International Journal for Multidisciplinary Research (IJFMR)

E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Evaluation and Assessment of Software Security Risks and Vulnerabilities Within the Realm of Secure DevOps

Babagana Ali Dapshima[1], Samaila Kasimu Ahmad[2]

[12]Department of Computer Science and Engineering, Sharda University, Greater Noida UP India

ABSTRACT

Security plays a vital role in successful software development and consistency, especially in a modern, fast-expanding world dominated by technologies. It guarantees that all privacy, integrity, and accessibility of information, coding, and services are protected. Achieving such requires a consistent consideration of security throughout all developmental stages of software. Although several methods for improving the quality of software have been created, there is still no clear method for effectively integrating security throughout each stage of the software developmental process. The inability to give adequate attention to security has been identified as the main cause of common flaws. Many forward-thinking businesses frequently employ a "patch and infiltrate" technique, taking security into consideration only after the execution of the task. Higher costs, project delays, neglect of verification and integration during the development life cycle of the project, reliance on external tools and parts, and ignorance among all factors contribute to the disregard of security during the SDLC. Nevertheless, Secure-DevOps is indispensable for the survival of companies in the Information and Communication Technology (ICT) landscape. Consequently, applying an effective approach that deals with security within every level of the SDLC is crucial. To bridge this gap, a comprehensive examination of effective techniques for developing secure software was presented, taking into account the financial and time constraints of a project. A framework for ensuring security built upon Secure-DevOps practices, which seamlessly integrates the best security measures into different stages of the SDLC, was identified. To validate this framework, a mathematical model has been employed. The study concludes that effectively incorporating Secure-DevOps practices throughout the Software developmental process, minimizes security risks and vulnerabilities.

Keywords: Evaluation, Risk. Security, Software, Secure-DevOps, Vulnerabilities.

1. Introduction

Security within software encompasses a multifaceted approach that spans software developmental phases, with a focus on the software's ability to identify and expose vulnerabilities(Al-Matouq et al., 2020). This process necessitates a proactive approach integrated into the software's pre-deployment stages, aiming to empower the software development team in their pursuit of excellence, ultimately facilitating smoother operations. Even minor errors can have profound financial repercussions in today's business landscape, potentially resulting in multimillion-dollar losses. Unfortunately, many software development companies fail to adhere to Secure-DevOps Optimal approaches for incorporating security within the Software Developmental process (Wen & Katt, 2019). This negligence is often driven by factors like insufficient awareness, concerns about time and cost overruns, reliance on third-party components, and a shortage of qualified professionals (Dapshima, 2024). Moreover, the growth of online applications has amplified the quantity of security weaknesses. (Ferdiansyah et al., 2023).

The SDLC serves as a mechanism for generating top-notch, cost-effective software within the shortest period possible. It presents a well-structured sequence of steps that greatly facilitates enterprises in creating high-quality, thoroughly tested, and immediately deployable software. (Cruz Zapata et al., 2018), opted that, all stages of software development are interdependent and equally significant. By embedding security measures throughout every phase of the SDLC, the resulting product becomes impervious to security threats. Achieving this level of security requires adhering to a secure SDLC process, which ensures that security-related activities are seamlessly integrated into the complete development process (Rai, 2020)."

Conventional security methods primarily center around safeguarding network systems and involve significant financial investments to ensure network security. These methods encompass intrusion detection systems (IDs), firewalls, encryption, antivirus software, and antispyware tools(Sibi Chakkaravarthy et al., 2019). Moreover, security is often treated as a secondary concern, typically addressed post-development through the "penetrate and patch" (P&P) approach, which involves creating patches for identified vulnerabilities. However, a major setback of this technique is that users frequently neglect to apply these patches, leaving room for potential attackers to exploit new vulnerabilities (Saleem & Naveed, 2020). According to a Risk IQ report, major companies face security vulnerability-related costs of up to $25 per minute, while crypto companies experience losses of nearly $2000 per minute due to cybercrimes (Saleem & Naveed, 2020). Another report from Positive Technologies reveals that nine out of ten online apps remain susceptible by dangers related to security, approximately thirty-nine percent every web domain can be accessed without authorization, and 64% of applications are at risk of data breaches (Saleem & Naveed, 2020). This report attributes 82% of vulnerabilities to code flaws and provides historical data on the severity of web application vulnerabilities. These findings highlight the pressing issue of security in today's era, emphasizing the need for careful consideration during software development life cycles (Tung et al., 2016).

