DOI: http://dx.doi.org/10.26483/ijarcs.v14i1.6949

Volume 14, No. 1, January-February 2023

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

SECURE SOFTWARE DEVELOPMENT LIFECYCLE: A CASE FOR ADOPTION IN SOFTWARE SMES

Wisdom Umeugo
Ph.D. Candidate
School of Computer and Information Sciences
University of the Cumberlands, Kentucky, USA

Abstract: Software is widely deployed and used for managing critical daily domestic, social, and economic activities. Due to software's economic value, software is a high-value target of malicious actors and a primary source of many information security vulnerabilities. Software must be engineered to be secure because of its value. Traditional approaches to software security treat software as an addon and have been proven inadequate at producing secure software. Practicing the secure software development lifecycle (SSDLC) is recommended in academic literature. Software SMEs must adopt and practice the SSDLC for increased security of published software. This paper explores the SSDLC and makes a case for its adoption with the goal of informing security decision-makers of Software SMEs.

Keywords: software security; application security; secure software; secure software development lifecycle; ssdlc; ssdlc adoption

I. INTRODUCTION

Software is increasingly pervasive, finding applications in every domain of human society [1], [2]. Due to the high dependence on software across the public and private sectors, assurance is needed that software will perform dependably in all circumstances [3]. Traditional approaches to securing software mainly focus on host, perimeter, and network security using intrusion detection and prevention systems, firewalls, and anti-malware [4]. These approaches fail to incorporate adequate protection into the software during its development and have proven inadequate at providing software assurance [3], [4]. A move to a software development lifecycle (SDLC) with security incorporated in every phase has been repeatedly advocated [4]–[6]. The commonly used term for this security-infused software development lifecycle is "Secure SDLC" (SSDLC). Software SMEs are the dominant publishers of software. Therefore, attempts at improving the security of released software must ensure that software SMEs adopt and practice the SSDLC. This research explores the SSDLC, its requirements, and its adoption challenges by software small and medium enterprises (SMEs).

II. SECURE SOFTWARE

The primary goal of software security is to guarantee the confidentiality, integrity, and availability of software components and data [4]. Confidentiality entails protecting software from unauthorized access [3]. Protecting software integrity refers to preventing the unauthorized modification of software components through tampering, corruption, and destruction [3]. Guaranteeing software availability requires ensuring that the software and the service or the data it provides are functional and accessible to authorized parties when needed [3]. The goal of software assurance is to ensure high confidence in software's freedom from exploitable vulnerabilities during the software's lifecycle [7]. Therefore, software must be secure throughout its lifecycle to achieve high software assurance. Security must thus be integrated into all stages of the SDLC [4]. For software to be considered secure, its risks must be managed, and it must be engineered to incorporate software security principles and properties from the early stages of its lifecycle [3], [8].

A. *Secure Software Principles*

Various software security principles guiding the design of software security solutions and processes have been published in the literature. White [9] identified and described eight commonly cited software security design principles in the literature. These are summarized as follows: (a) Open design: security should be dependent on implementation and not the design, which should not be hidden; (b) Fail-safe defaults: if the software fails, it should fall back to a secure default; (c) Least privilege: only the minimum set of privileges required to perform any action should be granted in any situation (d) Economy of mechanism: keep security designs and implementation as simple, understandable and straightforward as possible; (e) Separation of privilege: two or more independent conditions should be met to execute a security-sensitive activity successfully; (f) Complete mediation: authorization must be checked for every access request or to perform any action; (g) Least common Mechanism: mechanisms for accessing resources should not be shared between users and (h) Psychological acceptability: security features and controls should be usable. Alenezi and Almuairfi [10] included four more principles specified by the Open Web Application Security Project (OWASP): (a) Use whitelisting permission model: deny by default and explicitly specify what is permitted; (b) Detect intrusion: keep and monitor security-related access log entries; (c) Do not trust infrastructure: assume uncertainty about the operating environment, and (d) Do not trust external services: assume external services and systems the software communicates with are always insecure.

B. *Characteristics of Secure Software*

From a software assurance perspective, the software is secure when free from vulnerabilities, exhibits withstanding security properties under malicious attack, and is of correct design and implementation [3], [11]. Secure software's ability to exhibit attack tolerance depends on its inherent engineering to recognize attack patterns, resist misuse, and tolerate and recover from failures or errors [3]. Software must be engineered with the software security principles and properties from the start of its lifecycle to be secure [3], [11]. According to [3], when a fault is encountered in secure software, the software fails to secure defaults and recovers securely. This is important in safety-critical software, where faults can seriously impact life and property [3].

III. REQUIREMENTS FOR DEVELOPING SECURE SOFTWARE

Secure software requires an organized security-focused effort using effective security processes and tools [12], [13]. Integrating security into the SDLC requires staff with expertise, such as software security architects, and enforcing software security policies and controls [13]. Thus, secure software development involves software engineering and security engineering efforts supported by effective management policies and practices [13], [14].

