SORSOGON STATE UNIVERSITY
BULAN CAMPUS
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY
Bachelor of Science in Information Technology

ICT

Module 2

- *SECURITY SYSTEM DEVELOPMENT LIFECYCLE*

Learner's Reference Number (LRN)

Name of Student Strand/Year Address

Contact No.

BSIT 4

RICHARD G. RABULAN, MIT
INSTRUCTOR II

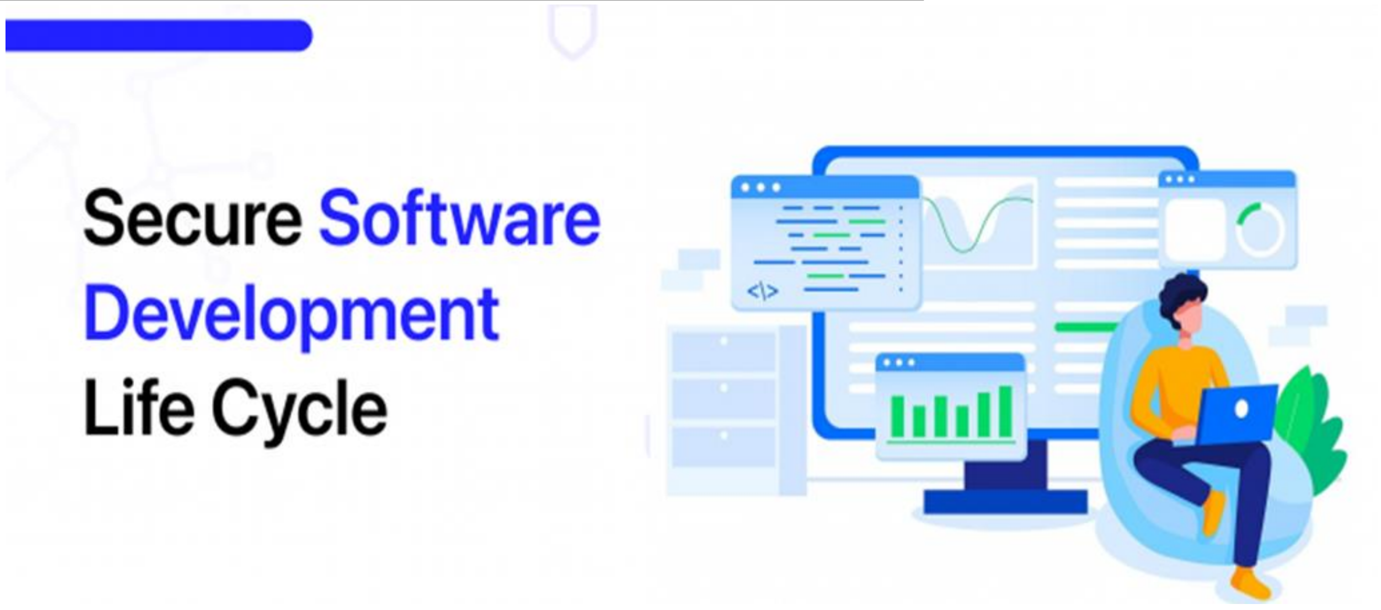
RICHARD RABULAN | 0977-357-1407 | richard.rabulan@sorsu.edu.ph

HOW TO USE THIS MODULE

Before starting the module, I want you to set aside other tasks that will disturb you while enjoying the lessons. Read the simple instructions below to successfully enjoy the objectives of this kit. Have fun!

1. Follow carefully all the contents and instructions indicated in every page of this module.
2. Write on your notebook the concepts about the lessons. Writing enhances learning that is important to develop and keep in mind.
3. Perform all the provided activities in the module.
4. Let your facilitator/guardian assess your answers using the answer key card.
5. Analyze conceptually the posttest and apply what you have learned.
6. Enjoy studying!

Lesson 1 – Security System Development Life Cycle



As technology evolving every year, so does the security needs of the company. For this module, we are going to tackle about the following questions met in SecSDLC.

- What is considered a secure software development life cycle – SSDLC?
- Why should you employ a secure software development life cycle?
- Benefits of secure software development life cycle
- How do you ensure a secure software development life cycle?
- What Is Considered A Secure Software Development Life Cycle?

Security System Development Life Cycle (SSDLC)

The SSDLC is a useful framework for managing the development, maintenance, and retirement of an organization's information security systems. It helps to ensure that security systems meet the needs of the organization and are developed in a structured and controlled manner. This can help organizations to protect their sensitive information, maintain compliance with relevant regulations, and keep their data and systems safe from cyber threats.