Security risks and vulnerabilities are increasingly alarming within the software industry. Despite ongoing initiatives to minimize these vulnerabilities through improvements in development methods and tools, the number of vulnerabilities and the subsequent computer security incidents stemming from their exploitation remain alarmingly high. Thus, considering such a threat, the paper titled ‘’Evaluation and Assessment of Security Risks and Vulnerabilities within Secure DevOps was proposed.

2. Review of Related Literature Review

2.1 Empirical Studies on Secure-DevOps to SDLC

In order to highlight the suggested practices for producing high-quality software, we will highlight recent research on the subject of secure software development in the "Literature Review" section. This review will also prepare the ground for introducing the suggested structure.

A comprehensive security framework by (Arizon-Peretz et al., 2022), is recommended for usage SDLC. Using this framework, In accordance with secure-related procedures along with SDLC standards, secure examinations and recommendations are developed(Bodin & Golberg, 2021). The analysis findings showed that this strategy provides dependable service with improved security and reliability.

Similarly, an extensive examination of the existing literature was conducted from various angles to pinpoint the most efficient methods for developing secure software (Humayun et al., 2022).
A systematic review of the literature was undertaken by (Hamid et al., 2020), pinpointing the most effective requirement practices, they introduced the RESMM (Requirements Engineering Security Maturity Model). Through usage of questionnaires and case studies, they tested this framework, and the outcomes indicate its usefulness and adaptability.
(Sugiantoro et al., 2020) Stated that the overall software development life cycle (SDLC) often neglects security considerations, leading to numerous failures to security. The study presents an appropriate methodology, a synthesis of flexible secure DevOps practices, to solve this challenge within the development of online applications. The design process is divided into three stages: conception, building, and transformation. The article also categorizes the weaknesses in security as well as frequent issues along with challenges faced during the development of online applications. They assess the suggested structure using a survey approach, and the findings are satisfactory.
As stipulated by (Alenezi & Almuairfi, 2019), it is recommended to adhere to best practices in risk management to promote the creation of high-quality software.
(Alferidah & Jhanjhi, 2020), stress that, security holds a significant role in the software development process. Yet, past research has not clearly addressed its integration into the SDLC.
(Sotos Martínez et al., 2022) concludes that, possessing a deep comprehension of software and employing appropriate application methods leads to the creation of dependable and high-quality software. The research pinpointed issues that arose during security incorporation into the software developmental life cycle, along with suggesting viable remedies. Additionally, it delved into specific security concerns crucial for crafting secure software.
An exploration of security issues throughout various software developmental processes was conducted by (Alvi & Zulkernine, 2021), with evaluation conducted in collaboration with scientific society as well as software developers. The findings of the survey are analyzed by the Statistical Package for Social Sciences software, resulting in the proposition of security guidelines for diverse stages of SDLC.
As indicated by (Dapshima et al., 2023; Goutam & Tiwari, 2019), security was previously an afterthought for an extended period. Thus, such strategies is deemed inadequate with regard to modern swiftly evolving software business. Integrating Secure-DevOps practice right from the inception to the culmination of the software development process has becomes a necessity. The paper provides an outline of security strategies applicable across different phases of SDLC, underlining the pivotal role of effective governance in project success.
(Khan et al., 2021) Executed a systematic mapping study in figuring out prevailing secure-DevOps methodologies employed toward software developmental processes. The research used 118 studies as its main references and extracted 52 security methods from the findings of these chosen studies. The study's results showed that most of these security measures are put into place during the coding phase.
Above preceding discussion makes it clear that the use of Secure-DevOps process into various stages of SDLC is essential for ensuring high-quality software. Although there have been numerous studies emphasizing the significance of integrating security into SDLC, there remains room for further exploration in this area.