A. Proper Management Structure

A management structure emphasizing software security is needed to oversee the SSDLC [13]. Ransome and Misra [13] and Rindell et al. [15] recommended that the management of the SSDLC be placed under the software quality assurance office because including security in quality assurance will ensure it is emphasized during development.

B. Knowledgeable staff

Knowledgeable staff must oversee and implement security processes in the SSDLC [13]. The presence of security experts is influential in implementing secure software practices [16]. In addition to skilled software engineers who implement security into the product, security staff capable of thinking like adversaries and identifying and mitigating vulnerabilities in software should be involved [13]. Software security architects, engineers, and security champions are the common staff roles in the SSDLC. Software security architects and engineers are experts in software security. They perform security requirements analysis, risk analysis, security engineering, security testing, and security sign-off during each phase of the SSDLC [13]. Security champions are generally software engineering team members with backgrounds in or passion for software security who volunteer to help oversee SDLC security practices in the team they belong to [13].

C. Security policies

Security policies are governance tools for creating, implementing, and maintaining security processes [17]. Security policies provide high-level guidelines for managing security processes [17]. Software security processes typically involve various activities. For example, risk management includes identifying, analyzing, and mitigating risk [18].

D. Secure Software Frameworks and Maturity Models

Secure software frameworks and maturity models are required to guide the implementation, operation, and maturation of SSDLC practices. Various industry-standard software security maturity models and secure software development frameworks exist that identify, catalog, and organize software security processes and their related activities. Examples of frameworks that guide the implementation of software security-related processes are Microsoft Security development lifecycle (SDL), the Comprehensive, Lightweight Application Security Process (CLASP), and the Team Software Process for Secure Software Development (TSP-Secure) [3], [19].

E. Software security tools

Software security tools are also required when developing secure software to help ease and automate various software security tasks [13]. The quality and accuracy of the security tools are essential as they affect software developers' motivation to use them [20]. For example, Jamil et al. [20] reported that developers were demotivated from using Static Application Security Testing (SAST) tools by their high false-positive rates. Typical application security tools identified in the literature include vulnerability scanners, SAST, dynamic application security testing tools (DAST), fuzzing tools, and penetration testing tools [13], [21].

IV. SECURE SOFTWARE DEVELOPMENT LIFECYCLE

Software of all sizes and complexity must be dependable, and dependability is best achieved when software is engineered to be secure from the start [3]. An SSDLC where security processes are infused into all phases of the SDLC is critical to reducing security vulnerabilities in developed software [3], [8], [22]. Various authors recommend that the SSDLC should be risk-driven and that risk management activities should drive security activities at each phase of the SSDLC [3], [8], [23]. A summary of the security activities at each stage of the SDLC is depicted in Fig. 1.

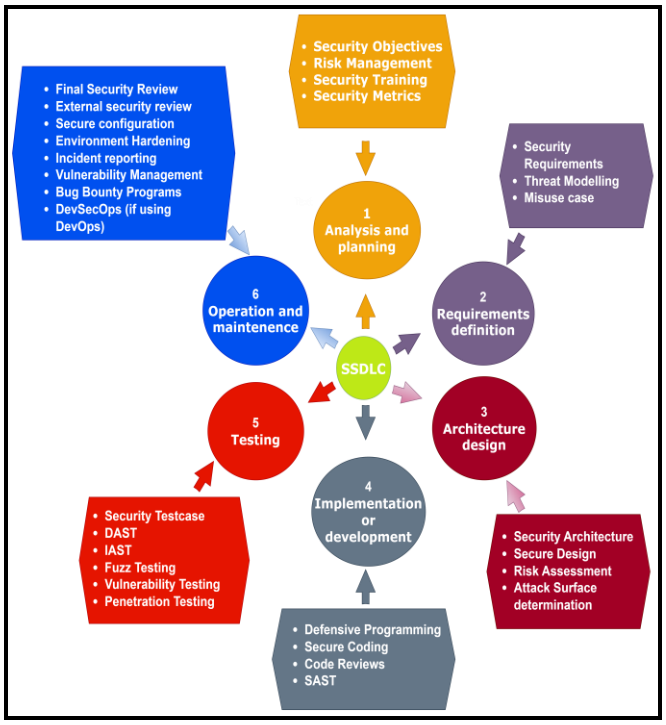

Figure 1. The Secure Software Development Lifecycle

A. Planning Phase

In the Planning phase of the SSDLC, a security assessment is made of the software's basic concept to determine the security objectives and profile [13]. A risk management framework is chosen, and preliminary risk management is conducted to inform stakeholders early of risks inherent to the software project's security and success [8]. Security training and awareness, general and tailored to specific roles, are also conducted in the planning phase [8].