Simply put:

"Secure Software Development Life Cycle (S-SDLC) is a development approach in which developers must always be mindful of possible security risks in all development life cycle phases; then incorporate security artifacts from planning to operation maintenance."

Security System Development Life Cycle (SecSDLC) is defined as the set of procedures that are executed in a sequence in the software development 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 Security 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 OF SECURITY SYSTEM DEVELOPMENT LIFE CYCLE

The **Security System Development Life Cycle (SSDLC)** is a framework used to manage the development, maintenance, and retirement of an organization's information security systems. The SSDLC is a cyclical process that includes the following phases:

- **System Investigation:** During this phase, the organization identifies its information security needs and develops a plan to meet those needs. This may include identifying potential security risks and vulnerabilities, and determining the appropriate controls to mitigate those risks. 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 priorly 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:** During this phase, the organization analyzes its information security needs in more detail and develops a detailed security requirements specification. 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.

- **Design:** During this phase, the organization designs the security system to meet the requirements developed in the previous phase. This may include selecting and configuring security controls, such as firewalls, intrusion detection systems, and encryption. There are Two types of Design: Logical Design and Physical Design.

  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.

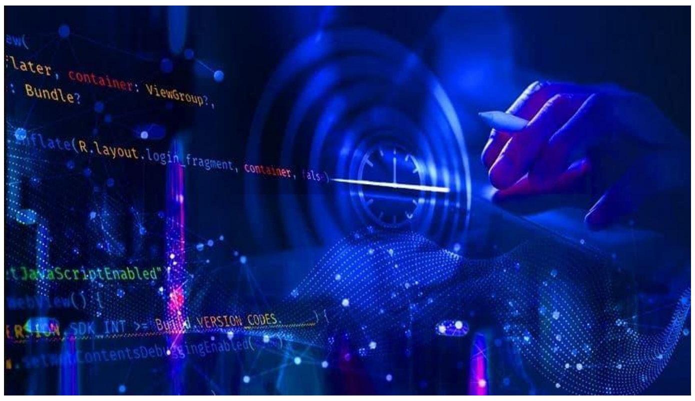
Physical Design: The technical teams acquire the tools and blueprints needed for the implementation of the software and application of the system security. During this phase, different solutions are investigated for any unforeseen issues which may be encountered in the future. They are analyzed and written down in order to cover most of the vulnerabilities that were missed during the analysis phase.

- **Implementation:** During this phase, the organization develops, tests, and deploys the security system. 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 security system has been deployed, it enters the maintenance phase, where it is updated, maintained, and tweaked to meet the changing needs of the organization. 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.

- **Retirement:** Eventually, the security system will reach the end of its useful life and will need to be retired. During this phase, the organization will plan for the replacement of the system, and ensure that data stored in it is properly preserved.

Why Should You Employ A Secure Software Development Life Cycle Important?



S-SDLC framework has become increasingly important due to the rise in security risks during a software project and the low response of the traditional software development life cycle.

Back to a couple of decades ago, the primary approach to software development is the Waterfall model. The application's life cycle is linear-sequential, meaning each phase must be finished before the next step begins. No overlapping is allowed.
All requirements, including security, are highly specified in the first place and laid the groundwork for the rest phases.
However, security risks are complicated and challenging to predict clearly in advance. Let alone the current world has many interconnections that criminals or nefarious users target at any time. When the security issues are addressed in the testing phase of the Waterfall model, the potential vulnerabilities might have been exploited for quite a time. Then, it requires significant effort and resources to fix them right.
Nowadays, many developers have switched to another development approach: Agile. This approach allows splitting up an extensive software application into multiple mini-releases. Agile offers improved flexibility and frequent testing of functions and security in all phases. Issues, if any, will be predicted and detected soon.

Benefits Of Secure Software Development Life Cycle

Secure software development is inevitable and vital, as mentioned above. In return, secure SDLC also significantly benefits the success of software applications.

Here are the benefits of Secure Software Development Life Cycle:

- ***<u>Enhance the responsibilities of the development team as a whole</u>***

It is critical to make security an ongoing concern in any software development. Since security requirements are applied to all phases, they keep stakeholders of the software project on the same page about security issues. They have to take part in and have responsibilities for the application to be protected without delay.

