# Unit 2 – Learning Outcomes: UML Modelling to Support Secure System Planning

## 1. Gain an awareness of the opportunities for integrating software security development practices at each stage of the agile SDLC.

To guarantee that security issues are handled from the beginning and throughout the development process, it is essential to integrate software security development practises into each Agile Software Development Life Cycle (SDLC) stage (Keramati & Mirian-Hosseinabadi, 2008).

We implement information security measures to reduce the chance that people, systems, or data will be utilised in ways that result in bodily danger, financial loss, or reputational damage. Agile is an iterative process. Thus, we must create pathways to reduce risk in each iteration and throughout the pipeline (Butler, 2022).

Below is an overview of how each phase of the agile SDLC might incorporate software security procedures:

### 1. Planning Phase:

According to EC-Council (2020) report, before planning, we must address security requirements by following critical questions during the most important but often contentious phase of the secure software development life cycle, which are:

1. Identify security requirements: This includes understanding the security risks and threats the software will face and the organisation's security requirements.
2. Develop a security plan: This plan should outline the security activities that will be performed throughout the SDLC, as well as the roles and responsibilities of the different stakeholders.
3. Create a security awareness program: This should educate developers and other stakeholders about best practices.

## 2. Design Phase:

Due to the growing cybersecurity risk and increased business liability awareness, Gumbley (2020) states that software development teams need efficient methods to integrate security measures into their work. Here are some procedures for this phase:

1. Use secure design principles: This includes using certain coding practices, such as input validation and output encoding.
2. Perform threat modeling: This involves identifying potential threats to the software and designing mitigations to address those threats.
3. Use secure development tools: There are several tools available that can help developers to write more secure code.

## 3. Development:

Since most modern programmes aren't created from the start, current application developers can't limit their attention to the code they write. Instead, developers rely on existing functionality, typically given by free, open-source components, to bring new features and value to the organisation as rapidly as possible (Snyk, 2022). The development and maintenance phase, for instance, might comprise the following in this scenario:

1. Implement security reviews: This involves having security experts review the code for security vulnerabilities.
2. Use automated security testing tools: These tools can help to identify security vulnerabilities in the code.
3. Adopt a DevOps approach: Security testing is automated in the SDLC process, particularly in DevOps. Collaboration between DevOps and engineers should be integrated into the Development and Maintenance Phase.

## 4. Testing Phase:

Security testing ensures software is safe from cyber-attacks and unauthorised inputs—non-functional testing checks for correct design and configuration (Moradov, 2021).

Security testing confirms that an organisation's systems, applications, and data adhere to security principles. For this phase, consider the following security procedures:

1. Penetration testing: This involves having a security expert attempt to exploit vulnerabilities in the software.
2. Application Security Testing: This includes testing software for security vulnerabilities, such as input validation errors, cross-site scripting (XSS) vulnerabilities, local privacy flaws, persistent cookies, unreliable SSL cyphers, and vulnerable URL parameters.

## 5. Deployment Phase:

Security should be a priority during the SDLC pipeline to identify and fix potential flaws before deploying to production, reducing the chances of discovery and mitigating their impact (Snyk, 2022). During this phase, it is essential to take into account the following security measures:

1. Securely configure the software: This entails employing appropriate configurations for the product's dependencies and secure default settings.
2. Keep an eye out for security flaws in the software with automated security monitoring tools.
3. Apply security patches to the software as soon as feasible to reduce the chance of exploitation.

## 2. Understand the range of ways in which software security can be problematic.

Software is crucial to our daily lives, from driving automobiles to using the internet to access banking services and flying. Also, handling sensitive data affecting our privacy, way of life, and livelihoods is essential. The software can, however, be safe if it has the proper security mechanisms in place, similar to being on a high wire without a net. Software must be secure to prevent potential repercussions (Allen, 2007).

Based on a report by Check Point Software (N.D.), the most recent OWASP Top Ten list is shown below, along with the vulnerabilities that might jeopardise software security and cause issues:

**1. Broken Access Control:** Access control systems prevent unauthorised access to data or functionality. Broken access control vulnerabilities allow attackers to bypass controls or grant excessive privileges. For instance, a web app may allow users to view others' accounts by altering the URL.

**2. Cryptographic Failures:** Cryptographic failures can occur due to implementation or configuration errors, compromising data privacy and security. These failures include a lack of encryption, misconfigurations, and insecure key management. Examples include using an insecure hash algorithm for password storage or failing to salt passwords properly.

**3. Injection:** Injection vulnerabilities occur when user input is not sanitised correctly before processing, which is particularly dangerous in languages like SQL, where user data and commands mix. Malicious information that includes single or double quotation marks can alter the command being processed.

