

xSt. Francis Institute of Technology
(An Autonomous Institution)
AICTE Approved | Affiliated to University of Mumbai

A+ Grade by NAAC: CMPN, EXTC, INFT NBA Accredited: ISO 9001:2015 Certified

Department of Information Technology

A.Y. 2025-2026
Class: BE-IT A/B, Semester: VII
Subject: Secure Application Development Lab

Student Name: **Keith Fernandes**

Student Roll No: **25**

Experiment – 2 : Study of Secure Software Development Life Cycle (SDLC)

Aim: To study secure Software Development Life Cycle (SDLC).

Objectives: After study of this experiment, the student will be able to

- Understand different steps of SDLC .
- Identify and learn different standards of cyber security.

Lab objective mapped: ITL703.2 : To **understand** the methodologies and standards for developing secure code

Prerequisite: Basic Knowledge of different stages SDLC

Requirements: Personal Computer, Windows operating system browser, Internet Connection etc.

Pre-Experiment Theory:

What is Secure SDLC and Why is it important ?

Secure System Development Life Cycle (SecSDLC) is defined as the set of procedures that are executed in a sequence in the Software Development Life Cycle (SDLC). It is designed such that it can help developers to create software and applications in a way that reduces the security risks at later stages significantly from the start. The Secure System Development Life Cycle (SecSDLC) is similar to Software Development Life Cycle (SDLC), but they differ in terms of the activities that are carried out in each phase of the cycle. SecSDLC eliminates security vulnerabilities. Its process involves identification of certain threats and the risks they impose on a system as well as the needed implementation of security controls to counter, remove and manage the risks involved. Whereas, in the SDLC process, the focus is mainly on the designs and implementations of an information system.

Phases involved in SecSDLC are:

- **System Investigation:** This process is started by the officials/directives working at the top-level management in the organization. The objectives and goals of the project are considered priority in order to execute this process. An Information Security Policy is defined which contains the descriptions of

security applications and programs installed along with their implementations in organization's system.

- **System Analysis:** In this phase, detailed document analysis of the documents from the System Investigation phase are done. Already existing security policies, applications and software are analyzed in order to check for different flaws and vulnerabilities in the system. Upcoming threat possibilities are also analyzed. Risk management comes under this process only.
- **Logical Design:** The Logical Design phase deals with the development of tools and following blueprints that are involved in various information security policies, their applications and software. Backup and recovery policies are also drafted in order to prevent future losses. In case of any disaster, the steps to take in business are also planned. The decision to outsource the company project is decided in this phase. It is analyzed whether the project can be completed in the company itself or it needs to be sent to another company for the specific task.
- **Physical Design:** The technical teams acquire the tools and blueprints needed for the implementation of the software and application of the system security. During this phase, different solutions are investigated for any unforeseen issues, which may be encountered in the future. They are analyzed and written down in order to cover most of the vulnerabilities that were missed during the analysis phase.
- **Implementation:** The solution decided in earlier phases is made final whether the project is in-house or outsourced. The proper documentation is provided of the product in order to meet the requirements specified for the project to be met. Implementation and integration process of the project are carried out with the help of various teams aggressively testing whether the product meets the system requirements specified in the system documentation.
- **Maintenance:** After the implementation of the security program, it must be ensured that it is functioning properly and is managed accordingly. The security program must be kept up to date accordingly in order to counter new threats that can be left unseen at the time of design.

Procedure:

Study the case study from references and discuss the proposed model to integrate security in SDLC.

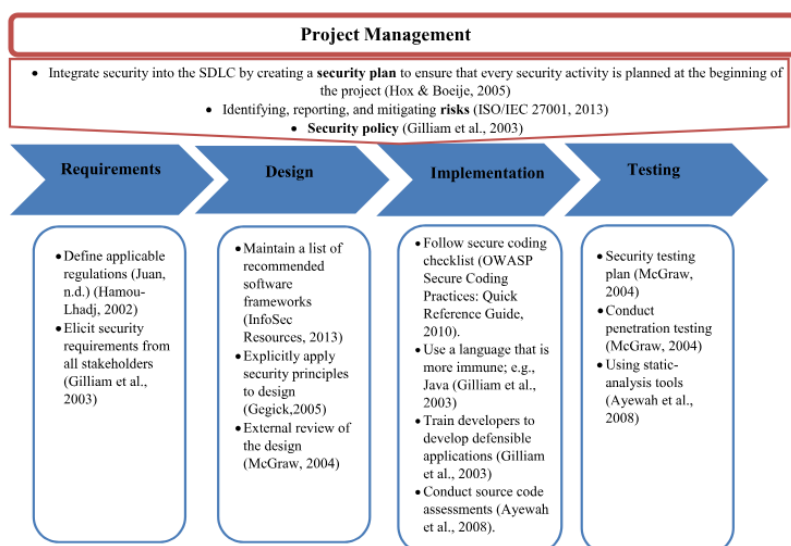


Figure 1. Preliminary research model. SDLC, software development life cycles.

The diagram presents a detailed model of integrating security into the Software Development Life Cycle (SDLC) through a structured and proactive approach. It begins with the **project management layer**, which emphasizes the importance of creating a security plan at the outset of the project. This includes defining a security policy, identifying and mitigating risks (as per ISO/IEC 27001), and ensuring every phase of the SDLC aligns with security goals.