2.2 Developmental Life Cycle of Software

It is an approach that encompasses a comprehensive blueprint outlining the steps for creating, upkeeping, replacing, and enhancing particular software. This life cycle establishes a systematic approach to enhance both software quality and the overall development process (Alnaseef et al., 2023).

2.2.1 Phases of Software Development Life Cycle

Although the approaches, techniques, and viewpoints regarding the creation of efficient and adaptable software services have evolved, the duties and steps involved have remained constant. It comprises an essential series of steps established for teams engaged in the creation and delivery of top-notch software (Jayalath et al., 2020). The individual phases of the SDLC are discussed as follows.

2.2.1.1 Planning and Requirement Analysis Phase.

The initial phase of software development, known as planning and requirement analysis. In this stage of the project, the team is tasked with recognizing, collecting, and outlining the existing issues, needs, requests, and customer anticipations associated with the software application or service. Various tasks linked to the requirements gathering stage may encompass the formulation of software specifications, development of a comprehensive plan, documentation, tracking of issues etc.

Furthermore, during the planning phase, preparations are made for quality assurance requirements and the identification of potential project risks.

2.2.1.2 Designing Phase

During the design stage of the project, the team is responsible for making critical decisions regarding the software solution's architecture and construction. This process may encompass the creation of design documents, the establishment of coding standards, and deliberations on the tools, methodologies, runtimes, or frameworks necessary to align with the software requirements and objectives identified during the initial requirements-gathering phase.

2.2.1.3 Development Phase

During this phase of work, teams construct software solutions according to the design choices that have been established. During this phase, teams aim to achieve the objectives and results that were established in the software requirements gathering stage by putting the solution into practice. The development process can encompass various groups of individuals, novel technologies, as well as unanticipated difficulties.

2.2.1.4 Testing and Integrating Phase

The code for execution is been assembled as well as validated throughout the testing and integration stages in order to prove its quality. High-quality inspection and testing are performed to ensure that the deployed products fulfill the specified quality and performance criteria. This portion of the process includes duties like testing individual components, integrating and completing testing, validating and verifying code, and identifying and disclosing any form of software issue or flaw."

2.2.1.5 Deployment Phase

The project's content is introduced into a real-world production environment at this project phase. It is the outcome of a team's efforts in gathering, designing, generating, and testing, and then displayed to the software service's clients and users. This step entails setting the necessary infrastructure, whether on-site or in conjunction with a cloud-based service, as well as developing an approach for distributing new versions of software to clients.

2.2.1.6 Operational and Maintenance Phase

During this phase, tasks may include examining, comprehending, and supervising network configurations, infrastructure settings, and the performance of application services in a live production environment. This procedure may also encompass addressing and overseeing incidents that arise because of any adjustments or modifications that affect the customer or user base.

2.3 Importance of Software Development Life Cycle

Managing software development can pose difficulties because of evolving demands, technological advancements, and the need for collaboration across various functions. To address these challenges, the software development lifecycle (SDLC) methodology offers a structured management framework with clearly defined milestones at each phase of the software creation process (Kudriavtseva & Gadyatskaya, 2023). Consequently, all parties involved reach a consensus on the objectives and requirements of the software development in advance, along with a roadmap to accomplish these objectives.
Here are several advantages associated with SDLC:

- Enhanced transparency of the development process for all stakeholders
- Task prediction, planning, and scheduling are more accurate.
- Improved control over risks and cost estimations.
- Methodical software delivery and heightened customer satisfaction.

