sciendo

Land Forces Academy Review
Vol. XXX, No. 1(117), 2025

CYBERSECURITY REAL-WORLD APPLICATIONS FOR THE SOFTWARE DEVELOPMENT LIFE CYCLE

Ebone MCCOY
Capitol Technology University, Laurel, Maryland, USA
emccoy@captechu.edu
ORCID: ID # 0009-0001-8355-4919

ABSTRACT

The rise in software development has intensified reliance on complex applications across sectors. However, this growth is paralleled by increased cybersecurity threats, revealing critical vulnerabilities within the Software Development Life Cycle (SDLC). Agile and DevOps methodologies, while offering speed and adaptability, often overlook vital security concerns, thus heightening exposure to cyber risks. This study addresses a novel research gap by examining how discrepancies in security integration across SDLC methodologies can lead to significant gaps in cybersecurity posture. Using a combination of case study scenarios, literature-informed analysis, and focus groups with subject matter experts, this research presents a comprehensive applied approach to cybersecurity in software development. Notably, the study engages participants with advanced expertise and real-world experience, whose insights enhance understanding of secure SDLC integration's theoretical and practical aspects. This multi-method approach provides emerging cybersecurity professionals with actionable hands-on strategies to identify and mitigate vulnerabilities, aligning with current industry standards. The study's findings underscore the urgency of standardized security practices within the SDLC, contributing to developing industry-wide best practices prioritizing security alongside speed and flexibility. By bridging theoretical frameworks with applied research, this study offers significant advancements for cybersecurity education and industry practice, preparing future professionals to address evolving cybersecurity challenges effectively.

KEYWORDS: Software Development Life Cycle (SDLC), security integration, software vulnerabilities, software assurance, secure coding practices, cybersecurity, DevOps

1. Introduction

The rapid expansion of software development has significantly heightened reliance on complex applications across various sectors, inadvertently escalating exposure to cybersecurity threats and revealing critical vulnerabilities within the software development lifecycle (Kinyua, 2020). Despite the sophistication of contemporary development methodologies, such as Agile and DevOps, these approaches emphasize speed and adaptability, frequently at the expense of essential security protocols. This oversight has resulted in vulnerabilities that malicious entities can exploit, underscoring the need for more robust security practices within these frameworks (Barabanov, 2018).

Embedding security into the software development lifecycle (SDLC) is imperative for constructing resilient software systems. However, inconsistencies

DOI: 10.2478/raft-2025-0015

in the depth and rigor of security integration across SDLC methodologies present substantial challenges. Traditional software development approaches often lack standardized frameworks for security incorporation, leading to inconsistent practices and fragmented security measures. This gap complicates the seamless security integration within the development cycle and increases the risk of significant breaches that can compromise the entire system (Valdés-Rodríguez, 2023; Seelaboyina et al., 2022). These discrepancies contribute to delayed identification of vulnerabilities and create critical cybersecurity posture gaps, exposing organizations to increased risks and limiting their ability to mitigate them proactively. Consequently, enhancing SDLC security through standardized, embedded practices from the initial requirements phase through deployment is an essential priority for industry practitioners seeking to fortify the security of increasingly complex software ecosystems (Eian et al., 2020).

2. Problem Statement

Despite establishing the Software Development Life Cycle as a software development framework, security integration is inconsistently addressed across projects (Khan et al., 2022). Software Supply chain attacks have increased triple-digit (Gartner, 2023). According to a report by the Ponemon Institute (2024), 59% of organizations reported experiencing a data breach linked to a vulnerability in their software supply chain, with 54% occurring in the past year. According to IBM, the average total cost of a data breach reached $4.88 million in 2024. This is a 10% increase since 2023 and the largest spike since the pandemic (IBM, 2024). The specific problem is exploring the best practices for cybersecurity risk management in the software development and deployment process.

3. Method and Nature of Study

This study utilizes case study frameworks, literature-informed research, and qualitative focus groups of subject matter experts. Employing case study scenarios and literature-informed research methodologies provides a robust applied research approach. The approach bridges the gap between theoretical understanding and practical application. This study employed a qualitative focus group research approach, utilizing two 60-90 minute focus group meetings. Each session included six participants, resulting in a total sample size of twelve cybersecurity subject matter experts. The participants were carefully selected based on several key criteria to ensure the findings' relevance, depth, and practical insight. Each participant held at least one recognized cybersecurity certification and possessed at least two years of professional cybersecurity experience, underscoring their technical credibility, and understanding of cybersecurity frameworks, practices, and risks. Furthermore, all participants had over five years of direct experience in SDLC, offering rich study insights into the complex interactions between development and security. The participants' educational background further solidified their expertise and enhanced the study's reliability, as each held a graduate degree in information technology or cybersecurity. This advanced education enabled participants to contribute informed perspectives on cybersecurity practices' theoretical underpinnings and practical implications across the SDLC.