B. Requirements definition phase

In the requirements definition phase of the SSDLC, security requirements are gathered and documented using requirement engineering techniques [8]. Security requirements include confidentiality, integrity, availability, authentication, authorization, session management, configuration, and environmental requirements [11]. These requirements are obtained from the preliminary threat models, available use cases, security policies, operational documents, and stakeholder requirements elicitation techniques [8]. Further threat modeling is performed at this phase to create misuse cases that demonstrate potential security exploits by threat actors and to define the expected behavior of the software under these circumstances [8].

C. Architecture and design phase

In the architecture and design phase of the SSDLC, a secure architecture for the software is designed by software security architects from the requirements and translated into lower-level implementable designs for developers [8]. The security architecture and controls are determined based on the risk mitigation plan [13]. Ruggieri et al. [24] recommended that security principles be incorporated into the design and that the design should be externally reviewed. Additional threat modeling and attack surface evaluation may be performed on the design until the design leaves an acceptable level of residual risk unaddressed [11].

D. Development Phase

In the development phase of the SSDLC, the secure design is translated into programming language code and configuration by software developers [8]. A secure programming language is chosen and used by software developers to implement security features. Several authors recommended that software developers practice defensive programming and follow secure coding guidelines for implementing misuse cases and reducing the software attack surface [8], [25]. Secure coding guidelines are general, such as OWASP Secure Coding Practices, or specific to a programming language [25]. Examples of programming language-specific secure coding guidelines are MISRA-C and CERT SEI-C for the C programming language, MISRA-C++ for C++, and CERT SEI-Java for Java [25]. Manual and automated code analysis is also performed during the development stage. Manual code analysis in the security context involves performing code vulnerability checks during peer code reviews [10]. Automated code analysis at the development stage is performed using static analysis tools that scan the code for vulnerabilities without executing the code [11], [13], [26].

E. Testing phase

In the Testing phase of the SSDLC, security tests specified by security architects and engineers are executed with the help of software quality assurance testers [13]. Security testing is conducted to verify software resilience under attack and is often called attack surface validation [11]. Security tests include Dynamic Application Security Testing (DAST), Interactive Application Security Testing (IAST), Fuzz testing, Vulnerability scanning, and Penetration testing [13], [21].

F. Operations and Maintenance phase

In the Operations and Maintenance phase of the SSDLC, the security team performs a final security review of the software’s release candidate [11], [13]; Certified third-party testers may be required to audit the software before release, especially when the software must pass some external certification and regulatory requirements [11]. The security architect signs off on the release candidate if the final security review is passed or fails with acceptable residual risk [11]. The production environment and network are hardened with minimum baseline security configurations, and the software is installed or deployed [11]. Vulnerability management, bug bounties, and secured DevOps pipelines are also implemented and updated at this stage.

V. SECURE SOFTWARE FRAMEWORKS

Absolute Software security is impossible due to many factors; however, the number of vulnerabilities can be drastically reduced by following secure software standards, practices, and recommendations for developing secure software [27]. These standards and industry best practices are maintained by various organizations such as the ISO, IEEE, NIST, and the U.S department of defense (DOD) [27]. Organizations developing software can adopt an industry-standard framework for implementing security over the entire SDLC. A brief review of the most cited frameworks is given as follows.

A. Comprehensive Lightweight Application Security Process (CLASP)

The Comprehensive Lightweight Application Security Process (CLASP) is a process for introducing security into the early stages of the SDLC [28]. CLASP is a comprehensive collection of methods, practices, roles, and resources for applying repeatable and measurable security to software development [3], [28]. CLASP provides views that are perspectives to approach CLASP, such as views based on best practices, roles in the framework, activity implementation, and vulnerability intended to be mitigated [28]. CLASP is designed to help individual developers develop secure software and for organizations to build secure software practices [27]. CLASP appears to no longer be updated by OWASP but is still widely referenced in current literature.

B. Security Assurance Maturity Model (SAMM)

OWASP developed the Security Assurance Maturity Model (SAMM) as a self-assessment model to guide the application, evaluation, and improvement of software security practices throughout the SDLC [29]. SAMM is prescriptive and flexible, allowing organizations to determine their target software security maturity level [29]. SAMM categorizes 15 software security practices into five software development business functions: governance, design, implementation, verification, and operations [29]. Each security practice contains a set of security-related activities structured into three maturity levels with associated success metrics [29]. Based on SAMM documentation, organizations first assess their software security posture and maturity level, then define their security targets and their target software security maturity level to apply SAMM. The SAMM implementation roadmaps are then used to achieve the targeted software security maturity level by implementing prescribed activities [29]

C. Building Security in Maturity Model (BSIMM)

The Building Security in Maturity Model (BSIMM) is a descriptive model of activities for evaluating and assessing software security initiatives [30]–[32]. BSIMM comprises 122 software security activities grouped into 12 practices and organized into four domains: governance, intelligence, secure software development (SSDL), and deployment [33]. BSIMM is best used as a yardstick for measuring other secure software development frameworks or comparing an organization's software security initiatives against others [30]–[32].