2.4 Security Development Operation (Sec-DevOps)

Sec-DevOps encompasses a set of practices, cultural principles, and tools that unite software security (Sec) development (Dev) and IT operations (Ops), with the aim of enhancing an organization's capacity to deliver applications and services quickly and securely. When applying DevOps methodologies, it becomes possible to expedite the delivery of new application features and incorporate frequent incremental updates(Akbar et al., 2022). It involves implementing internal security measures as an integral part of the DevOps process. It emphasizes improved teamwork and coordination among departments like development, operations, and security teams throughout the software development cycle. By utilizing tools like automated security testing and ensuring that all integration and deployment pipelines have proper security measures in place, this approach fosters a culture of collective responsibility for security(Akbar et al., 2022).
The term "Secure DevOps" is frequently employed to refer to procedures that involve incorporating security assessments and evaluations at various stages of software developmental lifecycle (Akbar et al., 2022).
In Sec-DevOps, security becomes an integral part of the entire development process, from its inception to completion. This means that security is seamlessly incorporated into all aspects of software design and operation. Consequently, security issues are detected and addressed at each stage of the development pipeline before they can evolve into vulnerabilities that malicious actors could exploit(Ruggieri et al., 2019) .
Sec-DevOps places security as a paramount element within the DevOps process. This results in quicker releases and more dependable implementations, all with enhanced security. It adopts a people-centric approach to software development, fostering effective collaboration among development, operations, and security teams, all working towards a shared objective.
Historically, in traditional approaches like the waterfall model and conventional DevOps, security was often an afterthought, either tacked on at the end or sidelined to expedite software deployment and release. In essence, it was frequently not given the priority it deserved (Yungaicela-Naula et al., 2022). Nevertheless, because of 90% of data breaches originating from flaws in application code, companies are

currently seeking more effective methods to incorporate security into their development processes in order to mitigate vulnerabilities and expenses.

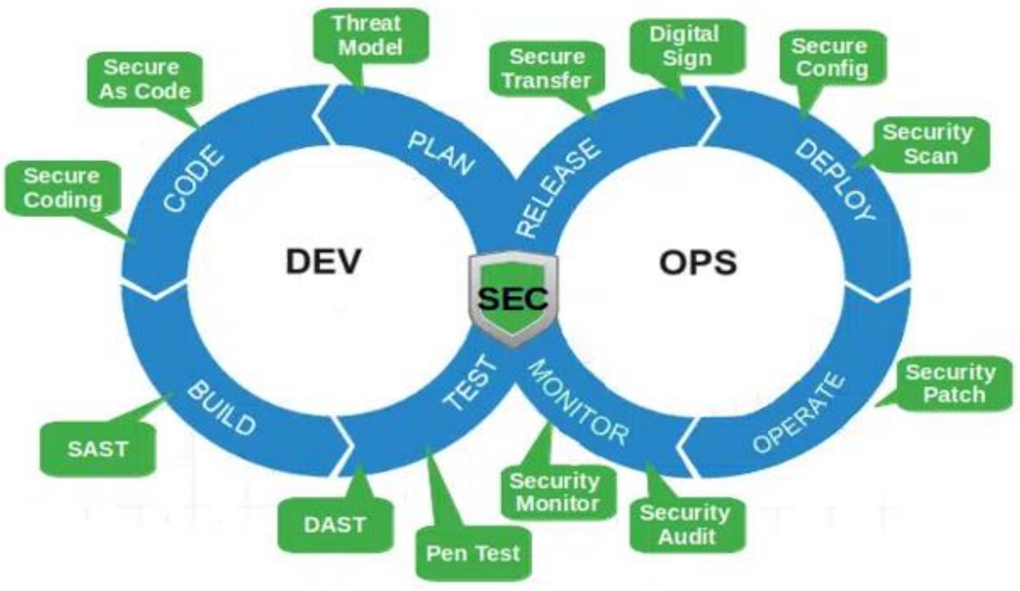

Figure 1. Sec-DevOps Process Flow.

2.5 Importance of Sec-DevOps in Software Development Life Cycle

The rise in cyber-attacks by 31% per company in 2021 and the staggering $4.24 million average cost of data breaches during the same year have made it evident that organizations are integrating application security into the software development process right from the start(Kritikos et al., 2019).