4. Literature Review

4.1. Software Assurance Requirements and Expectations

The National Institute of Standards and Technology (NIST) conceptualizes software assurance as the confidence level that a software product is devoid of vulnerabilities, whether these arise from

accidental coding oversights or are deliberately embedded for potential exploitation (NIST, 2022). Software products marred by numerous vulnerabilities pose substantial risks that should be deemed unacceptable by clients and end-users alike, as these weaknesses directly undermine the reliability and security of the software (Symth, 2017).

When security protocols are not rigorously integrated during the software development lifecycle, products are more likely to contain exploitable flaws that jeopardize the confidentiality, integrity, and availability of sensitive data. This lack of initiative-taking security consideration introduces entry points for malicious actors, compromising end-user data security and diminishing trust in the product (NIST, 2022). By incorporating stringent security measures throughout the development process, developers can significantly reduce these risks, enhancing the overall resilience and trustworthiness of the software. Ensuring software assurance through meticulous security practices is paramount to maintaining data protection and user trust. When developers prioritize security from inception to deployment, they fulfill a foundational responsibility to mitigate vulnerabilities, offering clients a product that safeguards sensitive information and exemplifies robust, responsible coding practices (Arega et al., 2024).

Vendors must meet stringent expectations in delivering software products, which frequently include performance, security, and stability benchmarks to satisfy client needs and safeguard operational integrity (Pargaonkar, 2021). For many organizations, comprehensive software assurance is critical to software purchasing decisions. This assurance underscores a commitment to quality, addressing potential vulnerabilities that could undermine an organization's information security posture (Arega et al., 2024).

Software assurance is a foundational measure to prevent the proliferation of inadequate software products that fail to meet essential security and quality standards. Software is more susceptible to hidden vulnerabilities without robust assurance protocols, leading to significant security breaches and disruptions (Arega et al., 2024). Setting higher assurance standards, therefore, empowers vendors to produce software that not only aligns with security and performance requirements but also upholds organizational confidence in the product's reliability and safety (Pargaonkar, 2021; Arega et al., 2024). Implementing comprehensive software assurance processes is not merely a security enhancement; it establishes a framework for delivering high-quality software that clients can trust (Arega et al., 2024). Such practices bridge the gap between operational expectations and the vendor's responsibility to provide secure, resilient products, fostering an industry-wide commitment to excellence and security in software development.

4.2. Key Attributes of the SDLC

The Software Development Life Cycle (SDLC) is underpinned by critical phases such as Testing, Operations, and Maintenance, each designed to ensure robust security and functionality through systematic evaluation and ongoing patching of vulnerabilities (Felderer, 2016). Within the Testing phase, the SDLC incorporates multiple layers of assessment spanning code quality, performance, and security testing to confirm the software's readiness for deployment. This stage is vital for identifying and addressing potential weaknesses that could compromise the software's integrity and security (Lemke, 2018).

Omitting or inadequately conducting the testing phase introduces significant risks to deployment, as unaddressed vulnerabilities can lead to data leaks, security breaches, and even deployment failures. Common security

issues, including design flaws and implementation errors, often arise throughout the SDLC, underscoring the necessity of early and rigorous intervention. Proactively addressing these vulnerabilities reduces the overall risk and mitigates the high costs associated with later-stage corrections (Tung et al., 2016; Korir, 2023). Thus, integrating comprehensive testing within the SDLC fortifies the software's security and safeguards its operational viability. By committing to early, proactive testing and ongoing maintenance, developers can deliver secure, reliable products that meet the evolving demands of security-conscious industries, ensuring that critical software systems remain resilient against emerging threats.

4.3. Software Development Life Cycle Assessment

The Software Development Life Cycle is a traditional process of creating high-quality software from start to finish with structured steps (Humayun et al., 2022). The common phases within the software development life cycle models include planning, design, development, testing, and maintenance (Humayun et al., 2022). The SDLC framework offers a process to understand the development problem, create a plan, code the solution, test the program, and maintain the product (Gupta et al., 2021). According to NIST (2022), very few software development models include software security in the development process. Security implementation should be added and addressed in every SDLC model and integrated throughout the process (NIST, 2022). As depicted in Figure no. 1, the SDLC phases are commonly conducted in five phases: Plan, Design, Develop, Test, and Deploy (Hossain, 2023).