- ***<u>Reduce cost and effort in fixing security issues</u>***

Back in the time when the development teams often implemented security-related tasks during the testing phase, it was very late or even not at all to discover issues. Unfortunately, IBM confirmed that it took 15 times more to fix bugs detected while testing than those found during the design phase. Thus, the earlier the team can maintain security tasks during the software development life cycle, the cheaper they have to pay.

- ***<u>Build trust in the client and customers</u>***

According to the S-SDLC, all software project stakeholders are included in the security considerations. Your client and end-users, therefore, have a good impression of your business. In contrast, if your software is not secure enough, you put their information at risk of attacks.

How To Ensure Secure Software Development Life Cycle?

As advantageous and critical as the secure software development life cycle, all development teams strictly follow the best rules in their activities.
Build a secure software development policy

Since security risks are hard to predict and prevent, development teams must create a security baseline. It is a set of guidelines to detail the security procedures and practices to reduce vulnerabilities during a software life cycle.

For instance, the policy includes instructions on how to view, assess, and demonstrate security in specific phases. There is also a discussion on the risk management approaches and necessary processes to protect your software application (separation of environments for developing, testing, and operating.)

The policy will work as a vehicle for managers of the development team to

implement a secure software's strategic methodology. Once the policy is applied, the engineering team must also create a gap analysis to contrast the practical activities in the software development life cycle against the baseline. If a gap exists, it needs addressing in the early phases to reduce the efforts and expense.
You must also build a checklist to closely and frequently monitor the policy and keep them updated and enhanced. If possible, some development organizations have security experts evaluate and create a plan for a secure software development life cycle.

Continuously train the developing teams on the policy

Organize training sessions for the teams, including developers, architects, designers, testers, etc., to update them on the above policy. Each team member must understand their duties and strictly perform their daily tasks. In addition, security awareness also targets other stakeholders involved in the project and the organization.

Most sessions are firstly traditional theoretical classrooms. They help equip the stakeholders with security software development life cycle policy. However, it is recommended to narrow the critical security points in slides or records in advance so that attendants can have brief yet practical knowledge. Examples of common security risks and solutions need to be included.

Next, conduct exercise training to assess how well the stakeholders intertwine the theories with their practice environments. There must be experienced mentors who can on-job review and instruct the trainees during the cycle.

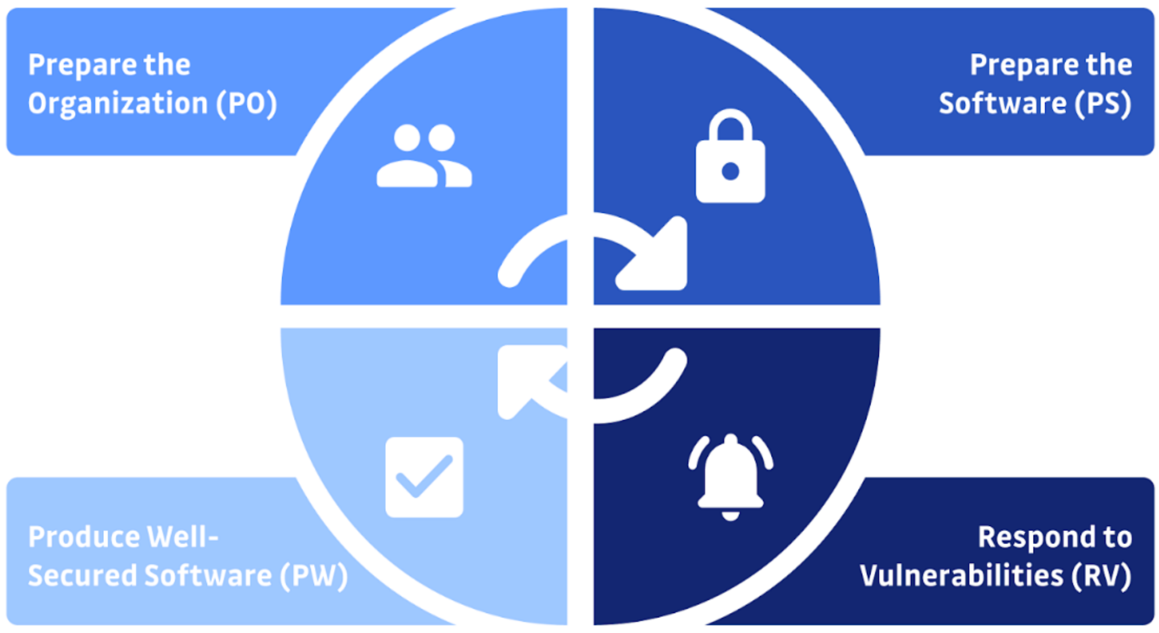
Pick up an S-SDLC framework to guarantee consistency

