UDC 004.056.53:512.6

Information Security Challenges in an Enterprise-Grade Software Development Lifecycle

Kamil Mahomedov

National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

Abstract

In an era of escalating cyber threats and digital complexity, the integration of information security into the software development lifecycle (SDLC) is imperative for building trustworthy enterprise-grade software systems. This literature review synthesizes and critically evaluates over 30 scholarly and industry sources to identify current practices, frameworks, and tools for SLDC implementation. It explores prominent cybersecurity frameworks, such as Microsoft's SDL, OWASP SAMM, NIST SSDF, and assesses how well they accommodate modern cloud security practices within contemporary SDLCs. Special attention is given to the DevSecOps paradigm, which integrates automated security checks and developer engagement into continuous integration and delivery pipelines, and to SBOMs as a means of exposing and managing third-party component risks in complex supply chains. Findings reveal persistent challenges related to integration with agile workflows, cost, lack of standardized metrics, and organizational resistance (i.e. the human factor). The overall result is the amalgamation of software security best practices extracted from the examined literature into a concise overview to assist further research in this area. The paper concludes with a call for more adaptable, scalable, and measurable security practices that align with modern software development methodologies aimed at facilitating the enterprise-grade integration and delivery of code.

Keywords: SDLC, CI/CD, DevSecOps, SBOM, cyberattacks, vulnerabilities, threat models, scalability

Introduction

The integration of information security throughout the Software Development Lifecycle (SDLC) has become a foundational requirement for mitigating modern software threats. As organizations face increasingly sophisticated cyberattacks, the consequences of inadequate security — including data breaches, financial losses, and reputational harm — grow more severe. Despite the availability of mature tools and methodologies, widespread vulnerabilities often stem from treating security as an afterthought rather than a foundational aspect of software engineering. This can happen due to several reasons, ranging from budgetary constraints to insufficient level of security awareness, which in most cases boils down to lack of organizational maturity.

1. Methodology

This literature review aims to explore the evolution of security practices in the SDLC by synthesizing key research contributions, critically evaluating the strengths and weaknesses of different models, tools, and frameworks aimed at addressing security concerns in software development, focusing particularly on challenges faced in enterprise-grade environments. The goal is to establish a foundation for future work toward measurable, scalable, and integrated security practices across heterogeneous software development contexts.

1.1. Research type

This study employs a qualitative research design, grounded in an interpretive, narrative synthesis of existing scholarly and industry literature on integrating information security into the Software Development Lifecycle (SDLC). Qualitative methods are particularly well-suited for this inquiry because the objective is to understand how and why security practices have evolved, the contextual factors that shape their adoption, and the interplay between technical frameworks and organizational culture. Unlike quantitative meta-analysis, which aggregates numerical findings, qualitative synthesis allows

for a rich, nuanced examination of conceptual developments, practitioner experiences, and the emergent trends—such as DevSecOps and cloud-native security—that defy simple quantification.

The chosen topics span methodological, technical, and socio-cultural dimensions. A qualitative lens enables exploration of these multifaceted aspects in their real-world context. The emergent phenomena should also be mentioned: practices like DevSecOps, which enables continuous security automation, or Software Bill of Materials (SBOMs), which aims at creating a transparent overview of all software components, are relatively recent and not yet amenable to large-scale empirical measurement, however they can be effectively captured through interpretive analysis.

The idea is that by synthesizing diverse sources, the study contributes to theory development, identifying relationships among security frameworks, software development processes, and organizational best practices.

1.2. Literature selection

A comprehensive search was conducted in academic and professional databases, including IEEE Xplore, ACM Digital Library, Scopus, Web of Science, and Google Scholar. Search strings combined terms such as "secure SDLC," "DevSecOps," "threat modeling," and "SBOM." Inclusion criteria required that studies be peer-reviewed, published in reputable outlets between 2005 and 2024, and focused on enterprise-scale software development contexts. Exclusion criteria filtered out articles lacking empirical grounding or practical relevance (for example, purely theoretical cryptography research).

