Security in SDLC – Secure Software Development Lifecycle – SSDLC

Authors: Theodor Adam, Florin Andrei, Larisa Gabudeanu, Victor Rotaru

Copy Editor: Alexandru Mircea Rotaru

The Software Development Lifecycle (SDLC) is a framework developed to design, build, test and deploy high quality software that meets or exceeds customer expectation and reaches completion within estimated time and budget.

The SDLC framework includes the following steps:

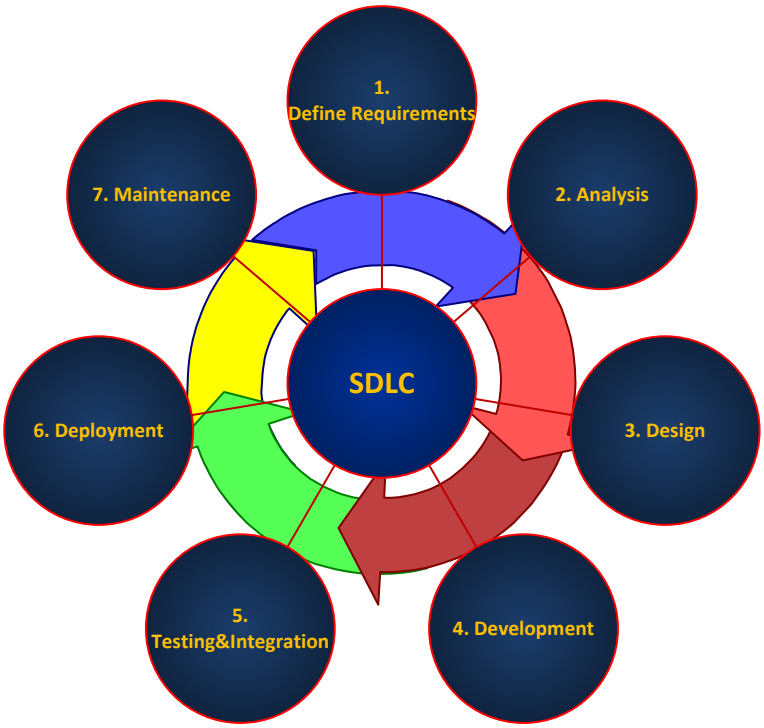

Figure 1: The SDLC Framework

1. **Define Requirements** – The requirements definition phase shapes the major functions and features of the intended application or system. It should be considered the following:
   a. Business requirements;
   b. Operational requirements;
   c. Security requirements;
   d. Architecture requirements;

The validation of the collected and defined requirements with the stakeholders of the intended application or system is very important.

2. **Analysis** – The requirements analysis will produce the specification for the intended application or system. Using the specification and the defined requirements it will be easy to shape the acceptance criteria you will use to assess the completeness of the application or system before deployment. This is the phase where the feasibility study is done.
3. **Design** – The requirements and specification from the previous phase will help in this stage for the creation of the conceptual design of the intended application or system. The workflows together with the detailed software architecture are created during this phase and the applied standards are defined.
4. **Development** – This is the phase where the design documentation developed in the previous phase is converted into an application or system that should meet the requirements and specification.
5. **Testing & Integration -** the testing team (or the quality assurance team) uses different frameworks to execute unit tests, functionality testing, system integration testing, interoperability testing as well as user acceptance testing in order to ensure that the code is clean (bug free) and the requirements are met.
6. **Deployment** – the tested application or system is moved to production. This phase includes the work necessary to deploy the final solution into the target production environment. Also, you need to create some documentation like: creating guides for installation, system operations, system administration, and end-user functionality. For complex software projects, you need to create a detailed plan for implementing the application or the system across the organization.
7. **Maintenance** – this is the final stage of the software development lifecycle and includes maintenance and regular updates. Through maintenance, the deployed application or system is fine-tuned according to the user's feedback about its performance. Also, periodically, the application or system is enhanced and upgraded in order to comply with the end user needs.

As it was defined, SDLC is a framework focused on software delivery, but can be enriched with best practices in order to improve the quality of its results. **Secure SDLC**

(SSDLC) is a collection of best practices enriching SDLC framework to make secure the software developed through SDLC.

Next, we will pass through SDLC presented above and add to it the common best practices used to make SDLC secure, or to transform SDLC in SSDLC.

