterminators: A Small British Firm Shows That Software Bugs Aren't Inevitable." IEEE Spectrum 42, 9 (September 2005): 36–41.

[35] S.T. Eckmann, G. Vigna, and R.A. Kemmerer, "STATL: An Attack Language for State-Based Intrusion Detection," Journal of Computer Security, IOS Press, Amsterdam, 2002, vol. 10, no. 1/2, pp. 71-104.

[36] Md Swawibe Ul Alam," Survey of Specification Languages for Cloud Security",

[37] T. Lodderstedt, D.A. Basin, and J. Doser, "SecureUML: A UML-Based Modeling Language for Model Driven Security," In Proc. of the 5th International Conference on the Unified Modeling Language (UML'02), Dresden, Germany, Springer, 2002, LNCS 2460/2002, pp. 426-441.

[38] M. Hussein and M. Zulkernine, "UMLintr: a UML profile for specifying intrusions," In Proceedings of the 13th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, Potsdam, Germany, IEEE CS Press, 2006, pp. 279–286.

[39] Microsoft. ASML. https://www.microsoft.com/en-us/research/project/ asml-abstract-state-machine-language/, 2000. [Online; accessed 06-March-2017].

[40] M. Raihan and M. Zulkernine. Asmlsec: An extension of abstract state machine language for attack scenario specification. In Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on, pages 775–782. IEEE, 2007.

[41] M. Graves and M. Zulkernine, "Bridging the Gap: Software Specification Meets Intrusion Detector," In Proc. of the 4th Annual Conference on Privacy, Security and Trust (PST'06), Ontario, Canada, pp. 265-274.

[42] Snort, www.snort.org. Last Accessed March 2009

[43] Michael Felderer, Matthias Bu¨chler, Martin Johns, Achim D. Brucker, Ruth Breu, Alexander Pretschner, "Security Testing: A Survey", Survey. In: Memon, A., (ed.) Advances in Computers, Volume 101. Elsevier , Cambridge, MA, USA , pp. 1-51. ISBN 9780128051580

[44] M. Gallaher and B. Kropp. The economic impacts of inadequate infrastructure for software testing. Technical Report Planning Report 02-03, National Institute of Standards & Technology, May 2002.

[45] G. Buehrer, B. W. Weide, and P. A. G. Sivilotti. Using parse tree validation to prevent sql injection attacks. In Proceedings of the 5th International Workshop on Software Engineering and Middleware, SEM '05, pages 106–113, New York, NY, USA, 2005. ACM

[46] K. Scarfone, M. Souppaya, A. Cody, and A. Orebaugh. Technical Guide to Information Security Testing and Assessment. Special Publication 800-115, National Institute of Standards and Technology (NIST), 2008.

[47] B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of unix utilities. Commun. ACM, 33(12):32–44, Dec. 1990.

[48] M. Felderer and E. Fourneret. A systematic classification of security regression testing approaches. International Journal on Software Tools for Technology Transfer, pages 1–15, 2015.

[49] S. Yoo and M. Harman. Regression testing minimisation, selection and prioritisation: A survey. Software Testing, Verification, and Reliability, 1(1):121–141, 2010.

[50] Abdullah Saad AL-Malaise AL-Ghamdi, "A Survey on Software Security Testing Techniques", International Journal of Computer Science and Telecommunications [Volume 4, Issue 4, April 2013]



**DOI: 10.24297/ijct.v16i7.6467**