D. Microsoft Security Development Lifecycle

Microsoft [34] describes its Security Development Lifecycle (SDL) as a secure software framework to integrate security and privacy into all phases of the SDLC. According to Microsoft (n.d.) [34], the goal of SDL is to ensure the development of secure software that meets compliance requirements at a reduced cost. Microsoft [34] further described the SDL as a cost-effective security optimization model for building security into the SDLC from scratch based on five capability areas that map roughly to stages in the SDLC: (a) Training, policy, and organizational capabilities, (b) Requirements and design, (c) Implementation, (d) Verification, (e) Release and response. Each of the capability areas contains recommended security-related activities. The SDL advocates the creation of security advisors and security champion roles for accountability in the early stages of the SDLC. The SDL is an adaptable and process-agnostic guide to map security activities and security-related deliverables into all phases of the SDLC [3].

E. Microsoft Security Development Lifecycle Agile

According to Microsoft, agile software development practices focus on rapidly creating features, leaving security in the backseat, thus posing a challenge to integrating SDL [35]. For this reason, Microsoft released the Security Development Lifecycle Agile (SDL-agile) as a guide for integrating SDL into agile Software development practices [36]. According to Microsoft, to successfully incorporate SDL into agile, the SDL activities must be reorganized to fit agile methods [35]. Also, any SDL activity performed to develop a feature must be lean or just enough for that feature. SDL-agile adapts SDL activities into three categories: activities performed once at the start of the project, activities completed in every sprint, and bucket activities which are optional activities performed in a sprint [36]. SDL-agile recommends performing threat modeling and final security review in addition to the standard implementation activities of the SDL in each sprint [36].

VI. THE COSTS OF FIXING SOFTWARE SECURITY INCIDENTS

Exploited and unexploited vulnerabilities in released software have consequences for the organization due to high costs and efforts to fix them [11], [13], [37]. One of the reasons authors have consistently advocated using the SSDLC is to reduce the number of vulnerabilities and the associated costs to fix them. Paul [11] estimated that the cost of fixing security bugs at the production stage of the SDLC is as much as 30 times the cost of fixing the same bug at the requirements stage. It is, therefore, more cost-effective to identify and fix vulnerabilities early in the SDLC [37].

The costs of fixing security incidents in deployed software are incurred in the form of service downtime, data breaches, punitive fines, loss of developer productivity, development cycle interruptions due to work on security patches, vulnerability scope creeps, and legal, customer trust, and reputation issues [2], [13], [38]. The impact and costs of exploited software vulnerabilities vary based on the industry, affected users, impacted resources, and the type and severity of the vulnerability [38]. The greater the sensitivity or importance of the industry and the perceived severity by the affected users, the greater the costs to fix the vulnerability [38]. For example, financial software caters to the sensitive finance industry and can severely impact client finances if vulnerable financial software is exploited [38].

Exploited software vulnerabilities can also lead to bad publicity, public relations crisis, and legal liabilities that can financially impact the software publisher. Anwar et al. [38] reported that vulnerabilities reported in the press hurt the affected organization's stock price. The large punitive settlement, stock price drop, and reputation damage Equifax suffered due to its 2017 data breach discussed earlier is an example of the high costs of software vulnerability in released software. Another software vulnerability exploits with a high remediation cost was the SolarWinds hack reported in December 2020 [39]. The estimated costs to SolarWinds in USD included $25 million in cyber insurance and cybersecurity improvements, $90 million in client indemnity, $90 million in forensics and incident response, and $100 billion in software and system recovery [39]. In addition, SolarWind's stock plummeted as much as 25 percent as its reputation fell, and customers lost trust in its products [40].

vulnerabilities in high-consequence software such as national security systems, banking software, and medical software can have a catastrophic impact if exploited because they may result in the loss of life, health, and financial well-being of impacted individuals [3]. Early discovery and remediation of vulnerabilities protects users, saves the organization money, and frees developers to focus on features rather than security patching [41].

VII. SSDLC ADOPTION CHALLENGES IN SMES

Tuape and Ayalew [42] estimated that 95 percent of software publishers are SMEs. According to the OECD office for SME and entrepreneurship performance [43], compared to large enterprises, SMEs are generally characterized by lower capital, productivity levels, technology adoption, internationalization, and research-centered innovation than large enterprises. Software SME SSDLC adoption remains inadequate. Alghamdi [44] revealed that only 51 percent of software SMEs that develop software in-house adopted secure software practices during all phases of the SDLC. The reasons noted for the low SSDLC adoption in software SMEs are summarized as follows.