Secure DevOps enhances overall code security, saving time and money while also enhancing marketability (Miller & Heymann, 2018). It achieves this by:

- Reducing costs through early vulnerability detection in the development phase allows for swift mitigation, thus minimizing the need for last-minute fixes.
- Preventing poor encryption quality and insecure APIs by adhering to secure coding standards and guidelines defined by secure DevOps.
- Continuously improving results through root cause analysis in every development cycle.
- Promoting collaboration and communication among teams, breaking down silos.
- Increasing agility in responding to changes promptly.
- Enhancing the speed and adaptability of the security team.
- Providing more opportunities for quality and safety testing, automated builds, and optimizing code security without delaying release delivery cycles.
- Adding value through automation, freeing up team members to focus on high-priority tasks.
- Ensuring compliance with industry regulations such as GDPR and PCI DSS.
- Encouraging innovation by enabling teams to explore new ways of working and collaborating within a dynamic environment.
- Improving productivity by allowing developers to work more efficiently within the integrated security framework. With security addressed in each iteration, concerns at the end of the development cycle become unnecessary.

- Strengthening security and reducing the risks of breaches or ransomware attacks, leading to cost savings.
- Building customer trust in your products and services, thereby enhancing your brand's reputation.

2.5.1 Component of Sec-DevOps

1. **Security as code (SaC),** which involves integrating security directly into the existing tools within the DevOps pipeline, emphasizing automation. This essentially means it is a tool that automatically scans only the modified sections of the code, rather than the entire codebase(Miller & Heymann, 2018).
2. **Infrastructure as code (IaC),** which defines the DevOps tools employed to establish and update infrastructure components. These tools include Salt Stack, Puppet, Chef Automation Platform, and Red Hat Ansible. With IaC, infrastructure configuration is represented as a code file, making it easily modifiable and distributable (Assal & Chiasson, n.d.). In the event of system issues, it is replaced with a new one instead of resorting to traditional manual configuration changes or one-off scripts.

2.6 Causes of Security Flaws in Software Development Life Cycle

An explanation of a well-organized classification of the primary causes behind security flaws in the Software Development Life Cycle is provided in Table 1

Table 1. Causes of Security flaws in SDLC

S/N	SDLC PHASES	CAUSES OF SECURITY FLAWS
1	Planning and Requirements	+ Inadequate or ambiguous requirements for security + Non-threat modeling throughout the requirement assessment
2	Designing	+ Improper security set-ups during architectural and design phase + Insufficient security considerations in the system's architecture + Inadequate measures regarding protection and encryption of data
3	Development	+ Unsafe coding techniques + Using outdated or insecure libraries + lack of security testing as well as code review + Lack of code guidelines with a security emphasis.
3	Testing	+ Penetration testing and security testing are inadequate + Inability to recognize and fix security flaws
5	Deployment	+ Poorly managed configuration + Incorrectly set up security measures + Not applying updates and patches + Unsafe deployment techniques
6	Maintenance	+ Continued maintenance without taking security measures. + Inadequate knowledge and understanding of security measures. + Poor or sluggish reaction to security-related incidents

3. Methodology

A mathematical model has been introduced in assessing and evaluating the security risks as well vulnerabilities associated with the Software Developmental processes, Life Cycle process, in compliance with Secure-DevOps. This model offers a structured approach to identifying and mitigating potential security threats, enhancing overall software security practices. Figure 2. Capture Proposed Framework for Secured Software Development Life Cycle