1.3. Data extraction and synthesis

From the initial pool of over 50 records, approximately 20 core studies were selected for detailed review. Information was extracted on key dimensions: security integration models, tooling and automation practices, organizational and cultural enablers, and evaluation metrics. Using an iterative coding process, concepts were grouped into thematic categories, enabling comparative analysis across frameworks such as Microsoft SDL, OWASP SAMM, and NIST SSDF.

1.4. Organizational pattern

The body of the literature review is organized thematically, rather than strictly chronologically. This choice is motivated by several reasons.

Firstly, it makes sense due to comparison of existing frameworks being one of the main targets of this review. By grouping content thematically (e.g., "Frameworks and Methodologies," "Development and Testing Practices," "Emerging Trends"), readers can directly contrast different approaches side-by-side.

Secondly, it's worth noting that themes such as automation, organizational culture, and metrics recur throughout the SDLC phases – a thematic structure brings these connections into focus, allowing to highlight the cross-cutting issues.

It is also meant to facilitate the discovery for enterprise practitioners, who often seek guidance by topic (e.g., "How do I embed threat modeling?") rather than by publication date, making thematic organization more actionable.

While a chronological overview could illustrate the historical evolution of secure SDLC practices, thematic organization better serves the paper's objective of providing a coherent, integrated picture of current best practices and research gaps. This pattern ensures that each major theme is treated in depth, drawing on the full spectrum of relevant literature, regardless of its publication year, provided the recency criterion is still met.

2. Related work

2.1. Development of security challenges in literature

The literature on information security challenges within the enterprise-grade software development lifecycle (SDLC) has evolved significantly, reflecting the increasing complexity and urgency of securing software systems. Early insights from [1] emphasized the tendency to incorporate security measures late in the development process, often resulting in costly fixes post-deployment. They advocated for a paradigm shift towards integrating security considerations throughout all phases of the SDLC, arguing that this proactive approach is essential for safeguarding corporate data and network resources against potential threats.

Building on this foundation, [2] called for a more comprehensive understanding of security

methodologies within software engineering. They highlighted the necessity of educating developers about security risks and mitigation strategies, challenging the perception that security features are merely an additional expense rather than a critical investment in protecting information and customer data. Their survey underscored the need for a cultural change in the industry to prioritize security at every stage of software development.

Authors in [3] further explored security issues in the SDLC, focusing on static analysis and risk management. They examined various tools and methodologies aimed at enhancing software security, advocating for a more analytical approach to identifying vulnerabilities early in the development process. Their work contributed to the discourse on the importance of integrating security practices within software design and testing phases.

Later [4] shifted the focus to practical tools for developers, specifically static code analysis (SCA). He argued that while SCA can effectively identify security flaws, it is often applied too late in the development cycle. It was proposed that integrating security checks within popular integrated development environments (IDEs) would facilitate earlier detection of vulnerabilities, thereby reducing the likelihood of insecure code being deployed.

The focus on developer engagement continued with [5], where the importance of fostering a culture that encourages developers to adopt automated security tools was identified. Authors noted that prior models, such as the Secure Development Lifecycle (SDL), faced resistance due to their prescriptive nature. Their research suggested a shift towards more flexible Security Capability Maturity Models, emphasizing the alignment of security practices with business goals and the need for lightweight best practices that resonate with developers' realities.

Authors in [6] expanded on the importance of security integration throughout the SDLC, advocating for regular code reviews and the use of automated tools to identify vulnerabilities. They emphasized that security checks should not be confined to the testing phase but should be a continuous process throughout development. Their insights reinforced the necessity of a structured approach to security that includes penetration testing and adherence to security standards during deployment and maintenance.

In the same year, [7] highlighted the role of periodic security testing and the need for ongoing developer education in secure coding practices. They pointed out that the integration of automated security tools can significantly lighten the workload of security engineers and improve the security posture of software projects. Their findings underscored the critical role of training and mentorship in fostering a culture of security awareness among developers.