See the figure below for the Software Development Life Cycle.

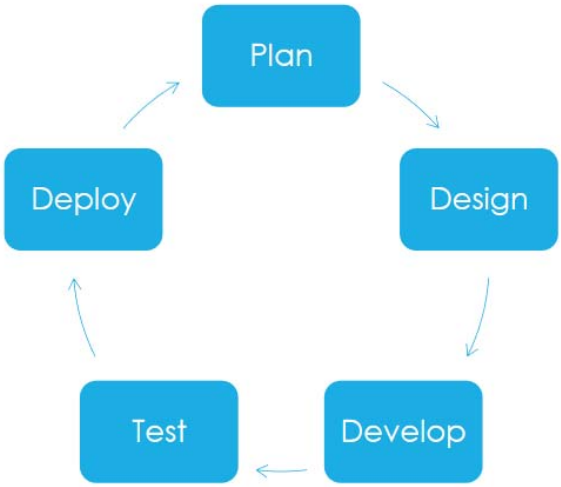

Figure no. 1: *Software Development Cycle Phases*
(Source: Gupta et al., 2021)

●**Phase 1**: The planning phase consists of defining the scope of concepts and preparing quality assurance and risk management plans. Documentation summarizing all ideas, and a complete view of the full plan should be prepared (Hossain, 2023).

●**Phase 2**: Junior design team members create a design document specification and pass it to senior members and project stakeholders for revision. The design document is assessed based on various criteria, including budget, time, user-friendliness, risk, and integration (Tiky, 2016).

●**Phase 3**: Once the design decisions are implemented, programmers should develop the software accordingly while following the coding standards defined by the organization. During the development phase, programmers must write functional specifications to record all functions provided at a technical level (Hossain, 2023).

●**Phase 4**: The software should be tested upon completion of development. The testing phase can be implemented as a sub-stage within all phases, but it should be conducted twice at the programmer, end-user, and quality assurance expert levels. This ensures successful software execution, comprehensiveness, and correctness (Hossain, 2023).

●**Phase 5**: The final decision of the Software Development phase is whether the product should be deployed to the production environment and approved by management. Along with a deployment guide, a guideline document should be provided for installation, administration, and end users (Hossain, 2023).

5. Security Principles

The following principles should be implemented during the software development life cycle:

●**Defense in Depth**: The principle of defense in depth suggests that where one control would be reasonable, more controls that approach risks in different fashions are better. When used in-depth, controls can make severe vulnerabilities extraordinarily difficult to exploit and thus unlikely to occur. With secure coding, this may take the form of tier-based validation, centralized auditing controls, and requiring users to be logged on all pages (OWASP, 2020).

●**Least Privilege**: The principle of least privilege recommends that accounts have the least privilege required to perform their business processes. This encompasses user rights and resource permissions such as CPU limits, memory, network, and file system permissions (OWASP, 2020).

●**Avoid Security by Obscurity**: Security through obscurity is a weak security control that often fails when it is the only control. The security of key systems should not rely on keeping details hidden (OWASP, 2020).

●**Keep Security Simple**: Certain software engineering fads prefer overly complex approaches to straightforward ones. Developers should avoid using double negatives and complex architectures when a simpler approach would be faster and simpler (OWASP, 2020).

●**Fix Security Issues Correctly**: Once a security issue has been identified, it is important to develop a test for it and understand the root cause. When design patterns are used, the security issue is widespread across all code bases. Developing the right fix without introducing regressions is essential (OWASP, 2020).

6. Common Vulnerabilities in Software Development

Existing software vulnerabilities present critical challenges within cybersecurity, as they provide potential entry points for malicious exploitation that can significantly compromise the integrity of information systems (Seelaboyina et al., 2022). Research highlights that many of these vulnerabilities emerge during the design and coding phases, often due to insufficient prioritization of security protocols. Among the most frequently identified vulnerabilities are insecure coding practices, weak authentication mechanisms, and insufficient data validation, each exposing systems to heightened security risks (Seelaboyina et al., 2022).

These vulnerabilities underscore the importance of integrating comprehensive security measures throughout the software development lifecycle (Seelaboyina et al., 2022). Insecure coding practices can lead to software flaws that attackers readily exploit, while weak authentication mechanisms fail to restrict unauthorized access, leaving systems vulnerable adequately. Similarly,

inadequate data validation can result in data integrity issues, facilitating attacks like SQL injection and buffer overflow. These weaknesses amplify the risk of serious data breaches and system compromises, particularly if not addressed early in development (Seelaboyina et al., 2022). To mitigate these risks, developers must embed rigorous security practices within the foundational software design and coding stages. By addressing common vulnerabilities through secure coding standards, robust authentication protocols, and thorough data validation, organizations can significantly reduce the likelihood of exploitation and enhance the overall security posture of their information systems (Seelaboyina et al., 2022).

