

# Journal of Information and Cybersecurity

## ARTICLE INFO

**Article History:** Received: 24-05-2026, Revised: 20-07-2026, Accepted: 22-08-2026

**\*Corresponding author Email:** [angsokroeurn.phdscholar@lincoln.edu.my](mailto:angsokroeurn.phdscholar@lincoln.edu.my)

**DOI:**

This is an open access article under the CC BY 4.0 license

(<http://creativecommons.org/licenses/by/4.0/>).

Published by ITAP Publisher.



Vol 2026, No.1

## Bridging NIST 800-64 Standards with Modern DevSecOps: A Comprehensive Review for Integrating Automated SAST, DAST, and SIEM/SOAR into the Secure SDLC

Sokroeurn Ang<sup>1\*</sup>, Mony Ho<sup>1</sup>, Sopheatra Huy<sup>1</sup>, Midhunchakkaravarthy Janarthanan<sup>1</sup>

<sup>1</sup> AI Computing and Multimedia Department, Lincoln Graduate Program, Doctor in Cybersecurity, Lincoln University College, Selangor, Malaysia

### Abstract

In a global threat landscape increasingly dominated by AI-driven exploits and sophisticated supply chain attacks, traditional reactive security models fail to provide adequate protection. While NIST Special Publication 800-64 offers a rigorous foundation for high-assurance environments, it is a critical mapping of the NIST standard to modern DevSecOps pipelines. The primary challenge lies in reconciling documentation-heavy governance with the rapid iteration requirements of contemporary software engineering. This study employs a qualitative systematic literature review methodology, synthesizing a comprehensive range of peer-reviewed research and international standards to develop a modernized secure SDLC roadmap. The research demonstrates that a secure-by-design posture is achievable through the total integration of automated security gates within CI/CD pipelines. Statistical synthesis from the reviewed literature indicates that shifting security to the earliest phases of development can reduce remediation costs by up to 60 times compared to post-deployment fixes. Furthermore, the integration of AI-enhanced SIEM and SOAR platforms facilitates real-time threat telemetry, reducing the Mean Time to Respond by 40-50%. Technical verification also confirms that the concurrent application of static and dynamic application security testing significantly increases vulnerability coverage, identifying over 85% of potential flaws. The study provides a novel, phase-by-phase operational roadmap that maps NIST 800-64 principles to modern automation tools such as Snyk, SonarQube, and Splunk. By establishing a bridge between regulatory rigor and DevOps agility, this research offers a scalable blueprint for organizations seeking to maintain high-assurance security standards in software security development environments.

**Keywords:** Secure SDLC, NIST SDLC, DevSecOps, Cybersecurity, NIST 800-64, DevSecOps Automation, and Software Testing.

### 1. Introduction

Cybersecurity must be integrated into every phase of the system development process to counter the escalating complexity of modern cyberthreats. The primary benefit of embedding security into the System Development Life Cycle (SDLC) is the early identification and mitigation of vulnerabilities; a fundamental principle which is significantly more cost-effective than introducing security at a later stage [11], [18]. This concept serves as the historical cornerstone of the National Institute of Standards and Technology (NIST) guidelines, specifically NIST Special Publication 800-64 [2]. While NIST 800-64

remains the foundational structural benchmark for high-assurance environments, modern engineering requires its synthesis with the more recent NIST SP 800-218 (Secure Software Development Framework - SSDF) to address contemporary challenges such as software supply chain integrity and automated pipeline security [3], [58], [61].

In the contemporary landscape of frequent and sophisticated cyber-attacks, treating cybersecurity as an afterthought activity is no longer a viable strategy for software security [4], [6], [10]. Research indicates that the benefits of early integration are substantial; remediating a vulnerability during the design phase is significantly less costly than addressing the same flaw after deployment [11]. Despite these advantages, many organizations continue to struggle with the transition from a reactive model to a proactive, secure-by-design methodology [7], [13]. To bridge this gap, various frameworks have been developed to provide a structured approach to the Secure SDLC. While models such as the Microsoft SDL and OWASP SAMM are widely utilized, the NIST framework remains the gold standard due to its rigorous emphasis on risk management, compliance, and multidisciplinary engineering [14], [15], [40]. The transition to a "secure-by-design" methodology is best achieved by following the five distinct phases of the NIST SDLC [2]. This process begins with the Initiation Phase, where security roles are identified and formal risk assessments are conducted to ensure long-term organizational resilience [2]. As the cycle moves into the Development/Acquisition Phase, functional security testing becomes critical for analyzing security architecture [3], [37]. Following this, the Implementation/Assessment Phase requires the configuration and testing of security features before moving to production [1], [2]. Once live, the Operation/Maintenance Phase focuses on user authentication and Configuration Management (CM) to assess the impact of changes on the system's security posture [18], [41]. Finally, the Disposal Phase ensures that storage media are purged to prevent unauthorized data recovery [2], [3].

In modern software engineering, DevOps integrates development and operations through automated CI/CD pipelines to accelerate delivery. In the era of AI-driven exploits, the manual oversight traditionally associated with NIST phases is no longer sufficient. This necessitates the evolution toward Intelligent DevSecOps, where machine learning-enhanced SAST and DAST tools provide real-time telemetry, allowing the 'Roadmap' to function as an automated living system rather than a static set of rules [65]. DevSecOps extends this paradigm by embedding security into every phase, transforming it into a shared responsibility [21], [56], [60]. To secure the SDLC, this study recommends a total roadmap for integrating cybersecurity through automated DevSecOps pipelines. This involves performing comprehensive security testing via Static Application Security Testing (SAST) for source code analysis and Dynamic Application Security Testing (DAST) for runtime analysis [36], [57]. Furthermore, as software supply chains become primary targets, the integration of a Software Bill of Materials (SBOM) is now essential for transparency across third-party components and reducing insecure custom development, leveraging vendor security maturity, and enforcing standardized by introducing COTS [62]. For continuous monitoring, the implementation of Web Application Firewalls (WAF) alongside Security Information and Event Management (SIEM) and Security Orchestration, Automation, and Response (SOAR) is vital for automated threat mitigation [59], [63]. Finally, conducting a cybersecurity audit at least annually remains essential for detecting evolving control weaknesses. By analyzing the strengths and weaknesses of various models in relation to NIST standards, this study provides a comprehensive roadmap for achieving a secure, efficient, and cost-effective development lifecycle in an increasingly volatile digital environment [33], [40], [55].

### *1.1 Problem Statement*

Although NIST Special Publication 800-64 provides a rigorous foundation for securing high-assurance systems, its reliance on manual, documentation-heavy governance is increasingly misaligned with the automation and speed of modern DevSecOps environments [2], [40]. This creates a compliance–agility paradox in which security is perceived as a barrier to rapid delivery rather than an embedded quality attribute [64]. The challenge is intensified by AI-driven attacks and supply chain threats that exceed the effectiveness of manual oversight. While NIST 800-64 defines required security objectives, it offers limited guidance on operationalizing them through automated CI/CD pipelines, security-as-code, and real-time telemetry [62], [65]. Consequently, organizations struggle to translate NIST requirements into actionable controls such as automated SAST, DAST, and SIEM/SOAR, resulting in fragmented, reactive security and increased exposure to evolving threats.

### *1.2 Research Objectives and Contributions*

The primary aim of this research is to comparatively study to secure SDLC, the study discusses the following specific objectives:

- To synthesize NIST 800-64 lifecycle phases with contemporary automated security orchestration, specifically Static and Dynamic Application Security Testing (SAST and DAST) [8], [36], [57].

- To operationalize a modernized Secure SDLC roadmap that replaces manual phase-gates with continuous, automated security checkpoints [1], [2].
- To identify and quantify the economic and operational impact of integrating real-time threat telemetry and automated incident response through SIEM and SOAR platforms [59], [63].
- To establish a hybrid governance model that maintains high-assurance compliance without impeding the speed of cloud-native development pipelines [33], [41].