In the **Requirements** phase, applicable regulations are defined (e.g., data protection laws), and security requirements are collected from all stakeholders to ensure compliance and clarity from the beginning. The **Design** phase focuses on maintaining a list of recommended secure frameworks, explicitly applying security principles (such as least privilege or defense in depth), and conducting an external review of the system architecture to catch potential vulnerabilities early.

During the **Implementation** phase, developers follow secure coding practices using checklists like those from OWASP, prefer secure languages like Java, and undergo training to build secure and defensive code. Source code is also reviewed and assessed using static code analysis tools to catch errors before they become threats. Finally, in the **Testing** phase, security is validated through a well-defined testing plan, penetration testing, and the use of static analysis tools to detect vulnerabilities that might have been missed during development.

This model supports a “**shift-left**” approach by embedding security early and continuously across the SDLC. It ensures that the final product is not just functionally sound but also resistant to cyber threats, compliant with regulations, and developed with a strong focus on secure practices at every stage

Compare Software Development Life Cycle (SDLC) and Secure SDLC

Aspect	SDLC	Secure SDLC
Focus	Functionality, performance, and usability	Security along with functionality
Security Integration	Security often considered only in testing	Security is integrated at every stage
Risk Assessment	Performed after major design decisions	Performed early and iteratively
Tools Used	IDEs, Testing frameworks	SAST, DAST, threat modeling tools, secure coding tools

Output	Working software	Secure and robust software
Cost of Fixing Bugs	High, if found late	Reduced, due to early detection
Diagram	Software Development Life Cycle (SDLC) 	Secure Software Development Life Cycle (SSDLC) 

Describe how secure coding can be incorporated into the software

Secure coding is the practice of writing computer programs in a way that guards against the introduction of security vulnerabilities. Incorporating secure coding into the software development process ensures that applications are resistant to cyber-attacks, data breaches, and unauthorized access. It is not a one-time task but a continuous, integrated part of every stage of the Software Development Life Cycle (SDLC).

1. Define and Enforce Secure Coding Guidelines

Organizations should create or adopt standard secure coding practices such as the OWASP Secure Coding Guidelines, SEI CERT Coding Standards, or language-specific best practices. These guidelines help developers write code that avoids common vulnerabilities like SQL injection, cross-site scripting (XSS), buffer overflows, etc.

2. Integrate Security in the Development Environment

Secure coding begins right in the IDE: Use plugins or linters (e.g., ESLint for JavaScript, Pylint for Python) that flag insecure patterns in real time. Enable automated security scanning tools in the build environment that scan code for vulnerabilities before it's committed or deployed.

3. Conduct Regular Developer Training

One of the most effective ways to promote secure coding is by educating developers. Regular training and workshops on topics like the OWASP Top 10 vulnerabilities, secure API usage, cryptography basics, and secure design patterns ensure that developers stay updated with evolving security threats.

4. Perform Code Reviews with a Security Focus

During peer code reviews, reviewers should not only focus on performance and functionality but also examine:

- Input validation and sanitization
- Proper error handling and logging
- Avoidance of hardcoded secrets or credentials
- Use of secure libraries and functions

Checklists based on secure coding standards can be used to guide the review process.

5. Use Security Testing Tools

Secure coding isn't just about writing secure code—it also includes verifying that code is secure:

1. **Static Application Security Testing (SAST):** Tools like SonarQube, Fortify, or Snyk scan source code to identify vulnerabilities before execution.
2. **Dynamic Application Security Testing (DAST):** Tools like OWASP ZAP or Burp Suite analyze running applications for runtime vulnerabilities.
3. **Interactive Application Security Testing (IAST):** Combines the best of both SAST and DAST during actual execution of the app.

6. Secure Dependencies and Libraries

Third-party libraries and open-source packages may contain known vulnerabilities. Developers should:

1. Use **Software Composition Analysis (SCA)** tools (e.g., Snyk, OWASP Dependency-Check) to scan dependencies.
2. Avoid using outdated or unmaintained packages.
3. Regularly monitor vulnerability databases (e.g., CVE) for reported issues.

7. Apply DevSecOps Principles

Security should be part of the CI/CD pipeline. By integrating security checks (like secret scanning, dependency checks, and code analysis) into build and deployment workflows, developers are alerted to issues early. This ensures secure coding remains continuous and automated.

8. Secure Error Handling and Logging

Secure coding also involves handling exceptions carefully:

1. Avoid exposing sensitive information in error messages (e.g., stack traces or database queries).
2. Log events securely without logging sensitive data like passwords or tokens.
3. Use secure logging libraries and formats to ensure log integrity.

Post-Experiments Exercise:

Extended Theory: Nil

Post Experimental Exercise:

Questions:

- List the major types of coding errors and their root causes.
- Describe good software development practices and explain how they affect application security.

Conclusion:

- Write what was performed in the experiment.
- Write the significance of the topic studied in the experiment.

References:

- Case study: <https://onlinelibrary.wiley.com/doi/epdf/10.1002/sec.1700>
- <https://snyk.io/learn/secure-sdlc/>
- <https://www.checkpoint.com/cyber-hub/cloud-security/what-is-secure-coding/>
- <https://snyk.io/learn/secure-coding-practices/>