7. Recommendations from Literature

7.1. Agile approach/Immutable software

In the realm of software development, a variety of methodologies serve to guide and optimize the life cycle process. Agile Software Development, for example, offers a framework designed to enable adaptability and outcome-oriented results, allowing software programs to respond fluidly to evolving conditions and user demands (Al-Saqqa et al., 2020). This approach's iterative nature accommodates changes and ensures continuous alignment with customer requirements, fostering a responsive development environment where collaboration among team members is integral to achieving high-quality outcomes (Al-Saqqa et al., 2020). Through ongoing adjustments, Agile significantly mitigates risks associated with static development processes and supports sustained improvement throughout the software's life cycle.

7.2. Secure Software Development

Very few software development life cycle (SDLC) models provide comprehensive guidance on security, necessitating the deliberate integration of secure development practices across all stages of any SDLC model. Embedding security measures throughout the development process serves several essential purposes: it reduces the number of vulnerabilities in the final product, lessens the potential impact of exploiting any residual vulnerabilities, and addresses the underlying causes of security flaws to prevent their recurrence (NIST, 2022). A consistent, standardized approach to integrating security across various SDLC methodologies is critical, offering a unified framework to mitigate vulnerabilities and reduce cyber-attack incidence (Valdés-Rodríguez et al., 2023). Implementing security measures earlier in the development cycle minimizes vulnerabilities, strengthening the software's resilience against evolving security threats.

7.3. Supply Chain Assessment

The software supply chain has emerged as a critical vulnerability in cybersecurity, increasingly targeted by attackers seeking to exploit weak links. Effective supply chain security begins with a rigorous evaluation of the security policies designed to protect the product and its end users (Boyens et al., 2021; CISA, 2021). Particular attention is warranted when vendors outsource services, such as distribution or data centers, as these arrangements introduce additional risk layers that necessitate careful scrutiny. Assessing whether the vendor provides ongoing maintenance or security patches is essential for understanding the level of software assurance embedded within the product. As emphasized by the Cybersecurity Infrastructure and Security Agency (CISA, 2021), initiative-taking measures to avoid acquiring vulnerable software include implementing security requirements for all suppliers, integrating security protocols within the software development life cycle, identifying, and addressing known vulnerabilities in source code, and restricting procurement to pre-

approved suppliers. These practices establish a robust framework to protect the software supply chain from compromise and ensure a higher security standard across the product's life cycle.

8. Recommendations from the Focus Group Data

Data was collected from two separate focus groups of six participants in each group. There were twelve total participants. The focus group participants were selected for their expertise, each holding a cybersecurity certification, with over two years of cybersecurity experience and more than five years in the software development life cycle (SDLC). Additionally, all participants held a graduate degree in information technology or cybersecurity, ensuring depth and relevance in their insights. Conducting two separate focus groups of six participants each, rather than one group of twelve, enhanced both the validity and triangulation of data in several ways:

➢ **Enhanced Validity through Group Dynamics and Richer Data**

●Smaller focus groups allowed for a more intimate setting, encouraging participants to share deeper insights, offer detailed explanations, and engage in more reflective discussions. With twelve participants in one group, individuals may feel less inclined to participate actively, potentially leading to a dominance of a few voices and a reduction in data richness.

●Smaller groups reduced social pressures and the "groupthink" phenomenon, where participants might align their responses with perceived group consensus, which can dilute the authenticity of responses. This resulted in more independent contributions, enhancing the internal validity of the data collected.

➢ **Data Triangulation through Cross-Group Comparison**

●Conducting two separate focus groups enabled methodological triangulation by comparing the responses from each group. This cross-group comparison provided an additional layer of insight, helping to verify whether themes and patterns are consistent across groups. Similar insights emerged independently across both groups, supporting the convergent validity of the findings.

The following data collection questions guided the focus group conversations:

"What specific strategies or practices have you found effective in integrating security controls within each phase of the SDLC, and how can these practices be adapted or scaled across teams to address emerging cybersecurity threats?"

This question encourages participants to reflect on their direct experiences and share effective security integration practices that can be generalized or tailored for broader use. It aimed to uncover real-world solutions adaptable across varied project teams and organizational contexts, fostering a practical approach to SDLC security.

Each group brainstormed to create master lists of recommendations. After the master lists were created, each group voted and reached a consensus on the top recommendations they thought were the most critical. The final lists were compared between the two groups. The results were developed based on the common highest-voted recommendations between the two groups.