### 1.3 Research Contributions

This study provides several key contributions to the field of secure software engineering and cybersecurity governance:

- The NIST-DevSecOps Integration Matrix: A comprehensive mapping that translates abstract NIST 800-53 and 800-64 controls into specific, automated technical tools and CI/CD triggers [2], [41].
- Evidence-Based Return on Security Investment (ROSI): A thematic synthesis of an extensive range of peer-reviewed literature that validates the cost-effectiveness of the shift-left paradigm, demonstrating significant reductions in remediation costs and recovery times [11], [18].
- The Secure-by-Design Blueprint: A transformative roadmap for practitioners and researchers that demonstrates how security can function as a continuous, shared quality attribute rather than a terminal compliance activity [7], [13].
- Framework Validation: A systematic analysis of how concurrent testing methodologies and Software Bill of Materials (SBOM) generation effectively mitigate modern supply chain risks [62].

## 2. Literature Review

The evolution of Secure Software Development Lifecycles reflects a fundamental shift from reactive, documentation-centric compliance models toward proactive, automation-driven security engineering [1], [5], [6]. Contemporary research increasingly recognizes that cybersecurity can no longer be treated as a discrete phase or post-deployment activity, but rather as a continuous, lifecycle-wide quality attribute embedded within modern development practices [7], [11], [18]. Through this synthesis, unresolved gaps in existing research are identified, motivating the need for an integrated, NIST-aligned DevSecOps roadmap.

**Table 1.** Phase-by-Phase NIST-DevSecOps Integration Matrix.

| NIST 800-64 Phase          | DevSecOps / CI/CD Integration                                                    | Cybersecurity Tools & Tests                                    | Governance & Audit Mechanism                             | Justification / Reference                                                                    |
|----------------------------|----------------------------------------------------------------------------------|----------------------------------------------------------------|----------------------------------------------------------|----------------------------------------------------------------------------------------------|
| 1. Initiation              | Pre-Commit / Planning: Security requirements as "Security-as-Code" user stories. | SBOM Generation, Threat Modeling, Risk Assessment.             | Verification of security roles; formal risk acceptance.  | SBOM ensures 3rd-party transparency early in the cycle [62].                                 |
| 2. Development             | CI Build Pipeline: Automated security triggers on every code commit.             | SAST (Static Analysis), IDE Security Plugins, Secret Scanning. | Verify security vulnerability and remediation.           | SAST facilitates "Shift-Left" by identifying proprietary flaws before compilation [8], [11]. |
| 3. Implementation          | CD Deployment: Automated staging/production push with security gates.            | DAST (Dynamic Analysis), IAST, Container Scanning, WAF Config. | Configuration Management (CM) audit; Runtime validation. | DAST validates runtime security and environmental configurations [1], [36], [57]             |
| 4. Operation & Maintenance | Continuous Monitoring: Real-time telemetry feedback loop to developers.          | SIEM & SOAR Integration, AI-driven threat hunting.             | MTTR Reduction; Automated incident response playbooks.   | SOAR/SIEM integration provides "Active Orchestration" for AI-driven threats [59], [63].      |

|             |                                                                    |                                                                                   |                                                                  |                                                                                                            |
|-------------|--------------------------------------------------------------------|-----------------------------------------------------------------------------------|------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| 5. Disposal | De-provisioning:<br>Automated<br>Infrastructure-as-<br>Code (IaC). | Media Sanitization<br>(Purge,<br>Cryptographic<br>Erase, Physical<br>Destruction) | Final compliance<br>certificate; Data<br>integrity verification. | Prevents "Data<br>Remanence" via NIST<br>800-88 standards<br>during hardware<br>decommission [2],<br>[61]. |
|-------------|--------------------------------------------------------------------|-----------------------------------------------------------------------------------|------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|

### 2.1 Comparative Analysis of Secure SDLC Frameworks

Early research on secure software engineering emphasized the multidisciplinary nature of building trustworthy systems, integrating risk management, requirements engineering, and architectural design into traditional SDLC models [1], [5], [34]. Foundational studies established that security must be systematically engineered rather than retrofitted, forming the conceptual basis for modern Secure SDLC frameworks [6], [7]. NIST Special Publication 800-64 formalized this perspective by defining a five-phase lifecycle—Initiation, Development/Acquisition, Implementation/Assessment, Operation/Maintenance, and Disposal—designed to support high-assurance and compliance-driven environments [2]. Alongside NIST, alternative frameworks such as the Microsoft Security Development Lifecycle (SDL) and OWASP Software Assurance Maturity Model (SAMM) gained widespread adoption [4], [14], [15]. Microsoft SDL emphasizes prescriptive security activities embedded into development workflows, while OWASP SAMM adopts a maturity-based approach that allows organizations to incrementally improve security practices [17]. Other specialized methodologies, such as SQUARE, focus on security requirements engineering and early risk identification [9]. Comparative analyses consistently show that while these models offer valuable guidance, they differ substantially in scope, prescriptiveness, and alignment with regulatory and compliance requirements [12], [16], [40]. Despite their strengths, the literature highlights a shared limitation across Secure SDLC frameworks: most were designed for relatively static development environments characterized by infrequent releases and manual oversight [20], [21]. NIST SP 800-64, while rigorous and comprehensive, assumes documentation-heavy governance that does not naturally align with automated CI/CD pipelines [2], [40]. Conversely, maturity-based and process-centric models often lack explicit operational mappings to security tooling and runtime enforcement mechanisms [14], [17]. Recent studies advocating alignment between NIST 800-64 and the Secure Software Development Framework (SSDF, NIST SP 800-218) acknowledge the need for automation and supply chain security but stop short of providing a unified, phase-by-phase operational blueprint [3], [58], [61]. This study reveals a critical gap between high-assurance standards and the realities of modern, cloud-native software engineering.

**Table 2.** Comparative Secure SDLC Framework Analysis.

| Framework           | Security Integration Model      | SDLC Enforcement Mechanism           | Operational & Supply Chain Coverage           | Regulatory / Standards Suitability      |
|---------------------|---------------------------------|--------------------------------------|-----------------------------------------------|-----------------------------------------|
| NIST 800-64         | Risk-driven, compliance-centric | Manual phase-gate controls           | Limited (minimal runtime & SBOM focus)        | High (federal & high-assurance systems) |
| NIST SSDF (800-218) | Secure-by-design principles     | Policy-driven, partially automated   | Strong supply-chain focus, limited operations | Medium (guideline-oriented)             |
| Microsoft SDL       | Developer-centric security      | Prescriptive development practices   | Low (weak runtime & supply-chain coverage)    | Low–Medium (enterprise best practice)   |
| OWASP SAMM          | Maturity-based improvement      | Self-assessed process maturity       | Low (tool-agnostic, limited operations)       | Low (non-regulatory)                    |
| Proposed Framework  | Lifecycle-integrated DevSecOps  | Automated security-as-code pipelines | End-to-end (SBOM, SIEM, SOAR)                 | NIST alignment                          |

### 2.2 Cost effective for secure SDLC

A central consensus in the literature is that the timing of security integration has a direct and measurable impact on development cost and long-term sustainability [11], [18]. Empirical studies consistently demonstrate that vulnerabilities identified during the initiation or design phases are significantly less expensive to remediate than those discovered after

deployment [11], [18]. This economic principle underpins the widely adopted “shift-left” paradigm, which advocates embedding security requirements, threat modeling, and risk assessments at the earliest stages of the SDLC [19], [25]. The concept of Return on Security Investment (ROSI) is frequently cited as a primary driver for early security integration [19], [51]. By providing cost effective, minimizing rework, and lowering incident response costs, shift-left practices contribute to both financial efficiency and organizational resilience [2], [11]. Establishing security roles, formal risk assessments, and architectural controls during the initiation phase has been shown to reduce downstream remediation effort and operational disruption [2], [18]. However, the literature also reveals limitations in existing economic analyses. Many cost-benefit models assume mature security tooling, skilled personnel, and organizational readiness that may not exist uniformly across industries or organizational sizes [11], [18]. In particular, small-to-medium enterprises and highly regulated sectors often face resource constraints that complicate the adoption of early-stage security practices [51], [53]. These findings suggest that while the economic rationale for shifting left is theoretically robust, its practical realization depends heavily on automation, standardization, and lifecycle-aligned governance—dimensions that remain insufficiently addressed in current Secure SDLC research.