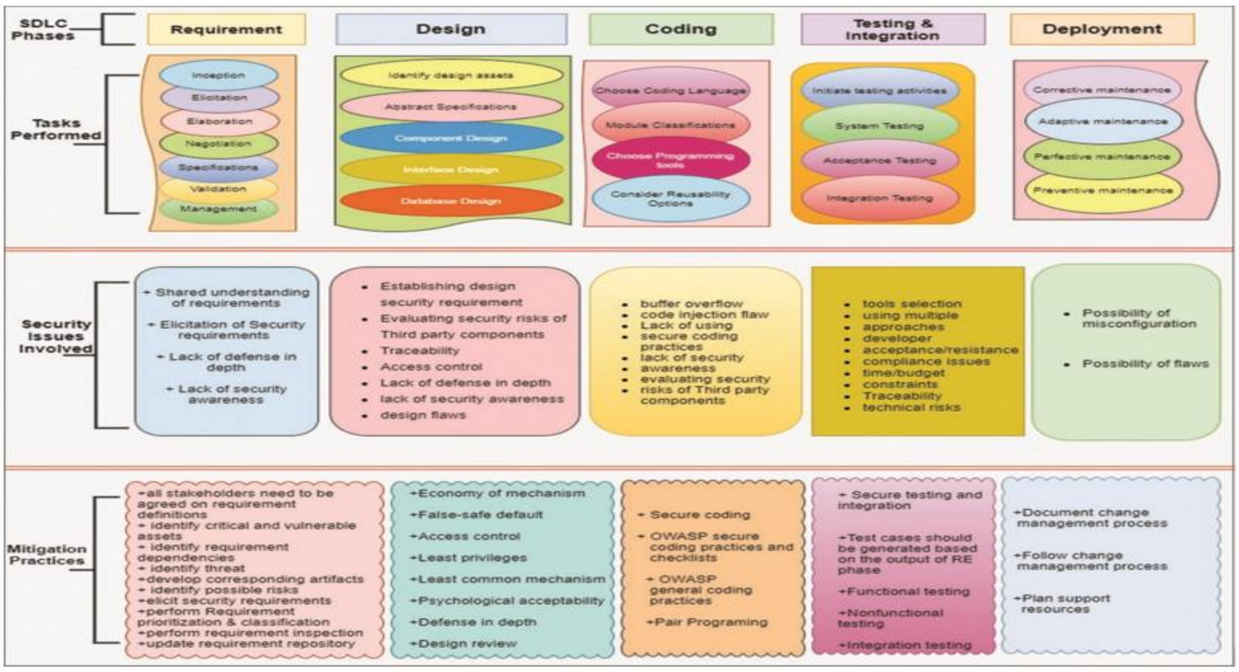
SDLC Phases
Requirement
Design
Coding
Testing & Integration
Deployment
Tasks Performed
Inception
Elicitation
Elaboration
Negotiation
Specifications
Validation
Management
Identify design assets
Abstract Specifications
Component Design
Interface Design
Database Design
Choose Coding Language
Module Classifications
Choose Programming tools
Consider Reusability Options
Initiate testing activities
System Testing
Acceptance Testing
Integration Testing
Corrective maintenance
Adaptive maintenance
Perfective maintenance
Preventive maintenance
Security Issues Involved
Shared understanding of requirements
Elicitation of Security requirements
Lack of defense in depth
Lack of security awareness
Establishing design security requirement
Evaluating security risks of Third party components
Traceability
Access control
Lack of defense in depth
lack of security awareness
design flaws
buffer overflow
code injection flaw
Lack of using secure coding practices
lack of security awareness
evaluating security risks of Third party components
tools selection
using multiple approaches
developer acceptance/resistance
compliance issues
time/budget constraints
Traceability
technical risks
Possibility of misconfiguration
Possibility of flaws
Mitigation Practices
all stakeholders need to be agreed on requirement definitions
identify critical and vulnerable assets
identify requirement dependencies
identify threat
develop corresponding artifacts
identify possible risks
elicit security requirements
perform Requirement prioritization & classification
perform requirement inspection
update requirement repository
Economy of mechanism
False-safe default
Access control
Least privileges
Least common mechanism
Psychological acceptability
Defense in depth
Design review
Secure coding
OWASP secure coding practices and checklists
OWASP general coding practices
Pair Programing
Secure testing and integration
Test cases should be generated based on the output of RE phase
Functional testing
Nonfunctional testing
Integration testing
Document change management process
Follow change management process
Plan support resources

Figure 2. Proposed Framework for Secured Software Development Life Cycle

4. Analysis and Discussion of Results

4.1 Algorithm of the Proposed Methodology

Incorporating security best practices right from the project's outset through its deployment is essential in Sec-DevOps. These practices not only enhance the security and quality of the software but also have a minimal impact on the project's budget and timeline. The detailed methodology is outlined in the algorithm below.