One of the most important steps for both SDLC and SSDLC is the **Requirements Definition phase**. In order to shape properly the security requirements, you need to perform a risk assessment and use it as reference, to define these security requirements. Moreover, it is very important to identify any security considerations for business requirements, operations requirements and architecture requirements. There are some security aspects you should consider, regardless the size of your project:

- Access control. Identification and Authentication;
- Data security. Security of data in transit (communications protection). Security of data at rest (information protection and integrity);
- Media protection;
- Physical protection;
- System protection and integrity;

A good guideline for all these security aspects is NIST 800-171 Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations, https://csrc.nist.gov/publications/detail/sp/800-171/rev-2/final.

Other very important aspects you should consider from the Requirement Definition phase are the legal and regulatory constraints. For example laws like GDPR, Sarbanes-Oxley and HIPAA put constraints on data collection, processing and dissemination. This may affect how you will design your security for your application or system. Therefore, for each of your requirements topic you develop, you should define a dedicated topic of security constraints that apply and present them in the Security Requirements.

The **Analysis phase** it is also a validation step for the collected requirements and constraints during the Requirements Definition phase. If the feasibility study you perform during this phase reveals some compliance or regulatory issues because of a poor security requirements collection, the project should return to the Requirements Definition phase. The specification developed at this phase should embed security specification and the acceptance criteria you design should address all security requirements collected during Requirement Definition phase.

During the **Design phase**, you need to perform a Risk Analysis, or a Threat Modelling and at the end to review the design in order to ensure that all the security requirements and specifications are enough to protect against the threats identified in the studied threat or risk

models. The solution architecture should embed all security requirements and specification. The overall technical solution should assure end-to-end security specification as defined.

At this phase, a Security Testing Plan should be developed based on the threats identified during Threat Modelling or Risk Analysis exercise.

For the **Development phase** the goal is to make sure that the written code is clean and well written. Your organization should adhere to rigid coding standards. If code runs or plays a part in gathering information for mission-critical applications, it is too important to leave it to chance and should be controlled by coding standards that are constantly kept up-to date. Obtaining the right standards and keeping them current with the latest best practices should be a top priority for organizations with a software development team.

For the **Testing and Integration phase** a security test plan should be defined prior starting the Testing and Integration phase. The security test plan can be derived directly from the results of threat modelling. Unit testing and Integration testing are the main testing tasks performed in this phase, but are not enough to ensure the security of the developed application or system.

A code review is required in order to ensure that there is no issue with the intended application or system and that coding standards are respected and applied correctly. Ensuring continuous code quality, both in the development and maintenance phases, reduces considerably the costs and risks of security and reliability issues in software as well. The code review can be either manual or automated using technologies such as static application security testing (SAST). These open-source components are usually checked using Software Composition Analysis (SCA) tools.

The key objectives of the code review are:

- The design goals are being met;
- The security objective are being met;
- The implementation is robust;
- The coding standards are respected and applied correctly;

Keep in mind that software is developed by humans, and humans make mistakes. The later the issues are discovered in the SDLC, the more difficult they are to correct and the more work that may need to be redone as a result. Static code analysis tools are not capable of detecting every potential vulnerability within an application because some vulnerabilities are only apparent at runtime, and static code analysis tools do not execute the code that they are examining. For the runtime vulnerabilities, penetration testing will reveal them and this should be done before moving the application or system to production.

The **Deployment phase** should start with a strong security testing before moving the application or system to production. A reliable security testing phase should contain not only vulnerability assessment or penetration testing scenarios, but also a testing phase of the incident response, especially if the application or system you are planning to test is mission-critical.

Platform security cannot be ignored, for while the application itself might be secure, the platform it operates on might have exploitable flaws. Therefore, platforms need to be made secure by taking appropriate measures like turning off unwanted services, running the machines on the least privilege principle, and making sure there are security safeguards such as IDS, and firewalls.

The overall security testing of the entire system should be performed in order to ensure that the developed application or system together with the platform it operates are secured and can allow users to use it. Penetration testing will identify the vulnerabilities and will allow the implementation team to fix them in order to ensure a secure configuration within production environment.

Keep in mind that once an application or system is deployed in production, security vulnerabilities become exponentially costlier to fix. Regardless of the sophistication of the software and thorough testing there will always be glitches and bugs.

There are also on-going tasks during the **Maintenance phase** because security is an on-going process and updates, patches and enhancements to the application code are constantly required. It is a cycle that repeats itself, but security, even at the time of these modifications, must always be in focus. Performing penetration testing after implementing important updates for the application or supporting platform should be part of your Operational Assurance process.