### 2.3 DevSecOps

The emergence of DevSecOps represents a paradigm shift in which security is embedded as a shared responsibility across development, operations, and security teams [20], [21], [56]. By integrating security controls directly into CI/CD pipelines, DevSecOps aims to reconcile rapid deployment cycles with robust security assurance [22], [60]. Nevertheless, the literature identifies a persistent “compliance–agility paradox,” wherein the velocity of Agile and DevOps practices conflicts with the documentation and control requirements imposed by regulatory and standards-based frameworks [60], [64]. Empirical studies of practitioner experiences indicate that security is often perceived as a bottleneck unless it is fully automated and seamlessly integrated into development workflows [20], [21]. To address this challenge, contemporary research increasingly advocates for “security-as-code,” whereby policies, controls, and risk checks are expressed as machine-readable artifacts enforceable through automated pipelines [24], [25], [26], [43]. This approach enables continuous compliance by transforming governance requirements into executable controls [44], [56]. Despite growing consensus on security-as-code, the literature lacks a standardized, NIST-aligned operational model that translates high-level compliance objectives into enforceable CI/CD security gates across the entire SDLC [44], [47], [64]. As a result, many organizations implement fragmented or tool-centric solutions that fail to achieve holistic lifecycle integration, underscoring the need for a roadmap that preserves NIST rigor while enabling DevSecOps agility.

### 2.4 Technical Verification: Comparative AST Methodologies

Application Security Testing (AST) constitutes a core technical pillar of Secure SDLC implementations [6], [30], [32]. The literature extensively examines the strengths and limitations of Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST). SAST is widely recognized for its effectiveness in identifying source-code vulnerabilities early in the development lifecycle, supporting shift-left objectives and reducing remediation costs [8], [37]. However, static analysis is inherently limited in its ability to detect runtime configuration issues, authentication flaws, and environment-specific vulnerabilities [36], [57]. DAST complements static analysis by evaluating applications in their running state, enabling the detection of vulnerabilities related to session management, input validation, and deployment configurations [30], [36]. Comparative studies consistently conclude that neither SAST nor DAST is sufficient in isolation [13], [57]. Instead, hybrid AST approaches that combine static, dynamic, and interactive testing techniques provide superior vulnerability coverage and reduce blind spots across the SDLC [38], [45], [46], [55], [57]. Nevertheless, the literature also identifies persistent challenges associated with AST automation, including false-positive rates, tool interoperability, and scalability in microservices and containerized architectures [13], [55]. While hybrid testing improves detection efficacy, few studies explicitly address how AST tools can be systematically governed and mapped to lifecycle phases within standards-based frameworks such as NIST 800-64 [2], [40]. This disconnect between technical testing capabilities and formal lifecycle governance reinforces the need for an integrated, standards-aligned roadmap.

### 2.5 Operational Resilience: SIEM, SOAR, and Continuous Monitoring

The Operation and Maintenance phase of the Secure Software Development Lifecycle has evolved substantially in response to the increasing sophistication, automation, and velocity of contemporary cyberthreats [2], [10]. Traditional security approaches—characterized by periodic log reviews, manual incident handling, and static configuration management—have proven inadequate for defending against modern attack vectors such as web application exploits, advanced persistent threats, and AI-assisted attack automation [23], [31], [65]. As a result, contemporary literature emphasizes the necessity of

continuous monitoring mechanisms that provide real-time visibility and adaptive defense capabilities throughout system operation [41], [59]. Security Information and Event Management (SIEM) systems constitute the foundational layer of operational security by aggregating, normalizing, and correlating security events across heterogeneous infrastructures [41], [50]. SIEM platforms enable centralized visibility, compliance reporting, and forensic analysis, making them indispensable for organizations operating in regulated or high-assurance environments [40], [41]. However, multiple studies highlight inherent limitations of SIEM-only deployments, particularly the generation of excessive alerts, limited contextual awareness, and delayed response caused by analyst fatigue and manual triage processes [59], [63].

To address these shortcomings, Security Orchestration, Automation, and Response (SOAR) platforms have emerged as a critical complement to SIEM. SOAR systems extend visibility into actionable resilience by automating incident response workflows through predefined playbooks, policy-driven actions, and integration with threat intelligence sources [59], [63]. Empirical studies demonstrate that SIEM–SOAR integration significantly reduces Mean Time to Detect (MTTD) and Mean Time to Respond (MTTR), enabling organizations to counter high-velocity and AI-driven attacks that exceed the speed of manual intervention [63], [65]. Consequently, recent literature increasingly recognizes SIEM–SOAR integration as a baseline capability for modern Secure SDLC implementations, particularly in cloud-native, containerized, and microservices-based architectures [59], [63], [65]. Despite extensive research on SIEM and SOAR within operational security and incident response domains, their systematic integration into Secure SDLC frameworks remains underexplored [41], [42]. Most existing studies treat monitoring and response as post-deployment concerns rather than as continuous feedback mechanisms capable of informing earlier SDLC phases such as design, development, and security testing. This separation limits the ability of telemetry-driven insights to refine threat models, improve application security testing strategies, and guide architectural hardening decisions across the lifecycle [49], [52]. Accordingly, the literature reveals a critical gap in which operational security telemetry is insufficiently leveraged as a core component of secure-by-design and resilient-by-design software engineering.

## 2.6 Supply Chain Integrity and Modern Governance

The increasing reliance on third-party, open-source, and cloud-native components has fundamentally expanded the software attack surface, positioning software supply chains as one of the most critical risk vectors in modern Secure Software Development Lifecycles [54], [62]. Contemporary cyber incidents demonstrate that adversaries increasingly exploit vulnerabilities in upstream dependencies rather than directly targeting proprietary codebases. As a result, recent literature has shifted from isolated application security toward holistic supply chain risk management spanning the entire SDLC [27], [28], [29], [54]. To proactively address supply chain risks, researchers have proposed predictive modeling, software metrics, and text-mining techniques to identify vulnerable components prior to integration [27], [28], [29]. These approaches leverage historical vulnerability data and code artifacts to support early risk detection and informed component selection. While effective at improving visibility, such techniques are often applied in isolation and lack standardized integration with Secure SDLC governance frameworks, limiting their operational scalability and consistency across development phases [28], [54]. The emergence of the Software Bill of Materials (SBOM) has become a foundational governance mechanism for enhancing transparency, traceability, and accountability across software supply chains [62]. By formally enumerating all software components, dependencies, and their relationships, SBOMs enable organizations to rapidly assess exposure to newly disclosed vulnerabilities and supply chain compromises. Recent studies emphasize that SBOM generation must occur early in the SDLC and be continuously maintained to support dynamic dependency management and automated vulnerability correlation [62].