Since secure software has become a norm, experts have built many frameworks to help enhance security. Then, the development team can choose a specific framework to align their practices throughout the software development life cycle. Of which, we suggest using the framework of NIST (SSDF). This framework includes a set of sound, fundamental, and secure practices for software development, referring to the S-SDLC of OWASP, BSA, and SAFECode. With those practices, organizations can significantly reduce vulnerabilities and the potential exploitation of undetected vulnerabilities and address the root reasons for security issues to prevent their recurrences – following four stages.

- Firstly, prepare the organization (PO) by ensuring that its people, technology, and process are prepared and enhanced towards secure software development, either at the project or organizational levels.
- Secondly, protect the software (PS), including all components, from unauthorized and tampering access.
- Next, produce well-secured software (PW), so there are a minimum number of vulnerabilities once releasing the software.
- Finally, respond to vulnerabilities (PV) appropriately to address the root causes and prevent the vulnerabilities' recurrence in the future.

Every practice of the SSDF is outcome-based and contains tasks, implementation examples, and references.
• Practice: The practice is a brief statement with an explanation and its unique identifier to determine the definition and benefits of the practice.
• Task: Individual action(s) are required to complete a practice.
• Implementation example: The example indicates a given scenario helpful to demonstrate the practice.
• Reference: It is a secure practice document and how it maps to a particular task(s).

Apply the framework to the software development


Prepare the Organization (PO)
Prepare the Software (PS)
Produce Well-Secured Software (PW)
Respond to Vulnerabilities (RV)

We break down the four stages for reference if you use the NIST (SSDF) framework.

PO: practices, tasks, and examples

The preparation involves defining internal security requirements (policies and risk management approaches) and external requirements (laws, regulations, etc.)

Next, assign the software requirements roles and tasks so the development team can prepare role-specific training sessions and supporting tools. Tasks include information on identifying, communicating, and maintaining all security requirements throughout the life cycle.
Finally, define benchmarks to take notes of security standards achievements.

Examples of preparing the organization:

• Defining software development specifications such as architecture requirements and coding practices for developers.
• Agreeing on reviewing and updating requirements yearly or mainly after incidents.
• Assigning roles and training plans and preparing to update roles at any time.
• Specifying toolchain categories and relevant tools, then incorporating automation for the operation and management of the toolchain.
• Building an audit trail to keep track of SDLC-related actions.
• Determining KPI (key performance indicators), using an automatic toolchain to collect feedback, and reviewing and documenting a security checklist to achieve defined standards.

PS: practices, tasks, and examples

It is paramount to protect code and enhance the integrity of software from the first phase until it reaches the end customers. The developers must safeguard the code from tampering and unauthorized access, following the least-privilege principle.

Examples of protecting the software practices:

- Storing the code in restricted-access and secure repositories
- Using version control to track all changes in code
- Publishing cryptographic hashes used for released software
- Only using trusted authorities for singing code

PW: practices, tasks, and examples

When producing well-secured software, the teams go through many steps involving various practices and actors.

The application is under review to align security requirements before the team evaluates third parties' compliance with these requirements. Within time, the team reviews and tests all code using automated and manual means. Meanwhile, developers must follow best practices in writing codes and configuring the developing process. In the end, configure secure default settings to protect the software out of the box.

Examples of producing well-secured software:

- Educating the team about secure developing best practices and risk assessment techniques
- Lesson-learning from previous releases to address as many potential risks and security requirements as possible
- Stating requirements of secure software development life cycle in 3rd-party contracts and policies to manage their risks
- Only developing the software in environments mandating safe coding practices
- Implementing peer reviews, penetration testing, and dynamic/ static analysis testing to scan all underlying vulnerabilities, then documenting results
- Establishing a repository of reusable trusted developing components
- Testing all approved default configurations of security

RV: practices, tasks, and examples

Vulnerabilities are still possible even if the development team strictly follows the secure software development life cycle.
When there is a vulnerability, it is essential to fix the issue and gather data to prevent future recurrence. The vulnerability is prioritized to reduce the window of threat.