Keep in mind that at the end of its lifecycle, a software or a platform must be disposed properly in order to maintain the same level of security for your operational environment. Sometimes, the disposal might be performed through a dedicated project where all aspects are considered in order to minimize the risks.

Conclusion

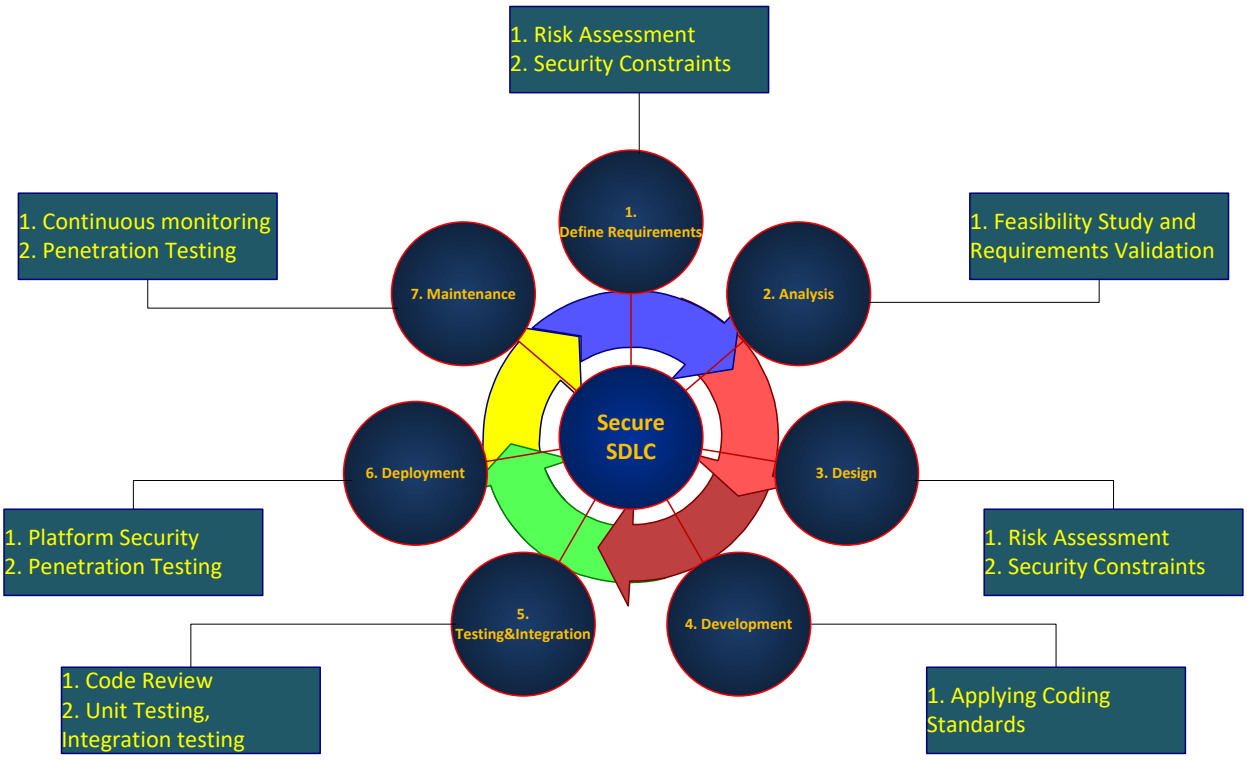

Figure 2: The SSDLC Framework

Above, the Figure 2 presents SDLC with the main best practices needed to make it Secure SDLC or SSDLC.

The continuously evolving threats demand organizations to improve their security posture and therefore, a framework like SDLC must be enriched with the best practices required to make it become SSDLC and deliver secure software.

As presented in this article, secure development lifecycle or secure SDLC helps developers and organizations plan, create, deploy and maintain secure software because security becomes part of the software development process in each of its stages and controls that the definition, analysis, design, development, testing, deployment and maintenance phases deliver secure results.

Mature implementations of SDLC already had in place the best practices to make it secure before being defined as Secure SDLC because in time, it became a necessity to embed security best practices in their Software Development Lifecycle and they understood that security must be everywhere. It should begin at project inception and be on the mind of every engineer during requirements analysis, design, coding, testing and deployment. This is the only way that security can be reliably improved in every application or system you build to deliver secure software and solutions.