However, the literature indicates that SBOM adoption is frequently treated as a static compliance artifact rather than as a living control integrated across all NIST SDLC phases [2], [62]. Beyond technical controls, governance and assurance mechanisms remain essential for validating the effectiveness of automated security practices. Formal cybersecurity audits are widely recommended as a means of identifying systemic weaknesses, policy gaps, and architectural flaws that automated tools may fail to detect [33], [35], [41]. In high-assurance and regulated environments, periodic audits serve as a critical validation layer, ensuring alignment with standards such as NIST 800-64 and broader risk management requirements [2], [40]. Nevertheless, existing literature provides limited guidance on how audit findings can be systematically reintegrated into automated DevSecOps pipelines, resulting in a persistent disconnect between compliance validation and continuous security improvement [40], [48]. Collectively, the literature underscores that effective supply chain security cannot be achieved through isolated tooling or point-in-time assessments. Instead, it requires a lifecycle-integrated governance model in which SBOM management, predictive risk analysis, and formal audits operate as continuous feedback mechanisms within the Secure SDLC. The absence of such an integrated governance approach

represents a critical gap in current research and practice, reinforcing the need for a comprehensive roadmap that aligns supply chain integrity, automated DevSecOps controls, and NIST-based assurance principles.

### 3. Methodology

This study adopts a rigorous, multi-stage qualitative research methodology grounded in a Systematic Literature Review (SLR) and comparative synthesis to develop a modernized Secure Software Development Lifecycle framework based on figure 1. The primary objective of this methodology is to address the research gaps, the lack of an integrated operational mapping between NIST 800-64, automated DevSecOps practices, application security testing, operational telemetry, and governance mechanisms.

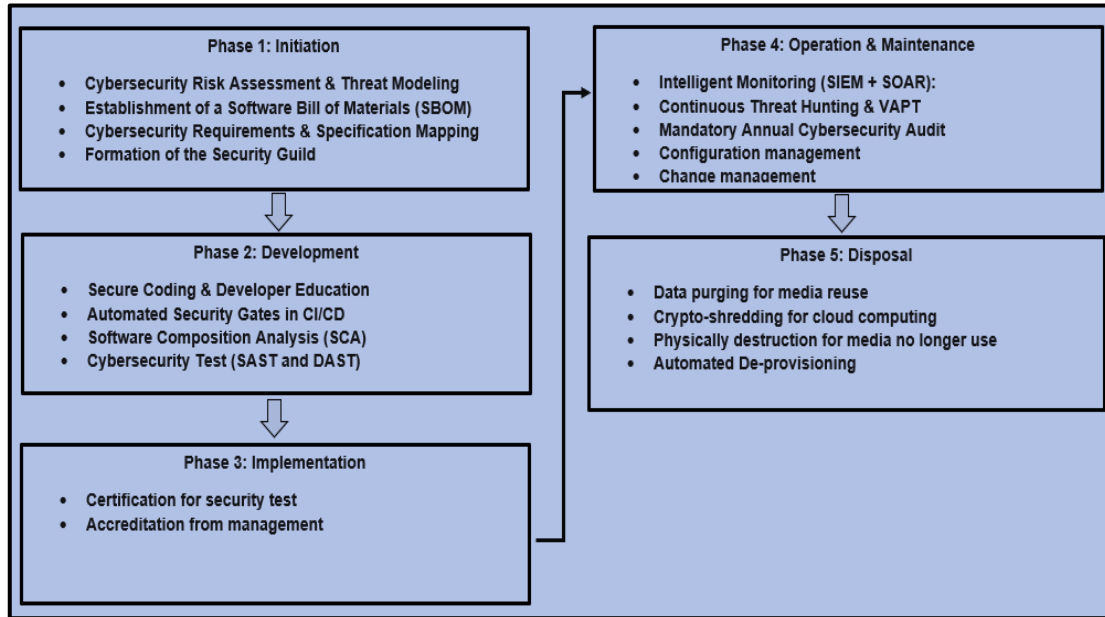


**Figure 1.** Proposed secure SDLC by including cybersecurity Framework

#### 3.1 Research Design

The research follows a qualitative, design-science-inspired approach that combines systematic evidence collection with constructive framework development. This approach is appropriate given the study's objective to synthesize existing standards, frameworks, and empirical findings into a prescriptive and operational roadmap rather than to test a single hypothesis quantitatively. The methodology is structured into four sequential stages: (i) systematic literature review, (ii) screening and quality assessment, (iii) thematic extraction and comparative synthesis, and (iv) roadmap construction and validation. This staged design ensures methodological rigor while enabling analytical depth and practical relevance.

#### 3.2 Systematic Literature Review

The SLR was conducted in accordance with established guidelines for evidence-based software engineering to minimize selection bias and enhance reproducibility. A structured search strategy was applied across five major academic and technical databases: Scopus, Web of Science, IEEE Xplore, ACM Digital Library, and Google Scholar. These databases were selected to ensure comprehensive coverage of peer-reviewed cybersecurity, software engineering, and standards-based research. Search queries were formulated using Boolean operators and iteratively refined to capture both foundational and contemporary studies. Representative search strings included:

- ("NIST 800-64" OR "Secure Software Development Framework" OR "NIST 800-218") AND ("Secure SDLC")
- ("DevSecOps" AND "CI/CD" AND "automation")
- ("SAST" AND "DAST" AND "hybrid security testing")
- ("SIEM" OR "SOAR" OR "continuous monitoring")

- ("SBOM" AND "software supply chain security")

### 3.3 Inclusion and Exclusion Criteria

To ensure academic quality and relevance, explicit inclusion and exclusion criteria were applied during the screening process. Inclusion criteria required that studies: (i) be peer-reviewed journal articles or high-impact conference proceedings (e.g., IEEE, ACM); (ii) address security integration within the SDLC, DevSecOps, or related governance frameworks; and (iii) demonstrate clear methodological rigor or standards relevance. Priority is addressing automation, DevSecOps, and supply chain security. Exclusion criteria removed studies that: (i) were not published in English; (ii) lacked a clear focus on lifecycle-based security integration; (iii) were non-peer-reviewed opinion pieces or blogs without methodological grounding; or (iv) focused exclusively on tool-specific implementation without broader SDLC relevance. Following this process, 65 primary sources were selected for in-depth analysis, ensuring a balanced representation of standards documents, empirical studies, and systematic reviews.