Authors in [8] addressed the emerging DevSecOps paradigm, which aims to incorporate security into the agile development process. Their systematic review identified numerous challenges faced by practitioners in adopting DevSecOps, including the need for automation and the balance between rapid delivery and security. They called for a greater focus on developer-centric security tools to facilitate this integration.

Most recently, [9] addressed the ongoing challenges in secure software development, emphasizing the need for continuous security validation throughout the coding and testing phases. They advocated for thorough security assessments and the implementation of robust security measures before software deployment. Their work encapsulated the evolution of secure software development practices, stressing that regular code reviews and updates are essential to mitigate emerging vulnerabilities.

2.2. The importance of SSDLC

Together, these articles illustrate a clear trajectory in the literature, from recognizing the necessity of integrating security throughout the SDLC to advocating for specific methodologies and tools that facilitate this integration. The collective insights highlight the persistent challenges and evolving strategies in addressing information security within enterprise-grade software development, underscoring the importance of a proactive and comprehensive approach to safeguarding software systems.

Several studies underscore the importance of integrating security from the outset of the SDLC. Davis [10] argues that information security should be "built in" rather than "bolted on" after development, emphasizing the idea of a Secure Software Development Lifecycle (SSDLC). This perspective shifts the traditional approach to security, which often treats it as an afterthought, toward a proactive stance where security is an integral part of every phase of the SDLC, as illustrated in Figure 1.

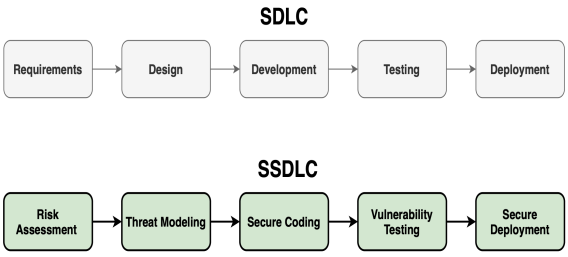

Figure 1: SDLC vs. SSDLC comparison

This approach is further reinforced by Jones and Rastogi in [11], where it is suggested that earlier and more frequent security interventions can drastically reduce vulnerabilities in the final product.

Newton et al. [12] also argue that introducing security as an early-stage consideration within Agile projects reduces long-term complexity and cost. NIST further supports this view by recommending security activities be initiated from the concept phase through to deployment [13].

The integration of security into the SDLC is not only about technical practices but also involves adopting frameworks that guide developers through secure processes. NIST's Secure Software Development Framework (SSDF) is a notable standard that offers a flexible and comprehensive approach to integrating security practices across SDLC stages [14]. SSDF consists of practices grouped into four categories: Prepare the Organization, Protect the Software, Produce Well-Secured Software, and Respond to Vulnerabilities.

Davis [10] and Saeed et al. [15] note that although frameworks like SSDF offer structure, they often demand specialized skills and consistent oversight. These practices can be demanding for smaller organizations with limited security budgets or staff, which presents a challenge to widespread adoption.

2.3. Security practices in development and testing phases

During the development phase, secure coding practices are a primary focus. Saeed et al. [6] highlight the use of OWASP's Top Ten vulnerabilities as a key resource for development teams to ensure secure application design. Tools like Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) offer automated analysis to detect vulnerabilities both during coding and runtime.

However, SAST tools are often context-dependent and may not capture complex runtime issues, while DAST tools require skilled manual oversight, making them more resource-intensive. Integrating DAST into CI/CD pipelines ensures that security testing is not an isolated post-development activity but part of the continuous development process [6].

2.4. Emerging trends: DevSecOps, Cloud Security, SBOMs

Recent research spotlights the DevSecOps paradigm, which addresses the issue of the so-called "knowledge silos" by treating security-related issues as a shared responsibiliy across development, operations, and security teams. Case studies demonstrate that embedding automated security checks within CI/CD processes fosters a "shift-left" culture, reducing the latency between code commit and vulnerability detection This approach fosters a collaborative environment that helps reconcile the traditional tension between rapid deployment and stringent security controls [14], [15]. However, adoption challenges persist. As noted in [17], cultural resistance, skill gaps, and tooling fragmentation often slow implementation.