1) *Security is still viewed as an addon:* The traditional view of security as a secondary feature and addon prevents adequate security from being built into software during development [5], [13].

2) *Complexity and cost:* SSDLC is complex and costly to implement and practice leading to management overhead and lack of management support [14], [44], [45].

3) *Widespread adoption of the agile model:* The agile model focuses on quickly developing usable software, making the incorporation of security practices in all stages of the agile SDLC challenging [46], [47].

4) Reliance on developers for security: Software SMEs rely more heavily on developers than processes [48]. Security in the SDLC is thus being left to developers, who usually do not have the security expertise to adequately manage and implement secure software practices [13], [49].

5) Lack of Developer acceptance of secure software practices: Developers may not be motivated to accept the SSDLC. According to Assal and Chiasson [50], amotivation reasons cited by developers included a perceived lack of security competence due to a lack of resources and inadequate support. Other contributors to developer amotivations include a lack of interest from induced passiveness or exclusion from security responsibilities, optimistic bias against attack, the erroneous perception that exploits have negligible impact, and personal philosophical resistance [50]. Assal and Chiasson [50] also noted that developers' security motivations were more significant in larger enterprises than in SMEs.

6) Complexity of existing SSDLC frameworks: Most software security guidelines and secure software frameworks are more suited to software development in large enterprises than for SMEs [44], [46].

7) SDLC methodology Constraints: The SDLC methodology practiced presents constraints to implementing the SSDLC. As mentioned earlier, agile development is now widespread in the software industry for building various types and scales of software [12], [51]. Agile SDLC has a notable incompatibility with security practices consistently highlighted by various authors [47], [52]–[54]. Microsoft's acknowledgment of this incompatibility led to the release of the previously discussed SDL-agile framework. There is also a lack of industry-standard practical frameworks for incorporating the SSDLC in the less popular SDLC models like the crystal method, the big bang model, extreme programming, prototyping, and rapid application development.

8) Poor software security awareness and education: Lack of software security awareness and education hinders SME SSDLC adoption [55]. Poor security awareness also impacts developers' acceptance of security practices. Witschey et al. [56] showed that developer security training and exposure to tools positively impacted the developer's intention to adopt security tools.

9) Lack of security policies and strong security culture: Similar to the adoption of security practices, security policies, and strong security culture are important requirements for successful software security practices adoption.

VIII. CONCLUSION

Ensuring that published software is secure remains an important ongoing topic. Software must be built with the assurance that it can withstand misuse and malicious attack. Software has to be engineered from the start to be secured to achieve a high level of security assurance. Adopting the SSDLC enables software to be designed, developed and deployed securely. Both the software itself and threats evolve, introducing new exploitable vulnerabilities. The easy distribution and widespread use of software make fixing vulnerabilities expensive and the costs of exploits higher for software SMEs. Efforts should be made to increase software SME SSDLC adoption. Future work should further explore the reasons for the inadequate SSDLC adoption in SMEs and identify potential practical remedies that can be implemented by SME software security decision-makers and government policymakers.

IX. REFERENCES

[1] R. A. Khan and S. U. Khan, "A preliminary structure of software security assurance model," in Proceedings of the 13th International Conference on Global Software Engineering, New York, NY, USA, May 2018, pp. 137–140, doi: 10.1145/3196369.3196385.

[2] E. Venson, R. Alfayez, M. M. F. Gomes, R. M. C. Figueiredo, and B. Boehm, "The impact of software security practices on development effort: an initial survey," in 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Sep. 2019, pp. 1–12, doi: 10.1109/ESEM.2019.8870153.

[3] U.S. Department of Homeland Security, "Security in the softwarelifecycle: Making software development processes—and software produced by them—more secure. DRAFT Version 1.2. ," 2006, Accessed: Jan. 28, 2023. [Online]. Available: http://www.cert.org/books/secureswe/SecuritySL.pdf.

[4] R. A. Khan, S. U. Khan, H. U. Khan, and M. Ilyas, "Systematic mapping study on security approaches in secure software engineering," IEEE Access, vol. 9, pp. 19139–19160, 2021, doi: 10.1109/ACCESS.2021.3052311.

[5] H. Al-Matouq, S. Mahmood, M. Alshayeb, and M. Niazi, "A maturity model for secure software design: A multivocal study," IEEE Access, vol. 8, pp. 215758–215776, 2020, doi: 10.1109/ACCESS.2020.3040220.

[6] R. Fujdiak et al., "Managing the secure software development," in 2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Jun. 2019, pp. 1–4, doi: 10.1109/NTMS.2019.8763845.

[7] W. J. R. Nichols and T. Scanlon, "DoD Developer's Guidebook for Software Assurance," 2018.

[8] M. Alenezi and S. Almuairfi, "Security risks in thesoftware development lifecycle," IJSEA, vol. 8, no. 3, pp. 7048–7055, Sep. 2019.