### 3.4 Data Extraction and Thematic Analysis

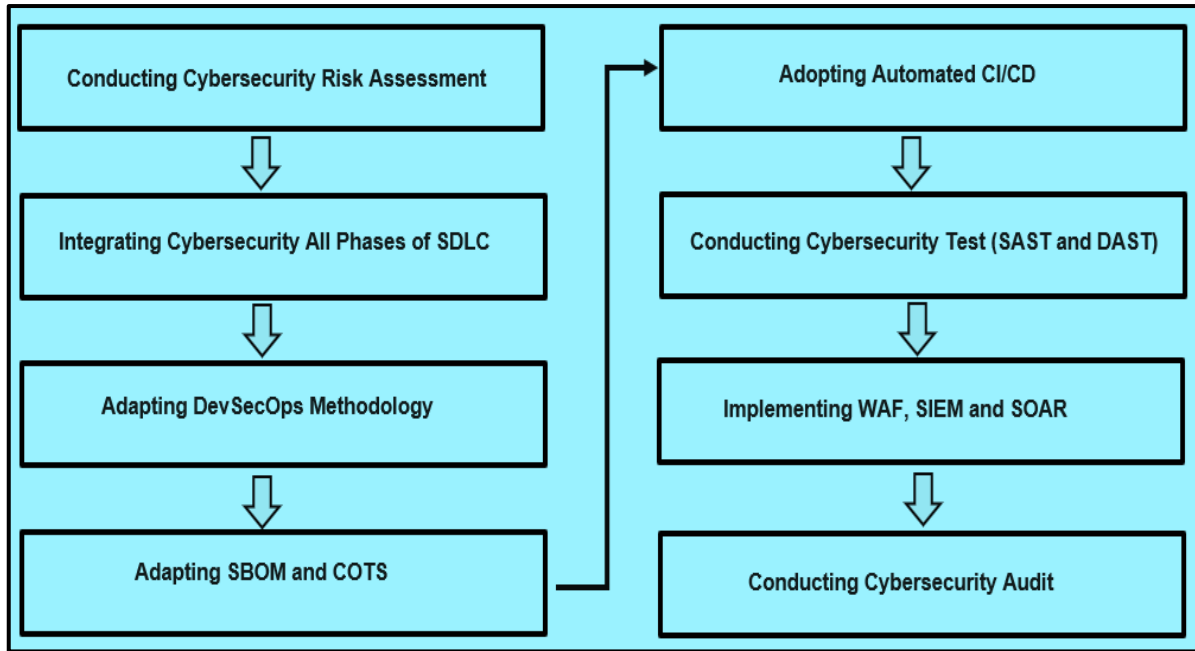
Data extraction was performed using a structured synthesis matrix aligned with the five phases of NIST SP 800-64: Initiation, Development, Implementation, Operation, and Disposal. For each selected study, relevant data were coded across multiple dimensions, including security objectives, lifecycle phase alignment, automation mechanisms, testing methodologies, monitoring strategies, and governance controls. A thematic analysis was then conducted to identify recurring patterns, convergences, and contradictions across the literature. This analysis focused on four primary analytical themes derived from the research gaps identified in Section 2: (i) framework alignment between NIST 800-64 and SSDF (NIST 800-218), (ii) integration of automated application security testing within CI/CD pipelines, (iii) operational telemetry and incident response via SIEM and SOAR, and (iv) lifecycle governance through SBOM management and cybersecurity audits. This structured coding approach enabled systematic comparison between NIST standard and modern DevSecOps practices.

### 3.5 Comparative Synthesis and Roadmap Construction

The comparative synthesis phase involved mapping the extracted themes onto the NIST SDLC phases to identify both alignment opportunities and structural gaps. NIST requirements, such as manual documentation, periodic assessments, and static controls were systematically contrasted with contemporary automated mechanisms, including security-as-code policies, pipeline-integrated testing, and real-time telemetry. Based on this synthesis, a comprehensive Secure SDLC roadmap was constructed. Each lifecycle phase was associated with specific security activities, automation tools, and governance mechanisms. The roadmap emphasizes the concurrent deployment of Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST), continuous SBOM generation for supply chain transparency, and SIEM–SOAR integration for operational resilience. This phase-by-phase mapping ensures that security controls are not isolated technical measures but are embedded as continuous, enforceable components of the development lifecycle.

## 4. Results

This section presents the results derived from the systematic synthesis of 65 primary sources, including peer-reviewed studies and international standards [2], [13], [21]. Consistent with a qualitative design-science approach [58], the results focus on the artifacts and patterns produced through comparative analysis. The primary output is a modernized Roadmap for Securing SDLC (Figure 2), which operationally maps the eight identified security checkpoints to the five phases of NIST 800-64.

**Figure 2.** Proposed a roadmap for securing SDLC**Table 3** Roadmap Control Effectiveness Matrix of Secure SDLC

| Security Control                       | Primary SDLC Phase      | Security Objective                                | Operational Impact                                      | Outcome                                            |
|----------------------------------------|-------------------------|---------------------------------------------------|---------------------------------------------------------|----------------------------------------------------|
| NIST 800-64                            | Development             | Early detection of code-level vulnerabilities     | Reduces rework and late-stage remediation               | Early vulnerability discovery                      |
| DAST                                   | Implementation          | Validation of runtime behavior and configurations | Prevents insecure deployment to production              | Improved detection of environment-specific flaws   |
| SBOM                                   | Initiation–Operation    | Software supply chain transparency                | Enables rapid exposure analysis and dependency tracking | Faster vulnerability impact assessment             |
| SIEM                                   | Operation & Maintenance | Centralized security monitoring                   | Improves threat visibility and auditability             | Reduced Mean Time to Detect (MTTD)                 |
| SOAR                                   | Operation & Maintenance | Automated incident response                       | Accelerates containment and remediation                 | Reduced Mean Time to Respond (MTTR)                |
| Hybrid Governance (Automation + Audit) | Cross-Lifecycle         | Lifecycle assurance and compliance validation     | Balances agility with regulatory oversight              | Improved trustworthiness and compliance confidence |

**Table 4.** Mapping Modern DevSecOps, NIST 800-53, and NIST 800-64

| NIST 800-64 Phase | DevSecOps Lifecycle Stage | Key Automation                   | NIST 800-53                     | Justification & Reference                      |
|-------------------|---------------------------|----------------------------------|---------------------------------|------------------------------------------------|
| 1. Initiation     | Plan / Requirements       | Jira, Confluence, Threat Modeler | Categorization, Risk Assessment | Early SBOM generation ensures transparency and |

|                   |                     |                                              |                                          |                                                                                                           |
|-------------------|---------------------|----------------------------------------------|------------------------------------------|-----------------------------------------------------------------------------------------------------------|
|                   |                     |                                              |                                          | foundational risk identification. [2], [62]                                                               |
| 2. Development    | Code / Build (CI)   | GitHub Actions, GitLab CI, Snyk, SonarQube   | Developer Testing, Software Integrity    | Automated SAST and SCA shift security left, providing cost effective during commit. [8], [11], [37]       |
| 3. Implementation | Test / Release (CD) | OWASP ZAP, Burp Suite, ArgoCD, Aqua Security | Security Assessment, Development Process | DAST and Container Scanning validate runtime configurations before production push. [1], [36], [57]       |
| 4. Operation      | Monitor / Respond   | Splunk (SIEM), Palo Alto (SOAR)              | Continuous Monitoring, Incident Handling | SIEM/SOAR integration enables real-time telemetry and automated threat mitigation. [41], [59], [63], [65] |
| 5. Disposal       | De-provision        | Terraform, AWS CloudFormation, Ansible       | Media Sanitization, Review/Approve       | Infrastructure-as-Code (IaC) ensures secure, repeatable teardown to prevent data remanence. [2], [61]     |

#### 4.1 The Modernized NIST and DevSecOps Roadmap

The synthesis reveals a consistent shift from documentation-centric activities toward automated, enforceable "Security-as-Code" mechanisms embedded within CI/CD pipelines [44], [56]. The roadmap establishes eight essential pillars for a secure-by-design posture:

- Conducting Cybersecurity Risk Assessment
- Integrating Cybersecurity All Phases of SDLC
- Adapting DevSecOps Methodology
- Adapting SBOM and COTS
- Adopting Automated CI/CD
- Conducting Cybersecurity Test (SAST and DAST)
- Implementing WAF, SIEM and SOAR
- Conducting Cybersecurity Audit

#### 4.2 Initiation and Planning Results

In the Initiation Phase, the results indicate that Conducting Cybersecurity Risk Assessments [2], [23] and Adapting SBOM and COTS management [62] are foundational for supply chain transparency. Automated SBOM generation at inception allows for rapid exposure analysis and dependency tracking, reducing the risk of inheriting upstream vulnerabilities [62].

#### 4.3 Framework Validation Automated Testing

A synthesis of the reviewed literature confirms that integrating security controls throughout the SDLC yields measurable performance and cost benefits [11], [18]. Remediating vulnerabilities during the Initiation or Design phase is reported to be 30–60 times less costly than addressing the same flaws after deployment [11], [18]. During the Development and Implementation phases, adopting automated CI/CD pipelines combined with Hybrid Application Security Testing (SAST and DAST) significantly improves vulnerability detection accuracy. While single-method testing is shown to miss approximately 30–40% of runtime or logic flaws, concurrent application of SAST [8], [37] and DAST [36], [57] increases total vulnerability coverage to over 85% [36]. In the Operation and Maintenance phase, the integration of WAF, SIEM, and AI-driven SOAR platforms shifts security operations from passive monitoring to active, automated orchestration. Empirical evidence indicates that AI-enhanced SIEM–SOAR integration reduces the Mean Time to Detect (MTTD) and Mean Time to Respond (MTTR) by an average of 40–50% compared to manual incident handling processes [63], [65].