The following recommendations emerged from the focus groups.

➢ **Adopt a Security-First Mindset in Agile and DevOps Environments**

●**Integrate DevSecOps**: Embedding security into DevOps (DevSecOps) ensures security is a continuous process rather than an afterthought. This integration involves automating security checks, using vulnerability scanning tools, and deploying secure coding practices to address risks promptly without disrupting the development cycle.

To elaborate on the recommendation - Participant 1 in focus group 2 said, "*I think it is critical to make security part of DevOps from the start. The approach of integrating security into DevOps, or what we call DevSecOps, keeps security running alongside every other process. This approach means that security checks and scans happen continuously, so vulnerabilities are caught before they cause real issues. It's a way of assessing and managing risk as we go instead of facing costly fixes later*."

●**Security Training for Development Teams**: Teams must be educated on secure coding practices and common vulnerabilities (e.g., OWASP Top 10) to reduce the risk of introducing vulnerabilities. Regular security training helps developers understand the critical role security plays in Agile and DevOps environments.

To elaborate on the recommendation - Participant 6 in focus group 1 said, *"It is important that developers have training and expertise in secure coding approaches. This training and expertise are especially in agile environments because it reduces the chances of introducing new vulnerabilities."*

To elaborate on the recommendation - Participant 5 in focus group 1 said, "*When developers understand secure coding practices and how to spot common vulnerabilities, they can avoid errors that compromise security. It is critical that all developers are trained to understand cybersecurity risk management that applies to the software development process*."

➢ **Standardize Security Integration Across SDLC Phases**

●**Establish Security Standards and Guidelines**: Standardized security frameworks, such as NIST's Secure Software Development Framework (SSDF), help maintain consistency in security practices across different SDLC methodologies. This approach reduces gaps in security posture and ensures a collective understanding of security requirements.

To elaborate on the recommendation - Participant 2 in group 2 said, "*Organizations need to set clear security standards and stick to them. This process requires training all developers to understand and follow standardized security framework (like NIST's SSDF) to ensure that security practices are consistent, regardless of the methodology. This consistency helps reduce risk gaps, making risk assessment easier because everyone's on the same page regarding security requirements*."

●**Security Requirements Engineering**: During the requirements phase, include security as a core requirement, such as ensuring compliance with standards like ISO/IEC 27001 for information security management. Collaborate with stakeholders to identify and document security requirements early, reducing the likelihood of future vulnerabilities.

To elaborate on the recommendation - Participant 4 in group 1 said "*It is critical for organizations to have polices and processes that take security part of the requirements phase. Security needs to be in all engineering plans and processes from day one. During requirements gathering, make security a core requirement, like meeting ISO standards for information security. Early inclusion helps avoid later vulnerabilities, meaning less risk to manage down the road*."

➢ **Enhance Software Assurance for Quality and Security**

●**Secure Code Review and Static Analysis**: Automated static analysis tools like SonarQube and Checkmarx can detect code vulnerabilities early in the development process. By performing code reviews and static analysis, organizations can reduce the risks associated with poor coding practices, one of the primary sources of security flaws.

To elaborate on the recommendation - Participant 5 from group 2 said, "*At my current organization we use static analysis and code reviews. Organizations can have*

processes and policies where they use automated static analysis tools, like SonarQube or Checkmarx, catch vulnerabilities in code early. These tools are critical because early detection is essential for managing risks related to coding errors, which can be one of the biggest security issues."

●**Dynamic and Penetration Testing**: Testing software in runtime environments through dynamic application security testing (DAST) and penetration testing (using tools like Burp Suite or OWASP ZAP) helps identify vulnerabilities that only emerge when the software is operational. These tests are especially crucial in identifying logic flaws and exploiting risks.

To elaborate on the recommendation - Participant 6 in group 1 said, "*I lead my team in running tests in real-world conditions. We explore dynamic testing and penetration tests because it allows us to see how software behaves in real-world environments. These activities offer us a new layer of defense and quality control because it allows us to catch vulnerabilities in testing that might only show up when the software's running live, making it crucial for a thorough risk assessment*."

●**Software Supply Chain Security**: Implementing a software bill of materials (SBOM) for tracking third-party components is essential to identify and manage vulnerabilities within software dependencies. This practice helps prevent software supply chain attacks, which have seen significant increases.

To elaborate on the recommendation - Participant 3 in group 2 said, "*A make sure that we watch vendors and third-party dependencies closely. We constantly evaluate and review SBOM (software bill of materials) lets us track third-party software components and manage vulnerabilities they might carry. With supply chain attacks on the rise, it's about assessing and mitigating risk from any external code in our system*."