In parallel, the move to cloud-native architectures introduces novel attack surfaces. Organizations must now secure not only their application code but also infrastructure-as-code, container images, and managed services. Cloud security presents an evolving challenge in modern SDLCs. The shared responsibility model in cloud environments, as defined by NIST [13], requires software vendors to take increased responsibility for application-layer security. Practices such as secure API management, access control enforcement, and encryption standards are essential for ensuring robust cloud-native application security [14].

An increasingly critical component in this domain is the Software Bill of Materials (SBOM), which improves supply chain transparency. As Nicolaysen notes in [16], SBOMs are essential for tracking third-party component vulnerabilities, particularly considering recent attacks like SolarWinds and Log4Shell. Integration of SBOM generation into build pipelines ensures real-time visibility but poses challenges in version tracking and component licensing. The obvious benefit of SBOMs is the transparency of included software components. The components include build or

runtime dependencies, which can be presented as a graph, shown in Figure 2.

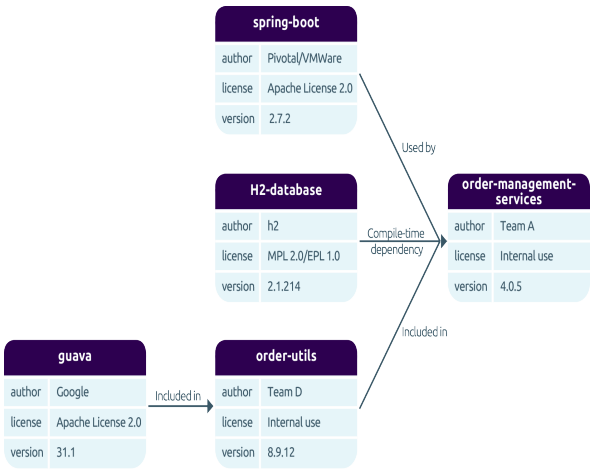

Figure 2: SBOM representation of a typical Spring Boot microservice

2.5. Critical evaluation of existing approaches

Still the limitations remain, despite the progress enabled by frameworks like NIST SSDF and OWASP SAMM. This boils down to approaches being fundamentally similar; distinct parallels can be drawn between the two frameworks, as outlined by Figure 3.

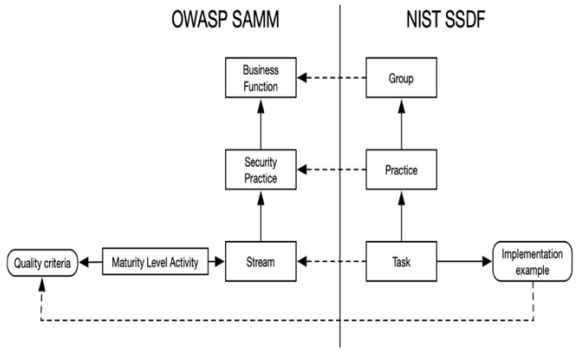

Figure 3: Structural comparison between OWASP SAMM and NIST SSDF

One major limitation is the absence of standardized security metrics to measure the effectiveness of implemented controls [15]. Without these metrics, organizations struggle to evaluate the maturity of their secure development practices.

Furthermore, resource and expertise constraints limit the practical application of these frameworks, especially in small and medium-sized enterprises (SMEs). Jones and Rastogi [11] observe that many secure development guidelines assume a baseline of cybersecurity knowledge that is not always present in Agile teams. Continuous education and awareness campaigns are necessary to mitigate these gaps.

3. Results and discussion

The review of literature on information security challenges in the enterprise-grade SDLC reveals a coherent progression of research focus and practical guidance. Across these bodies of work, a few clear results emerge.