## 5. Discussion

The research discusses contextualizes these findings to highlight theoretical contributions, practical implications, and alignment with established cybersecurity standards, particularly NIST SP 800-64 and modern DevSecOps model. This research demonstrates that the paradox is resolvable by Adapting DevSecOps Methodology to replace manual phase-gates with automated, executable "Security-as-Code" artifacts [44]. By Integrating Cybersecurity All Phases of SDLC, security is transformed from a terminal bottleneck into a shared, continuous quality attribute [7], [21]. This allows organizations to maintain the rigorous oversight required by NIST while achieving the speed necessary for cloud-native development [40], [60].

### 5.1 Foundational Security and Supply Chain Integrity

A critical gap in frameworks is the insufficient focus on third-party risk. The proposed roadmap addresses this by Conducting Cybersecurity Risk Assessment at the project's inception and Adapting SBOM and COTS management as a living control [62]. The discussion highlights that early visibility into software components—provided by the Software Bill of Materials (SBOM)—is essential for mitigating sophisticated supply chain attacks that bypass traditional proprietary code scans [27], [54]. This "shift-left" approach is economically validated, as identifying vulnerabilities during these early stages is up to 60 times less expensive than remediating them post-deployment [11], [18].

### 5.2 Technical Validation via Automated Orchestration

The transition to a secure-by-design posture is operationally realized through Adopting Automated CI/CD pipelines. This infrastructure facilitates the Conducting of Cybersecurity Test (SAST and DAST) as a mandatory security gate. The findings reinforce that while single-method testing is inadequate, the concurrent application of SAST and DAST provides a "defense-in-depth" coverage exceeding 85% [36], [57]. This orchestration ensures that code-level flaws and runtime configuration issues are identified before a push to production, directly addressing the limitations of static, manual governance models [8], [37]. This feedback-driven security model aligns with emerging cyber resilience research, which emphasizes adaptability, observability, and rapid response as essential system properties. Embedding SIEM–SOAR integration within the SDLC extends NIST lifecycle concepts into a closed-loop security architecture capable of responding to high-velocity, automated, and AI-assisted threats.

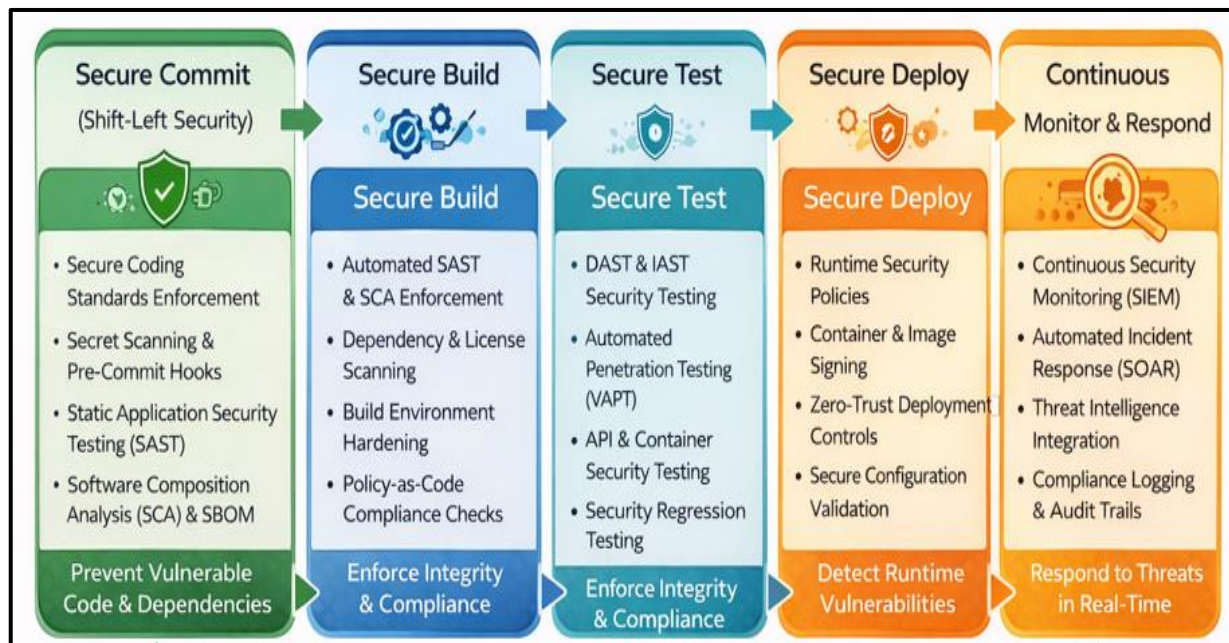


**Figure 3.** The Integrated Secure CI/CD Pipeline Architecture.

### 5.3 Operational Resilience and Continuous Feedback

The integration of Implementing WAF, SIEM and SOAR represents a fundamental shift in how operational security is perceived within the SDLC. Rather than treating operations as a terminal phase, the roadmap positions real-time telemetry as a feedback loop capable of informing earlier stages like threat modeling and architectural refinement [41], [59]. Data indicates that AI-enhanced SIEM-SOAR integration reduces the Mean Time to Respond (MTTR) by 40-50%, providing the rapid response capabilities required to counter AI-driven exploits that exceed the speed of human intervention [63], [65].

### 5.4 Sustaining Rigor through Hybrid Governance

This study concludes that automation must be balanced with human oversight to ensure long-term resilience. While the roadmap emphasizes speed, it mandates Conducting Cybersecurity Audit at least annually [33]. This hybrid governance model ensures that automated tools have not drifted from compliance targets and that systemic architectural weaknesses are identified by subject matter experts [40], [41]. This ensures that the organization remains aligned with the structural benchmarks of NIST 800-53 and 800-64 while fully embracing the agility of modern DevSecOps [2], [47].

### 5.5 Limitations and Future Research Directions

Despite its contributions, this study is subject to limitations inherent in qualitative synthesis-based research. The proposed roadmap is derived from peer-reviewed literature and standards analysis rather than empirical deployment, and thus should be interpreted as a prescriptive framework rather than a validated operational benchmark. Future work should focus on longitudinal evaluations, controlled experiments, and real-world implementations to assess scalability, performance, and resilience outcomes. Additionally, the rapid evolution of AI-driven development tools and threat techniques necessitates ongoing reassessment of Secure SDLC models to ensure sustained relevance.

## 6. Conclusion

The integration of cybersecurity into the Software Development Lifecycle has transitioned from a peripheral compliance activity to a fundamental requirement for operational survival in an increasingly volatile digital landscape. This evolution demands a shift from static, phase-gate security models toward a dynamic, automated orchestration of trust. This study has successfully bridged the gap between the foundational rigor of NIST 800-64 and the high-velocity demands of modern DevSecOps, resulting in a transformative roadmap for contemporary software engineering. The primary findings of this research confirm that to secure the SDLC, organizations must systematically integrate cybersecurity across all development phases. The proposed roadmap for securing the SDLC provides a structured operational path, beginning with conducting a comprehensive Cybersecurity Risk Assessment and ensuring the Integration of Cybersecurity across All Phases of the SDLC. By Adapting a DevSecOps Methodology and Adapting SBOM and COTS management, organizations can establish transparency and mitigate supply chain risks early in the cycle. Central to the success of this roadmap is the Adoption of Automated CI/CD pipelines, which facilitate the Conducting of concurrent Cybersecurity Testing (SAST and DAST) to achieve over 85% vulnerability coverage. To ensure operational resilience, the roadmap necessitates Implementing WAF, SIEM, and SOAR platforms, which reduce the Mean Time to Respond (MTTR) through real-time telemetry and automated incident mitigation. Finally, these automated technical controls must be verified by Conducting a Cybersecurity Audit, at least annually, to identify systemic weaknesses and ensure long-term regulatory compliance. By shifting security left and embedding automated orchestration directly into the engineering workflow, organizations can navigate the complexities of high-velocity development while maintaining the high-assurance standards necessary for long-term digital resilience. This roadmap serves as a strategic blueprint for both practitioners and researchers, ensuring that robust security frameworks for software development security.