[9] C. A. White, "Root causes of insecure internet of things and holistically addressing them," in 2020 International Conference on Computational Science and Computational Intelligence (CSCI), Dec. 2020, pp. 1066–1074, doi: 10.1109/CSCI51800.2020.00198.

[10] M. Alenezi and S. Almuairfi, "Essential activities for secure software development," IJSEA, vol. 11, no. 2, pp. 1–14, Mar. 2020, doi: 10.5121/ijsea.2020.11201.

[11] M. Paul, Official (ISC)2 guide to the CSSLP CBK. Auerbach Publications, 2013.

[12] M. G. Jaatun and D. Soares Cruzes, "Care and feeding of your security champion," in 2021 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), Jun. 2021, pp. 1–7, doi: 10.1109/CyberSA52016.2021.9478254.

[13] J. Ransome and A. Misra, Core Software Security. Auerbach Publications, 2018.

[14] N. Davis, W. Humphrey, S. T. Redwine, G. Zibulski, and G. McGraw, "Processes for producing secure software," IEEE Secur. Privacy Mag., vol. 2, no. 3, pp. 18–25, May 2004, doi: 10.1109/MSP.2004.21.

[15] K. Rindell, J. Ruohonen, and S. Hyrynsalmi, "Surveying secure software development practices in finland," in Proceedings of the 13th International Conference on Availability, Reliability and Security - ARES 2018, New

York, New York, USA, Aug. 2018, pp. 1–7, doi: 10.1145/3230833.3233274.

[16] S. L. Kanniah and M. N. Mahrin, "Secure software development practice adoption model: A delphi study.," Journal of Telecommunication, Electronic and Computer Engineering (JTEC), vol. 10, no. 2, pp. 71–75, 2018.

[17] M. H. Sharif, R. Datta, and M. Valavala, "Identifying Risks and Security Measures for E-Commerce Organizations.," Int. J. Eng. Appl. Sci. Technol, vol. 4, no. 5, 2019.

[18] A. Asadoorian, M. Alberto, and M. L. Ali, "Creating and using secure software," in 2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), Oct. 2020, pp. 0786–0792, doi: 10.1109/UEMCON51285.2020.9298046.

[19] J. C. Núñez, A. C. Lindo, and P. G. Rodríguez, "A preventive secure software development model for a software factory: a case study.," IEEE Access, vol. 8, pp. 77653–77665, 2020.

[20] A. M. Jamil, L. ben Othmane, A. Valani, M. Abdelkhalek, and A. Tek, "The current practices of changing secure software: An empirical study," in Proceedings of the 35th Annual ACM Symposium on Applied Computing, New York, NY, USA, Mar. 2020, pp. 1566–1575, doi: 10.1145/3341105.3373922.

[21] T. W. Thomas, M. Tabassum, B. Chu, and H. Lipford, "Security during application development: an application security expert perspective," in Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18, New York, New York, USA, Apr. 2018, pp. 1–12, doi: 10.1145/3173574.3173836.

[22] F. Mateo Tudela, J.-R. Bermejo Higuera, J. Bermejo Higuera, J.-A. Sicilia Montalvo, and M. I. Argyros, "On combining static, dynamic and interactive analysis security testing tools to improve OWASP top ten security vulnerability detection in web applications," Appl. Sci., vol. 10, no. 24, p. 9119, Dec. 2020, doi: 10.3390/app10249119.

[23] R. A. Khan, S. U. Khan, H. U. Khan, and M. Ilyas, "Systematic literature review on security risks and its practices in secure software development," IEEE Access, vol. 10, pp. 5456–5481, 2022, doi: 10.1109/ACCESS.2022.3140181.

[24] M. Ruggieri, T.-T. Hsu, and M. L. Ali, "Security considerations for the development of secure software systems," in 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), Oct. 2019, pp. 1187–1193, doi: 10.1109/UEMCON47517.2019.8993081.

[25] T. E. Gasiba and U. Lechner, "Raising secure coding awareness for software developers in the industry.," 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), p. 141, 2019.

[26] M. Noman, M. Iqbal, and A. Manzoor, "A survey on detection and prevention of web vulnerabilities," ijacsa, vol. 11, no. 6, 2020, doi: 10.14569/IJACSA.2020.0110665.

[27] R. Trifonov, O. Nakov, G. Pavlova, S. Manolov, G. Tsochev, and P. Nakov, "Analysis of the principles and criteria for secure software development," in 2020 28th National Conference with International Participation (TELECOM), Oct. 2020, pp. 125–128, doi: 10.1109/TELECOM50385.2020.9299567.

[28] Cybersecurity & Infrastructure Security Agency, "Introduction to the CLASP Process | CISA," Cybersecurity & Infrastructure Security Agency, Jul. 03, 2013. https://www.cisa.gov/uscert/bsi/articles/best-practices/requirements-engineering/introduction-to-the-clasp-process#:%7E:text=Comprehensive%2C%20Lightweight%20Application%20Security%20Process,guided%20by%20formalized%20best%20practices. (accessed Feb. 05, 2022).