Examples of responding to vulnerabilities:

- Building reporting and response program to a vulnerability
- Implementing automation to scan vulnerable data and code analysis
- Predicting the impact of the vulnerable and preparing resources to fix the issue
- Detecting the root causes and documenting the lesson-learn for future detection and prevention of recurrence

A Summary Of The S-SDLC Best Practices

The framework above covers almost all secure software development life cycle best practices. We want to summarize them and include other techniques for you to take away:

- Think about secure software development from the beginning
- Create a transparent policy and keep all members frequently educated and updated on that policy
- Employ a consistent framework for secure software development, for instance, the NIST's SSDF)
- Develop the software following the best practices and aligning to security requirements
- Secure the code integrity against any possible tampering and strictly regulate all contacts with the code
- Apply an early and frequent review and test on the code to examine any flaws and catch vulnerabilities
- Get prepared to mitigate the detected vulnerabilities in real-time and update the software to narrow down the window of exploitation attempts
- Add secure default settings to the software's function list
- Use action checklists to keep track of security policies and procedures within periodic intervals: weekly or monthly
- Stay proactive and agile on best practices, so the teams can stay ahead of what is coming and be well-prepared

Final Words

We know that applying the above practices for a secure software development life cycle requires a serious investment of efforts and resources. However, it will all begin by thinking with a security-first approach. Do not hesitate to note down the practices, and let us know if you have any concerns.

Common Security Risks in Software Development Life Cycle (SDLC)

Security risks in software development life cycle phases

Important as the Secure SDLC is, many security risks in Software Development Life Cycle occur without being foreseen due to the complex telecommunications, hacking activities through power systems, underlying internet-ware applications, etc.

It would be helpful if you keep yourself updated on the common risks of each phase and get prepared.

Risks in Planning Phase

Planning is the first stone for software development, in general, and security practices, in particular.

Yet, inadequate security training and awareness is a popularly overlooked risk in this phase. Imagine the supervisors do not appreciate the value of security and spend less on security activities; the employees will also ignore security practices in their tasks and actions.

In addition, there might be no specific framework for risk management stated during the planning. In the subsequent phases, when the risks happen, the development team does not know how to resolve them effectively. The delay in tackling risks results in high costs and a waste of resources.

Risks in Requirements Phase

We are trying to gather as much information as possible, and extensive data are stored or documented for further reference in the development and testing phases. You had better be careful against external security risks such as viruses, hacking, denial of services or power loss or unsystematic risks such as data loss, human error, or inside attacks.

Another security risk during the requirements phase is the shortage of security requirements which indicates what the software should not do. This is common since many development teams only spend time on functional requirements.

Risks in Designing Phase

As mentioned above, data exposure due to external or unsystematic risks is also among the underlying security issues during the designing phase.
Moreover, development teams usually forget architecture design adaptive to possible environmental changes. As a result, software development faces the risk of insecurities in changing cases.

Risks in Software Development

Experts confirmed that security risks occur most frequently during software development. The main reason is access control and source code management since the project is often divided into parts and assigned to different developers.

The writing codes might not be consistent from a security perspective. Plus, there is not enough attention to access and update history. At that time, the team needs to define well-secure coding guidelines and code-reviews procedure in the first place.

Risks in Testing Phase

Security problems are more complex to test than functions since they lack acceptance criteria, and the abstract properties for security testing are not inherently checkable. For instance, test cases for crashing situations often involve combined bugs at a time. Then, there is a risk that testers and developers cannot replicate the security problems and investigate the issues.

Risks in Deployment Phase

Deploying an application is a sensitive process due to various environments. During the transfer time, the code might be exposed to hackers trying to find vulnerabilities across systems and steal the data or damage the systems.

Risks in Operation and Maintenance

Even if the software is installed and runs smoothly on the live system, it is still under the possible security risks: both systematic and unsystematic, such as data exposure, hacking efforts, etc.

Best Security Practices in SDLC

Best practices to tackle security risks in the software development life cycle
Although the risks might differ from phase to phase, we have standard security practices for your attention to minimize the chances of security risks in the software development life cycle.

- Educating the development team on security SDLC
- Building risk management strategies to handle risks in SDLC
- Eliciting requirements carefully, including security requirements
- Building secure coding checklists and doing code reviews
- Running automatic test cases to detect difficult risks

Take notes of all security risks in the Software Development Life Cycle and discuss them next time in the planning phase of your project – be better than sorry. The earlier you detect the security risks, the more quickly and less money you have to spend during the project!