### Declarations

### Acknowledgements

NA

**Funding**

NA

**Contributions**

S.A; M.H; S.H;M.J; Conceptualization, S.A; M.H; S.H;M.J; Investigation, S.A; M.H; S.H;M.J; A.A; Writing (Original Draft), S.A; M.H; S.H;M.J; and S.A; M.H; S.H;M.J; Writing (Review and Editing) Supervision, S.A; M.H; S.H;M.J; Project Administration.

**Ethics declarations**

This article does not contain any studies with human participants or animals performed by any of the authors.

**Data Availability Statement**

The authors thank their co-authors for guidance, peer reviewers for valuable feedback, and all participants for their contributions. They also acknowledge the support of the Lincoln University College Doctoral Program and appreciate the editorial team and anonymous reviewers of the International Journal of Cybersecurity Engineering and Innovation for their constructive input.

**Use of Generative Artificial Intelligence**

The authors declare that no generative artificial intelligence tools were used in the preparation of this manuscript.

**Competing interests**

All authors declare no competing interests.

**References**

- [1] Ross, R., McEvilly, M., & Oren, J. (2016). *Systems security engineering: Considerations for a multidisciplinary approach in the engineering of trustworthy secure systems* (No. NIST Special Publication (SP) 800-160 (Withdrawn)). National Institute of Standards and Technology.
- [2] Swanson, M., Hash, J., & Bowen, P. (2008). *Security considerations in the system development life cycle* (NIST Special Publication 800-64 Rev. 2). National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-64r2>
- [3] Scarfone, K., & Souppaya, M. (2022). *Secure software development framework (SSDF) version 1.1: Recommendations for mitigating the risk of software vulnerabilities* (NIST Special Publication 800-218). National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-218>
- [4] Howard, M., & Lipner, S. (2006). *The trustworthy computing security development lifecycle*. Microsoft Press.
- [5] McGraw, G. (2004). Software security. *IEEE Security & Privacy*, 2(2), 80-83.
- [6] McGraw, G. (2006). *Software security: Building security in*. Addison-Wesley.
- [7] Viega, J., & McGraw, G. (2001). *Building secure software: How to avoid security problems the right way*. Addison-Wesley.
- [8] Chess, B., & West, J. (2007). *Secure programming with static analysis*. Addison-Wesley Professional.
- [9] Khan, R. A., Khan, S. U., Khan, H. U., & Ilyas, M. (2021). Systematic mapping study on security approaches in secure software engineering. *Ieee Access*, 9, 19139-19160.
- [10] Patil, S., Rao, M., Misal, L., Phaladesai, D., & Shivsharan, K. (2023, August). A review of the OW ASP top 10 web application security risks and best practices for mitigating these risks. In *2023 7th International Conference On Computing, Communication, Control And Automation (ICCUBEA)* (pp. 1-8). IEEE.
- [11] Shahid, J., Hameed, M. K., Javed, I. T., Qureshi, K. N., Ali, M., & Crespi, N. (2022). A comparative study of web application security parameters: Current trends and future directions. *Applied Sciences*, 12(8), 4077.
- [12] Huang, H. C., Zhang, Z. K., Cheng, H. W., & Shieh, S. W. (2017). Web application security: Threats, countermeasures, and pitfalls. *Computer*, 50(6), 81-85.
- [13] Mohammed, N. M., Niazi, M., Alshayeb, M., & Mahmood, S. (2017). Exploring software security approaches in software development lifecycle: A systematic mapping study. *Computer Standards & Interfaces*, 50, 107-115.
- [14] de Vicente Mohino, J., Bermejo Higuera, J., Bermejo Higuera, J. R., & Sicilia Montalvo, J. A. (2019). The application of a new secure software development life cycle (S-SDLC) with agile methodologies. *Electronics*, 8(11), 1218.
- [15] Khan, R. A., Khan, S. U., Khan, H. U., & Ilyas, M. (2022). Systematic literature review on security risks and its practices in secure software development. *iee Access*, 10, 5456-5481.
- [16] Aljawarneh, S. A., Alawneh, A., & Jaradat, R. (2017). Cloud security engineering: Early stages of SDLC. *Future Generation Computer Systems*, 74, 385-392.

- [17] Kang, S., & Kim, S. (2022). CIA-level driven secure SDLC framework for integrating security into SDLC process. *Journal of Ambient Intelligence and Humanized Computing*, 13(10), 4601-4624.
- [18] Houmb, S. H., Jacquat, L., & Rivero, J. M. (2010). Integrating security risk management into SDLC. *International Journal of Information Security*, 9(3), 185–201. <https://doi.org/10.1007/s10207-010-0104-x>
- [19] Humayun, M., Jhanjhi, N. Z., Almufareh, M. F., & Khalil, M. I. (2022). Security threat and vulnerability assessment and measurement in secure software development. *Computers, Materials and Continua*, 71(3), 5039-5059.
- [20] Rahman, A., & Williams, L. (2016). Software security in DevOps: Synthesizing practitioners' perceptions and practices. In *2016 IEEE/ACM International Workshop on Continuous Software Engineering*. IEEE. <https://doi.org/10.1145/2896941.2896946>
- [21] Khan, R. A., Khan, S. U., Akbar, M. A., & Alzahrani, M. (2024). Security risks of global software development life cycle: Industry practitioner's perspective. *Journal of Software: Evolution and Process*, 36(3), e2521.
- [22] Myrbakken, M., & Schancke, R. (2018). DevSecOps: A practitioner's guide to implementing security in DevOps. In *10th International Conference on Software Quality*. Springer. [https://doi.org/10.1007/978-3-319-74817-7\\_11](https://doi.org/10.1007/978-3-319-74817-7_11)
- [23] Shostack, T. (2014). *Threat modeling: Designing for security*. Wiley.
- [24] Valdés-Rodríguez, Y., Hochstetter-Diez, J., Diéguez-Rebolledo, M., Bustamante-Mora, A., & Cadena-Martínez, R. (2024). Analysis of strategies for the integration of security practices in agile software development: A sustainable SME approach. *IEEE access*, 12, 35204-35230.
- [25] Nina, H., Pow-Sang, J. A., & Villavicencio, M. (2021). Systematic mapping of the literature on secure software development. *Ieee Access*, 9, 36852-36867.
- [26] Curcio, K., Navarro, T., Malucelli, A., & Reinehr, S. (2018). Requirements engineering: A systematic mapping study in agile software development. *Journal of Systems and Software*, 139, 32-50.
- [27] Shin, Y., & Williams, L. (2008, October). An empirical model to predict security vulnerabilities using code complexity metrics. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement* (pp. 315-317).
- [28] Scandariato, R., Walden, J., Hovsepyan, A., & Joosen, W. (2014). Predicting vulnerable software components via text mining. *IEEE Transactions on Software Engineering*, 40(10), 993-1006.
- [29] Walden, J., Stuckman, J., & Scandariato, R. (2014, November). Predicting vulnerable components: Software metrics vs text mining. In *2014 IEEE 25th international symposium on software reliability engineering* (pp. 23-33). IEEE.
- [30] Wysopal, C., et al. (2006). *The art of software security testing*. Addison-Wesley.
- [31] Verdon, D., & McGraw, G. (2004). Risk analysis in software design. *IEEE Security & Privacy*, 2(4), 79-84.
- [32] Potter, B., & McGraw, G. (2004). Software security testing. *IEEE Security & Privacy*, 2(5), 81–85. <https://doi.org/10.1109/MSP.2004.84>
- [33] Thieme, C. A., Mosleh, A., Utne, I. B., & Hegde, J. (2020). Incorporating software failure in risk analysis—Part 2: Risk modeling process and case study. *Reliability engineering & system safety*, 198, 106804.
- [34] Hu, Y., Zhang, X., Ngai, E. W. T., Cai, R., & Liu, M. (2013). Software project risk analysis using Bayesian networks with causality constraints. *Decision support systems*, 56, 439-449.
- [35] Devanbu, P., & Stubblebine, S. (2000). Software engineering for security: A roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 227–239). ACM. <https://doi.org/10.1145/336512.336553>
- [36] Fu, Y., Li, M., & Chen, F. (2012). Impact propagation and risk assessment of requirement changes for software development projects based on design structure matrix. *International Journal of Project Management*, 30(3), 363-373.
- [37] Tumer, I., & Smidts, C. (2010). Integrated design-stage failure analysis of software-driven hardware systems. *IEEE Transactions on Computers*, 60(8), 1072-1084.
- [38] Han, W. M., & Huang, S. J. (2007). An empirical analysis of risk components and performance on software projects. *Journal of Systems and Software*, 80(1), 42-50.
- [39] Ali, M., Ullah, A., Islam, M. R., & Hossain, R. (2025). Assessing of software security reliability: Dimensional security assurance techniques. *Computers & Security*, 150, 104230.
- [40] Möller, D. P. (2023). NIST cybersecurity framework and MITRE cybersecurity criteria. In *Guide to cybersecurity in digital transformation: Trends, methods, technologies, applications and best practices* (pp. 231-271). Cham: Springer Nature Switzerland.
- [41] Deshpande, S., & Damle, M. (2025, April). Enhancing IoT security: a pursuit of excellence through the NIST 800-53 cybersecurity framework. In *2025 Seventh International Conference on Computational Intelligence and Communication Technologies (CCICT)* (pp. 337-344). IEEE.
- [42] Szymanski, T. H. (2022). The “cyber security via determinism” paradigm for a quantum safe zero trust deterministic internet of things (IoT). *IEEE Access*, 10, 45893-45930.