[29] OWASP Software Assurance Maturity Model, "OWASP SAMM," OWASP SAMM. OWASP software assurance maturity model. https://owaspsamm.org/ (accessed Jan. 28, 2023).

[30] K. Bernsmed, M. G. Jaatun, and P. H. Meland, "Safety Critical Software and Security - How Low Can You Go?," in 2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC), Sep. 2018, pp. 1–6, doi: 10.1109/DASC.2018.8569579.

[31] G. McGraw, "Software security and the building security in maturity model (BSIMM).," Journal of Computing Sciences in Colleges, vol. 30, no. 3, pp. 7–8, 2015.

[32] M. Shaikh, P. H. Ali Qureshi, M. Shaikh, Q. A. Arain, A. Zubedi, and P. Shaikh, "Security paradigms in SDLC requirement phase — A comparative analysis approach," in 2021 International Conference on Engineering and Emerging Technologies (ICEET), Oct. 2021, pp. 1–6, doi: 10.1109/ICEET53442.2021.9659614.

[33] BSIMM, "Software Security Framework ." https://www.bsimm.com/framework.html (accessed Feb. 27, 2022).

[34] Microsoft, "Microsoft Security Development Lifecycle Practices." https://www.microsoft.com/en-us/securityengineering/sdl/practices (accessed Oct. 28, 2022).

[35] Microsoft Learn, "Agile SDL: Streamline Security Practices For Agile Development | Microsoft Learn," Sep. 10, 2019. https://learn.microsoft.com/en-us/archive/msdn-magazine/2008/november/agile-sdl-streamline-security-practices-for-agile-development (accessed Jan. 28, 2023).

[36] M. G. Jaatun, "Architectural risk analysis in agile development of cloud software," in 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Dec. 2019, pp. 295–300, doi: 10.1109/CloudCom.2019.00050.

[37] D. Raghuvanshi, "Introduction to Software Testing.," International Journal of Trend in Scientific Research and Development (IJTSRD), vol. 4, no. 3, pp. 797–800, 2020.

[38] A. Anwar et al., "Measuring the cost of software vulnerabilities," ICST Transactions on Security and Safety, vol. 7, no. 23, p. 164551, Jun. 2020, doi: 10.4108/eai.13-7-2018.164551.

[39] R. Alkhadra, J. Abuzaid, M. AlShammari, and N. Mohammad, "Solar Winds Hack: In-Depth Analysis and Countermeasures," in 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), Jul. 2021, pp. 1–7, doi: 10.1109/ICCCNT51525.2021.9579611.

[40] L. Sterle and S. Bhunia, "On solarwinds orion platform security breach," in 2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI), Oct. 2021, pp. 636–641, doi: 10.1109/SWC50871.2021.00094.

[41] D. Votipka, R. Stevens, E. Redmiles, J. Hu, and M. Mazurek, "Hackers vs. testers: A comparison of software vulnerability discovery processes," in 2018 IEEE Symposium on Security and Privacy (SP), May 2018, pp. 374–391, doi: 10.1109/SP.2018.00003.

[42] M. Tuape and Y. Ayalew, "Factors affecting development process in small software companies," in 2019 IEEE/ACM Symposium on Software Engineering in

Africa (SEiA), May 2019, pp. 16–23, doi: 10.1109/SEiA.2019.00011.

[43] OECD, "SME Performance - OECD." https://www.oecd.org/cfe/smes/smeperformance.htm (accessed Mar. 01, 2022).

[44] F. Alghamdi, "Motivational company's characteristics to secure software," in 2020 3rd International Conference on Computer Applications & Information Security (ICCAIS), Mar. 2020, pp. 1–5, doi: 10.1109/ICCAIS48893.2020.9096815.

[45] D. Geer, "Are companies actually using secure development life cycles?," Computer, vol. 43, no. 6, pp. 12–16, Jun. 2010, doi: 10.1109/MC.2010.159.

[46] T. D. Oyetoyan, D. S. Cruzes, and M. G. Jaatun, "An Empirical Study on the Relationship between Software Security Skills, Usage and Training Needs in Agile Settings," in 2016 11th International Conference on Availability, Reliability and Security (ARES), Aug. 2016, pp. 548–555, doi: 10.1109/ARES.2016.103.

[47] I. A. Tøndel, D. S. Cruzes, M. G. Jaatun, and G. Sindre, "Influencing the security prioritisation of an agile software development project," Computers & Security, vol. 118, p. 102744, Jul. 2022, doi: 10.1016/j.cose.2022.102744.