**4. Insecure Design:** Design flaws in software can create vulnerabilities that compromise system security. For instance, a lack of authentication in an application that handles sensitive data leaves it unprotected, even if implemented perfectly.

**5. Security Misconfiguration:** Application security depends on both design and configuration. Default settings may be vulnerable, and users can unwittingly weaken system security by enabling unnecessary apps or leaving default accounts and passwords active. Exposing too much information through error messages is another standard configuration error.

**6. Vulnerable and Outdated Components:** Supply chain vulnerabilities constitute a significant concern. Malicious or vulnerable code can be inserted into commonly used libraries and third-party dependencies. Without visibility and scanning, organisations may be susceptible to exploitation. Failure to apply security updates to dependencies can open exploitable vulnerabilities to attack.

**7. Identification and Authentication Failures:** Weak authentication processes and failure to properly validate authentication information can lead to identification and authentication failures. A lack of MFA in an application can make it vulnerable to credential-stuffing attacks.

**8. Software and Data Integrity Failures:** The Software and Data Integrity Failures vulnerability in the OWASP Top 10 list addresses security weaknesses in an organisation's DevOps pipeline and software updates, which can lead to a breach like the SolarWinds hack. This includes using untrusted third-party code, failing to secure CI/CD access, and not validating updates. Attackers can exploit this by replacing trusted modules with malicious versions, allowing them to run malicious code.

**9. Security Logging and Monitoring Failures:** Failing to log significant security events or not monitoring log files properly can lead to security incidents. This vulnerability has moved up on the list of concerns and can hinder an organisation's ability to detect and respond to potential incidents in real-time.

**10. Server-Side Request Forgery:** SSRF is a vulnerability where a web app doesn't validate user-provided URLs. It is rare but can have a significant impact. Attackers can use it to bypass access controls and reach target URLs through vulnerable apps. The Capital One hack is an example of this type of attack.

In addition to OWASP, there are various other standards, guidelines, and certifications for software security to help ensure compliance with applicable regulations. These practices include Common Criteria, TOGAF, SAMM, BSIMM, ASVS, and SAFECode, as well as national and international standards like PCI, NIST, and ISO/IEC (Ramirez et al. 2020).

Software security tools and solutions come in a vast range. Organisations must develop a strategy, like any other security measure, to guarantee that software security solutions are still effective and serve their interests (Thales Group, N.D).

Thales Group (N.D) added that it is crucial that regularly patching software and enforcing the principle of least privilege can reduce the risk of attacks. Automating security tasks and educating employees on risks is vital. Write down software security policies and monitor activity to track threats over time. Organisations need to implement software security solutions to follow best practices.

**References:**

Keramati, H. and Mirian-Hosseinabadi, S.-H. (2008). *Integrating software development security activities with agile methodologies*. [online] IEEE Xplore. doi:https://doi.org/10.1109/AICCSA.2008.4493611.

Butler, N. (2022). *Security in Agile software development: a simple guide | Bigger Impact*. [online] Available at: https://www.boost.co.nz/blog/2022/02/security-in-agile-software-development.

EC-Council (2020). *What Are the Five Phases of the Secure Software Development Life Cycle?* [online] Cybersecurity Exchange. Available at: https://www.eccouncil.org/cybersecurity-exchange/application-security/what-are-the-five-phases-of-the-secure-software-development-life-cycle/#:~:text=Requirement%20Planning&text=While%20planning%20may%20be%20the [Accessed 20 Aug. 2023].

Gumbley, J. (2020). *A Guide to Threat Modelling for Developers*. [online] Available at: https://martinfowler.com/articles/agile-threat-modelling.html.

Snyk (2022). *Secure SDLC | Secure Software Development Life Cycle | Snyk*. [online] snyk.io. Available at: https://snyk.io/learn/secure-sdlc/. [Accessed 20 Aug. 2023].

Moradov, O. (2021). *Security Testing: Types, Tools, and Best Practices*. [online] Bright Security. Available at: https://brightsec.com/blog/security-testing/.

Allen, J. (2007). Why is Security a Software Issue? *EDPACS*, 36(1), pp.1–13. doi:https://doi.org/10.1080/07366980701500734.

Check Point Software. (N.D.). *OWASP Top 10 Vulnerabilities*. [online] Available at: https://www.checkpoint.com/cyber-hub/cloud-security/what-is-application-security-appsec/owasp-top-10-vulnerabilities/.

Thales Group (N.D.). *Software Security* | *What is software security?* [online] Available at: https://cpl.thalesgroup.com/software-monetization/what-is-software-security.

Ramirez, A., Aiello, A. & Lincke, S.J. (2020). *A Survey and Comparison of Secure Software Development Standards*. [online] IEEE Xplore. doi:https://doi.org/10.1109/CMI51275.2020.9322704.