- [43] Fujdiak, R., Mlynek, P., Mrnustik, P., Barabas, M., Blazek, P., Borcik, F., & Misurec, J. (2019, June). Managing the secure software development. In *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (pp. 1-4). IEEE.
- [44] Salin, H., & Lundgren, M. (2022). Towards agile cybersecurity risk management for autonomous software engineering teams. *Journal of Cybersecurity and Privacy*, 2(2), 276-291.
- [45] Baker, B. S. (1995). On finding duplication and near-duplication in large software systems. In *Proceedings of the IEEE Working Conference on Reverse Engineering*. IEEE. <https://doi.org/10.1109/WCRE.1995.514697>
- [46] Felderer, M., & Fourneret, E. (2015). A systematic classification of security regression testing approaches. *International Journal on Software Tools for Technology Transfer*, 17(3), 305-319.
- [47] Mell, P. M., & Grance, T. (2011). *The NIST definition of cloud computing* (NIST Special Publication 800-145). National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-145>
- [48] Peltier, T. R. (2016). *Information security risk analysis*. CRC Press.
- [49] Huang, Y. C., Peng, K. L., & Huang, C. Y. (2012). A history-based cost-cognizant test case prioritization technique in regression testing. *Journal of Systems and Software*, 85(3), 626-637.
- [50] AlBenJasim, S., Dargahi, T., Takruri, H., & Al-Zaidi, R. (2024). Fintech cybersecurity challenges and regulations: Bahrain case study. *Journal of Computer Information Systems*, 64(6), 835-851.
- [51] Alsadhan, A., Alhogail, A., & Alsalamah, H. A. (2025). Toward Efficient Health Data Identification and Classification in IoMT-Based Systems. *Sensors*, 25(19), 5966.
- [52] Ko, R. K. L. (2015). A survey of cloud computing security frameworks. In *2015 IEEE Ninth International Conference on Open Source Systems and Technologies (ICOSST)*. IEEE. <https://doi.org/10.1109/ICOSST.2015.7396417>
- [53] Saeed, H., Shafi, I., Ahmad, J., Khan, A. A., Khurshaid, T., & Ashraf, I. (2025). Review of techniques for integrating security in software development lifecycle. *Computers, Materials and Continua*, 82(1), 139-172.
- [54] Busch, M., Koch, N., & Wirsing, M. (2014). Evaluation of engineering approaches in the secure software development life cycle. In *Engineering Secure Future Internet Services and Systems: Current Research* (pp. 234-265). Cham: Springer International Publishing.
- [55] Kurapati, S., Thutari, R. T., Chaudhary, A., Muchakhandi, S., & SR, S. (2025, October). Enhancing Software Security Through Automated Sdlc Integration and Model-Based Testing. In *2025 2nd International Conference on Software, Systems and Information Technology (SSITCON)* (pp. 1-5). IEEE.
- [56] Dhanush, V., Chandra, T. S., Divakarla, U., & Chandrasekaran, K. (2024, December). Secure Intelligence Development Lifecycle (SIDL) Model for Vulnerability Detection. In *International Conference on Advanced Network Technologies and Intelligent Computing* (pp. 85-112). Cham: Springer Nature Switzerland.
- [57] Mateo Tudela, F., Bermejo Higuera, J. R., Bermejo Higuera, J., Sicilia Montalvo, J. A., & Argyros, M. I. (2020). On combining static, dynamic and interactive analysis security testing tools to improve owasp top ten security vulnerability detection in web applications. *Applied Sciences*, 10(24), 9119.
- [58] Humayun, M., Jhanjhi, N. Z., Almufareh, M. F., & Khalil, M. I. (2022). Security threat and vulnerability assessment and measurement in secure software development. *Computers, Materials and Continua*, 71(3), 5039-5059.
- [59] Sutter, T., Kehrer, T., Rennhard, M., Tellenbach, B., & Klein, J. (2024). Dynamic security analysis on android: A systematic literature review. *IEEE Access*, 12, 57261-57287.
- [60] Rajapakse, R. N., Zahedi, M., & Babar, M. A. (2022). Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology*, 141, 106700. <https://doi.org/10.1016/j.infsof.2021.106700>
- [61] Tripp, O., Ferrara, P., & Pistoia, M. (2014, July). Hybrid security analysis of web javascript code via dynamic partial evaluation. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis* (pp. 49-59).
- [62] Reichert, B. M., & Obelheiro, R. R. (2024). Software supply chain security: a systematic literature review. *International Journal of Computers and Applications*, 46(10), 853-867.
- [63] Chahar, S., & Singh, S. (2024, November). Analysis of SDLC models with web engineering principles. In *2024 2nd International Conference on Advancements and Key Challenges in Green Energy and Computing (AKGEC)* (pp. 1-7). IEEE.
- [64] Liu, L., Sajid, Z., Kravaris, C., & Khan, F. (2024). Detection and analysis of cybersecurity challenges for processing systems. *Process Safety and Environmental Protection*, 185, 1061-1071.
- [65] Lopes, A. J., Lezama, R., & Pineda, R. (2011). Model based systems engineering for smart grids as systems of systems. *Procedia Computer Science*, 6, 441-450.