First, embedding security at every phase of the SDLC demonstrably reduces both the frequency and severity of post-deployment vulnerabilities. Studies consistently report lower remediation costs and fewer high-risk defects when threat modeling, secure coding standards, and automated scans are performed continuously rather than in isolated stages [12], [15].

Second, frameworks such as NIST's SSDF and OWASP's SAMM, while differing in structure and scope, converge on key practices—talent enablement, process integration, and continuous feedback—that underpin any robust SSDLC. Their differences suggest that there is no silver bullet; instead, organizations must tailor these models to match their risk profiles, resource constraints, and cultural readiness.

Third, automation is indispensable. Tools that enforce security checks at commit time, integrated within CI/CD pipelines, relieve security teams of repetitive reviews and cultivate a "security-first" mindset among engineers. Yet tool adoption alone is insufficient without complementary training, incentives, and leadership support.

Reflecting on these results, it becomes evident that the technical solutions – frameworks, tools, scans – act as necessary, but not sufficient enablers. The real differentiator is organizational maturity: leadership commitment to secure development, investment in human capital, and willingness to measure security outcomes as rigorously as functional metrics. This observation opens a path for further inquiry: How can enterprises quantify security maturity in a standardized way that is both meaningful and actionable? What incentives and governance mechanisms best sustain cross-functional collaboration over time? Moreover, as cloud-native and microservices architectures proliferate, we must ask whether existing SDLC models can adapt quickly enough, or whether new paradigms are required, such as policy-as-code and real-time risk telemetry.

These questions, which emerge from the synthesis of analyzed literature, highlight that the challenges of secure SDLC are less about technical insufficiency and more about organizational maturity. While frameworks provide valuable scaffolding, they often assume resources and expertise unavailable to many enterprises. The lack of standardized metrics further obstructs adoption, leaving organizations unable to measure return on investment or compare maturity levels.

This newfound perspective enables us to propose a three-pillar conceptual model, visualized on Figure 4, to guide future SSDLC evolution. Each pillar can be described as follows:

1. Automation – Continuous integration of SAST, DAST, and SBOM tools in CI/CD pipelines.
2. Framework Integration – Flexible adoption of SDL, SAMM, or SSDF tailored to organizational risk profiles.
3. Organizational Maturity – Leadership buy-in, continuous training, and cultural incentives ensuring sustained adoption.

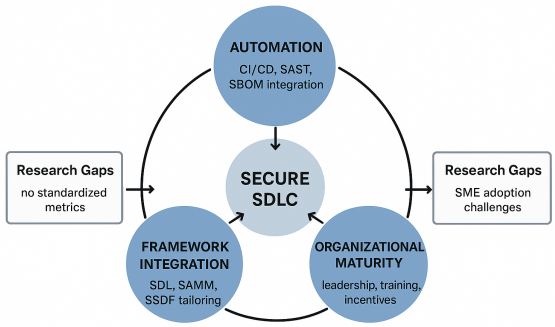

Figure 4: The three pillars of SSDLC

This model reframes SSDLC not as a checklist of practices, but as a balanced system where technical tools and human enablers reinforce one another.

Conclusions and future work

The landscape of securing enterprise-grade software through the SDLC has matured from ad hoc, end-of-cycle patching to structured, continuous integration of security controls. Key takeaways from our review include the criticality of weaving security requirements into every phase of development, the utility of flexible yet prescriptive frameworks (such as NIST's SSDF and OWASP's SAMM), and the transformative potential of DevSecOps practices and SBOM-driven supply-chain transparency. Notably, successful programs marry technical automation with organizational enablers—leadership buy-in, targeted training, and metrics that elevate security as a first-class development criterion.