➢ **Automate Security Checks in Testing and Deployment**

●**Continuous Integration and Continuous Deployment (CI/CD) Security**: CI/CD pipelines should include automated security checks, such as scanning for vulnerabilities, assessing dependencies, and verifying the security configurations of code deployments. Tools like Jenkins with security plugins can enforce security at each stage, making secure deployments more efficient and reliable.

To elaborate on the recommendation - Participant 4 in group 1 said "*An approach that we use is to automate security in CI/CD pipelines. What we have found is that by adding security checks into CI/CD pipelines we catch vulnerabilities as code moves through development stages. It's an efficient way to assess and manage risk continuously, making sure secure code goes live with fewer delays*."

●**Patch Management and Vulnerability Remediation**: Integrate patch management into the operations and maintenance phases. With vulnerability scanning and monitoring in place, teams can quickly identify and remediate vulnerabilities post-deployment, maintaining the security posture over time.

To elaborate on the recommendation - Participant 2 in group 2 said, "*We train developers in our organization to be proactive instead of reactive. As a result, we don't wait to patch vulnerabilities. We have built patch management and vulnerability remediation into our operations processes, so any new vulnerability is caught and fixed right away. We have found that this practice keeps the overall risk profile low, especially post-deployment*."

➢ **Employ Security Metrics and Continuous Monitoring**

●**Security Metrics**: Tracking key metrics, such as the number of vulnerabilities discovered per release or time to patch, provides insights into the

effectiveness of security practices. These metrics should be reviewed regularly to identify areas for improvement.

To elaborate on the recommendation - Participant 6 in group 1 said, "*You've got to track metrics to understand risk exposure. It is critical to include that in your processes and activities. This is important because security metrics, like the number of vulnerabilities per release or average time to patch, show how well security measures work. Reviewing metrics regularly helps identify any weak spots in security practices that could increase risk.*"

●**Continuous Monitoring with SIEM and Threat Intelligence**: Security Information and Event Management (SIEM) systems can monitor application behavior and detect unusual activity, which may indicate a potential security breach. Integrating threat intelligence into monitoring efforts helps organizations proactively identify and mitigate evolving threats.

To elaborate on the recommendation - Participant 2 in group 1 said, "*We use artificial intelligence tools as part of a Security Information and Event Management (SIEM) system to help us track and evaluate application behavior for unusual activity that might signal a breach. By continuously monitoring and integrating threat intelligence, we can assess and respond to risks proactively.*"

➢ **Implement Secure Software Delivery Assurance Programs**

●**Develop a Software Assurance Program**: Formalize a software assurance program aligned with NIST's definition to ensure a high degree of confidence that software products are free from critical vulnerabilities. This program should include periodic audits, certifications, and adherence to security benchmarks.

To elaborate on the recommendation - Participant 5 in group 1 said, "*In the last 2 years we formalized a software assurance program aligned with NIST standards which ensures that software meets high-security benchmarks, lowering the risk of critical vulnerabilities. This year we started a quarterly process of regular audits keep the security profile strong and build confidence in the software's security.*"

●**Vendor and Third-Party Risk Management**: If external vendors are involved, establish security expectations, and conduct regular security assessments of their products. Many breaches occur through third-party software vulnerabilities; managing these risks requires close vendor oversight and frequent evaluations of their security practices.

To elaborate on the recommendation - Participant 2 in group 2 said, "*Breaches through third-party software are common. As a result, it's a best practice to set clear security expectations with vendors and assessing them regularly is the key. It's a straightforward way to manage risks from outside the organization.*"

➢ **Apply Threat Modeling and Early Security Reviews**

●**Threat Modeling in Design Phase**: Implement threat modeling methodologies to identify and mitigate security risks in the design phase. Regular threat modeling helps address potential risks early, reducing the cost and complexity of security issues that could emerge later in the SDLC.

To elaborate on the recommendation - Participant 3 in group 2 said, "*I was trained to employ threat modeling during the design planning and implementation phases. I trained in the use of methods like STRIDE or PASTA because they are effective in identifying risks before coding even starts. This proactive approach has helped my organization reduce the cost of security risk management concerning flaws in need of fixes.*"

●**Security Reviews and Code Audits**: Conduct security design reviews and code audits at critical checkpoints. These reviews enable a holistic assessment of the software's security architecture and identify design flaws before moving forward with implementation.