[48] A. Tosun, A. Bener, and B. Turhan, "Implementation of a software quality improvement project in an SME: A before and after comparison," in 2009 35th Euromicro Conference on Software Engineering and Advanced Applications, Aug. 2009, pp. 203–209, doi: 10.1109/SEAA.2009.52.

[49] Z. A. Maher, A. Shah, S. Chandio, H. M. Mohadis, and N. H. B. A. Rahim, "Challenges and limitations in secure software development adoption - A qualitative analysis in Malaysian software industry prospect," IJST, vol. 13, no. 26, pp. 2601–2608, Jul. 2020, doi: 10.17485/IJST/v13i26.848.

[50] H. Assal and S. Chiasson, "Motivations and amotivations for software security.," SOUPS Workshop on Security Information Workers (WSIW). USENIX Association, p. 1, 2018.

[51] M. Choras et al., "Measuring and Improving Agile Processes in a Small-Size Software Development Company," IEEE Access, vol. 8, pp. 78452–78466, 2020, doi: 10.1109/ACCESS.2020.2990117.

[52] C. M. M. Bezerra, S. C. B. Sampaio, and M. L. M. Marinho, "Secure agile software development: policies and practices for agile teams," in Quality of information and communications technology: 13th international conference, QUATIC 2020, faro, portugal, september 9–11, 2020, proceedings, vol. 1266, M. Shepperd, F. Brito e Abreu, A. Rodrigues da Silva, and R. Pérez-Castillo, Eds. Cham: Springer International Publishing, 2020, pp. 343–357.

[53] F. Moyón, D. Méndez, K. Beckers, and S. Klepper, "How to Integrate Security Compliance Requirements with Agile Software Engineering at Scale?," in Product-Focused Software Process Improvement: 21st International Conference, PROFES 2020, Turin, Italy, November 25–27, 2020, Proceedings, vol. 12562, M. Morisio, M. Torchiano, and A. Jedlitschka, Eds. Cham: Springer International Publishing, 2020, pp. 69–87.

[54] D. Ionita, C. van der Velden, H.-J. K. Ikkink, E. Neven, M. Daneva, and M. Kuipers, "Towards Risk-Driven Security Requirements Management in Agile Software Development," in Information systems engineering in responsible information systems: caise forum 2019, rome, italy, june 3–7, 2019, proceedings, vol. 350, C. Cappiello and M. Ruiz, Eds. Cham: Springer International Publishing, 2019, pp. 133–144.

[55] M. Deschene, "Embracing security in all phases of the software development life cycle: A Delphi study," Undergraduate thesis, 2016.

[56] J. Witschey, O. Zielinska, A. Welk, E. Murphy-Hill, C. Mayhorn, and T. Zimmermann, "Quantifying developers' adoption of security tools," in Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015, New York, New York, USA, Aug. 2015, pp. 260–271, doi: 10.1145/2786805.2786816.

FIGURES

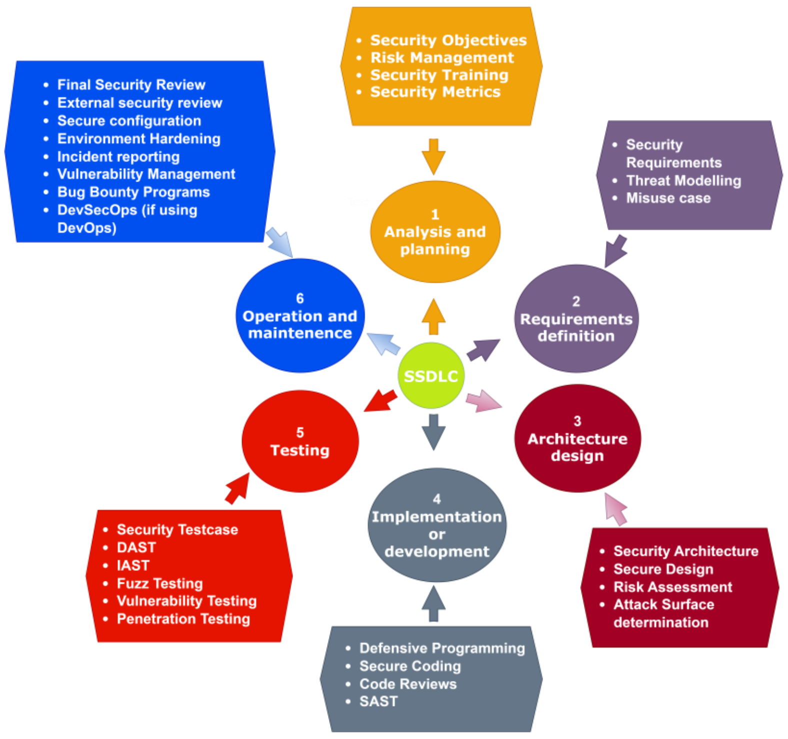

Figure 1 The secure software development lifecycle