Looking forward, three primary avenues merit deeper investigation. First, there is an urgent need to establish standardized, quantitative metrics for security maturity that are both lightweight enough for SMEs and robust enough for large enterprises. Without a common measurement vocabulary, it remains difficult to compare, benchmark, or demonstrate the ROI of secure-SDLC investments. Second, research should explore adaptive security frameworks that dynamically reconfigure controls based on real-time threat intelligence and deployment context—bridging the gap between static process models and highly agile, cloud-native architectures. Third, the human dimension warrants more empirical scrutiny: studies of incentive structures, team dynamics, and cultural interventions can reveal which organizational levers most effectively sustain long-term adoption of security practices.

Ultimately, the goal is to converge on security approaches that are not only scalable (i.e. capable of spanning monolithic and microservices ecosystems), but also measurable, so that teams can track progress and adjust course. Future work must therefore deliver methods and tools that integrate seamlessly into modern CI/CD pipelines, support continuous feedback loops, and empower practitioners to deliver secure code with the same velocity and reliability expected in today's enterprise software landscape.

References

[1] P. Falcarin, M. Morisio, P. Falcarin, and M. Morisio, "Developing Secure Software and Systems," 2004. [PDF]

[2] A. Uzunov, E. Fernandez, and K. Falkner, "Engineering security into distributed systems: a survey of methodologies," 2012.

[3] N. Nazir and M. Kashif Nazir, "A Review of Security Issues in SDLC," 2018. [PDF]

[4] Meng, X., 2018. Static Analysis of Android Secure Application Development Process with FindSecurityBugs. [PDF]

[5] C. Weir, I. Becker, J. Noble, L. Blair et al., "Interventions for Long Term Software Security:Creating a Lightweight Program of Assurance Techniques for Developers," 2020.

[6] Chin Eian, I., Ka Yong, L., Yeap Xiao Li, M., Affan Bin Noor Hasmaddi, N., & Fatima-

tuz-Zahra, undefined, 2020. Integration of Security Modules in Software Development Lifecycle Phases. [PDF]

[7] Davaindran Lingham, A., Tang Kwong Kin, N., Wan Jing, C., Heng Loong, C., & Fatima-tuz-Zahra, undefined, 2020. Implementation of Security Features in Software Development Phases. [PDF]

[8] R. N. Rajapakse, M. Zahedi, M. Ali Babar, and H. Shen, "Challenges and solutions when adopting DevSecOps: A systematic review," 2021. [PDF]

[9] Wen Ping, S., Cheok Jun Wah, J., Wen Jie, L., Bong Yong Han, J., & Muzafar, S., 2023. Secure Software Development: Issues and Challenges. [PDF]

[10] N. Davis, Secure Software Development Life Cycle Processes, Carnegie Mellon University: Software Engineering Institute, CMU/SEI-2013-TN-XX, 2013.

[11] R. L. Jones and A. Rastogi, "Secure coding: building security into the software development life cycle," Information Security Journal: A Global Perspective, vol. 13, no. 5, pp. 29–39, 2004.

[12] N. Newton, C. Anslow, and A. Drechsler, "Information security in agile software development projects: a critical success factor perspective," in Proc. 27th European Conference on Information Systems (ECIS 2019), Stockholm-Uppsala, Sweden, Jun. 2019.

[13] National Institute of Standards and Technology (NIST), Security Considerations in the System Development Life Cycle, NIST SP 800-64 Rev.2, Gaithersburg, MD, 2008.

[14] M. Souppaya, K. Scarfone, and D. Dodson, Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities, NIST SP 800-218, National Institute of Standards and Technology, Gaithersburg, MD, 2022.

[15] H. Saeed, I. Shafi, J. Ahmad, A. A. Khan, T. Khurshaid, and I. Ashraf, "Review of techniques for integrating security in software development lifecycle," Computers, Materials & Continua, vol. 82, no. 1, pp. 139–172, 2024, doi: 10.32604/cmc.2024.057587.

[16] Nicolaysen, T., Sasson, R., Line, M.B., Jaatun, M.G.: Agile software development:The straight and narrow path to secure software? International Journal of SecureSoftware Engineering (IJSSE) 1(3), 71–85 (2010)

[17] G. McGraw, Software security: Building Security in. Addison-Wesley Professional, 2006.