To elaborate on the recommendation - Participant 6 in group 2 said, "*At my organization we conduct and document regular and consistent security reviews at key stages of the development process. Basically, we engage in comprehensive processes that check the security architecture and code at critical points. We do this because it has allowed us to uncover design flaws early when the software's security risk management process is more cost effective*."

- **Enhance Incident Response and Recovery Plans**

●**Define Incident Response Protocols**: Ensure that incident response plans include protocols for handling software vulnerabilities, such as procedures for vulnerability disclosure and patch deployment. Incident response protocols should be well-documented practice improves response times during a breach.

To elaborate on the recommendation - Participant 2 in group 1 said, "*Every year, as an organizational and professional development activity, we gather team together in a brief retreat to define incident response protocols and practice them. These meetings function as training activities that have helped us ensure that everyone on the team is on the same page with knowledge and expertise about how to oversee any vulnerability if one arises. This process of collaborative learning and team engagement has helped us realize shorter response times and better risk management during incidents. We have been able to experience reduce faster containment and recovery times*."

●**Post-incident Reviews and Knowledge Sharing**: After a security incident, conduct post-incident reviews to analyze causes, apply lessons learned, and document any improvements. Sharing these findings across development and security teams strengthens organizational resilience and improves future response strategies.

To elaborate on the recommendation - Participant 6 in group 1 said, "*Our organizational leadership employs a culture of Just Cause principles within post-incident reviews fosters a culture rooted in transparency, accountability, and fairness. Just Cause principles emphasize that employees should be treated with respect and that corrective measures following incidents should be constructive, focusing on education, lessons learned, and improvement rather than shame, blame, and severe punishment. When organizations apply these principles in the aftermath of a cybersecurity or software security incident, they prioritize understanding the root causes and systemic factors rather than attributing fault to individuals. This approach encourages employees to engage openly in post-incident reviews, knowing that the focus remains on learning and improvement. By aligning with just cause principles, organizations enhance trust and foster an environment where employees feel safe to report vulnerabilities and suggest improvements, which collectively strengthens the organization's security posture*."

To elaborate on the recommendation - Participant 5 in group 1 said, "*In our organizations we use lessons learned workshops that allow us to bring together team members to discuss what happened during the incident and what could be done differently next time. This open conversation allows everyone to learn from the event, understand what worked well, and see where improvements are needed. This shared knowledge helps prevent similar incidents in the future by keeping the whole team informed and ready*."

To elaborate on the recommendation - Participant 1 in group 1 said, "*My organization has adopted a process of Post-Incident Review and Root Cause Analysis. During this process, our teams dig into the details of the incident to find out what caused the problem in the first place. By identifying the root causes, teams can*

address the specific issues that led to the incident and make adjustments to stop it from happening again."

To elaborate on the recommendation - Participant 3 in group one said, "*After incident, we have process and after-action briefings and reports. We use these activities to help us update our Incident Response Playbook. This playbook provides guidance that ensures that any new lessons learned are built into future response plans. This playbook provides clear steps for handling incidents, so updating it with recent insights keeps the team prepared and able to respond in a more agile and adaptable way. Together, these processes help strengthen cybersecurity across the software development life cycle by creating a culture of continuous learning and improvement*."

9. Conclusion

In summary, as technological advancements accelerate, the sophistication of vulnerabilities, threats, and attack strategies by adversaries will continue to evolve. The link between secure software development practices and data integrity is indisputable, underscoring the importance of integrating security measures within the software development life cycle (Eian et al., 2020). Security remains a pivotal discussion within the developer and software communities due to its inherent challenges and fundamental role in system stability. Integrating these recommendations across the SDLC provides a security foundation that continuously evolves with the software, reducing vulnerabilities and promoting a proactive approach to cybersecurity in software development.

REFERENCES

Al-Saqqa, S., Sawalha, S., & Abdel-Nabi, H. (2020). Agile software development: Methodologies and Trends. *International Journal of Interactive Mobile Technologies, 14 (11),* 246-70. Available at: https://doi.org/10.3991/ijim.v14i11.13269.

Barabanov, A.V., Markov, A.S., & Grishin, M.I. (2018). Current taxonomy of information security threats in software development life cycle. *2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT)*. DOI: 10.1109/ICAICT.2018.8747065. Available at: https://ieeexplore.ieee.org/abstract/document/8747065/.

Arega, K.L., Beyene, A.M., & Yitagesu, S. (2024). Security Assurance in the Software Development Process: A Systematic Literature Review. In: Rajagopal, S., Popat, K., Meva, D., Bajeja, S. (eds) Advancements in Smart Computing and Information Security. *ASCIS 2023. Communications in Computer and Information Science, Vol. 2040, 16-30*. Springer, Cham. Available at: https://doi.org/10.1007/978-3-031-59107-5_2.

Boyens, J., Paulsen, C., Moorthy, R.S., & Bartol, N. (2022). *NIST Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations*. National Institute of Standards and Technology. Available at: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-161r1.pdf.

Cybersecurity and Infrastructure Security Agency (CISA). (2021, April). *Defending against software supply chain attacks.* U.S. Department of Homeland Security, Available at: https://www.cisa.gov/publication/supply-chain-risks-information-and-communication-technology.

Eian, I.C., Yong, L.K., Li, M.Y.X., & Hasmaddi, N.A.B.N. (2020). Integration of security modules in software development lifecycle phases. *arXiv Preprint arXiv:2012.05540,*

Computer Science, Software Engineering. Available at: https://doi.org/10.48550/arXiv.2012.05540.

Felderer, M., Büchler, M., Johns, M., & Brucker, A. (2016). Chapter one – Security testing: A survey. *Advances in Computers, 101*, 1-51. Available at: https://www.sciencedirect.com/science/article/pii/S0065245815000649.

Gartner (2023). *Mitigating Enterprise Software Supply Chain Security Risks*. Gartner Research. Available at: https://www.gartner.com/en/documents/4893131.

Gupta, A., Rawal, A., & Barge, Y. (2021). Comparative Study of Different SDLC Models. *International Journal for Research in Applied Science & Engineering Technology, 9 (1)*. Available at: https://doi.org/10.22214/ijraset.2021.38736.

Hossain, M.I. (2023). Software development life cycle (SDLC) methodologies for information systems project management. *International Journal for Multidisciplinary Research (IJFMR), 5 (5)*. Available at: https://www.researchgate.net/publication/373800862_

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 & Continua*, *71* (3), 5039-5058. Available at: https://doi.org/10.32604/cmc.2022.019289.

IBM. (2024). *Cost of a data breach report 2024*. IBM Security. Available at: https://www.ibm.com/downloads/cas/1KZ3XE9D.

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. *Ieee Access, 10*. Available at: https://ieeexplore.ieee.org/abstract/document/9669954/.

Kinyua, J. (2020). *Cybersecurity in the software development life cycle. In book: Cybersecurity for Information Professions*. DOI:10.1201/9781003042235-12.

Korir, F.C. (2023). Software security models and frameworks: An overview and current trends. *World Journal of Advanced Engineering Technology and Sciences, 8 (2)*, 86-109. Available at: https://doi.org/10.30574/wjaets.2023.8.2.0078.

Lemke, G. (2018). *The software development life cycle and its application*. Senior Honors Theses and Projects. Available at: https://commons.emich.edu/honors/589/.

National Institute of Standards and Technology (NIST). (2022). *Secure software development framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities*. Available at: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf.

OWASP. (2020). *Secure Development and Integration*. Available at: https://owasp.org/www-project-developer-guide/draft/foundations/secure_development/.

Pargaonkar, S. (2021). The Crucial Role of Inspection in Software Quality Assurance. *Journal of Science & Technology, 2 (1)*. Available at: https://thescience brigade.com/jst/article/view/42.

Ponemon Institute. (2024). *The state of software supply chain security risks*. Black Duck. Available at: https://www.blackduck.com/content/dam/black-duck/en-us/reports/state-of-software-supply-chain-security-risks-ponemon.pdf

Seelaboyina, R., Vadla, S., & Teerthala, S. (2022). *Secure Software Development Life Cycle: An Approach to Reduce the Risks of Cyber Attacks in Cyber-Physical Systems and Digital Twins*. In: Gunjan, V.K., Kumar, A., Zurada, J.M., Singh, S.N. (eds) Computational Intelligence in Machine Learning. *ICCIML 2022*. *Lecture Notes in Electrical Engineering, Vol. 1106*. Springer, Singapore. Available at: https://doi.org/10.1007/978-981-99-7954-7_15https://link.springer.com/chapter/10.1007/978-981-99-7954-7_15.

Tung, Y., Lo, S., Shih, J., & Lin, H. (2016). An integrated security testing framework for secure software development life cycle. *18th Asia-Pacific Network Operations and Management Symposium, IEEE Xplore*. Available at: https://ieeexplore.ieee.org/abstract/document/7737238/.

Valdés-Rodríguez, Y., Hochstetter-Diez, J., Díaz-Arancibia, J., & Cadena-Martínez, R. (2023). Towards the integration of security practices in agile software development: a systematic mapping review. *Applied Sciences, 13 (7)*, 4578. Available at: https://www.mdpi.com/2076-3417/13/7/4578.