For the understanding of the algorithm, the following items are abbreviated as follows:

The requirements Phase is **RQ, the** Design Phase is **DG, the** Implementation Phase is **I, the** Testing Phase is **TI, the** Maintenance Phase is **MT, the** Security Risk is **SR,** and **the** Artifacts is **AF.**

Table 2. Algorithm of the Proposed Methodology

SN	COMMAND	MEANING
1	Start	
2	Apply ESRP ( )	// Use Engineering Secure Requirement Process
a.	Accept RQ	// All parties involved must reach a consensus on the definitions of requirements.
b.	Determine ACS	// Determine assets that are both crucial and susceptible
c.	Determine DR	// Determine Dependencies in Requirements.

d.	Determine SR	// Security Risk
e.	Create AF	// Create relevant Artifacts
f.	Recognize PH	// Recognize potential hazards
g.	Generate SR	//Generate Security Requirement
h.	Execute PCR	// Prioritize and categorize requirements
i.	Conduct ER	// Conduct a thorough examination of the requirements.
j.	Revise RR	// Revise the Requirement Repository.
3	If Steps from 'a' to 'j' achieve success, proceed to Step 5.	
4	Else Proceed to step 2	
5	Develop SDS ()	// Develop Secure Design Software
a.	Employ PEM	// Utilize the principle of Economy of Mechanism and strive to maintain a design that is as straightforward as possible.
b.	Employ PFSD	// Utilize the principles of False-Safe Default to ensure that any activity's failure prevents unsafe operations.
c.	Implement ACM	// Implement an Access Control Mechanism to ensure that each item undergoes authorization verification
d.	Grant MP	// Grant the minimum necessary permissions.
e.	Pursue LCM	// Least Common Method for limiting access to shared resources
f.	Adhere PAM	// Designs that are psychologically acceptable inherently include fundamental security measures.
g.	Apply DD	//Multilevel security is included in Defense in Depth
h.	Carryout DR	// To validate the design, a design review should be undertaken.
6	If steps A through H are completed successfully, proceed to step 8	
7	Else proceed to step 5	
8	Conduct SCP ()	// Conduct Secure Code Practice (SCP).
A	Adhere to SCPs	// Adhere to the Secure Code Practices
B	Adhere to GCPs	// Adhere to the General Coding Practices
C	Engage in PPs	// Engage in Pair Programming when it is feasible.
9	If Steps A through C prove to be successful, proceed to Step 11	
10	If not, proceed to step 8	
11	Conduct TII	// Conduct testing and integration in a secure manner.
a.	Create TCs	// Test cases should be created based on requirement phase results
b.	Conduct FTs	// Conduct testing that assesses the functionality
c.	Conduct NFTs	// Conduct testing for nonfunctional aspects
d.	Conduct Its	// Conduct Integration Testing
12	If Steps A through D are completed successfully, proceed to Step 14	
13	Otherwise, Proceed to 11.	
14	Execute M ().	
a.	Process MDC	//Process for Managing Document Changes
b.	Adhere PMC	//Adhere to the process of managing changes.

c.	Plan SRs	// Planning to support Resources
15	If Steps A through C are completed successfully, proceed to Step 17.	
16	Else, Proceed to 14	
17	Stop	

4.2 Evaluation of Frameworks through the Application of Mathematical Models

Before delving into mathematical modeling, we will initially establish the symbols and terminology employed within the mathematical model to enhance our comprehension. Table 3 provides a comprehensive list of the symbols and terms utilized in our mathematical model.

Table 3. Symbols employed in the mathematical model.

S/N	PARAMETERS	SYMBOL
1	Secure-DevOps	SDO
2	Requirement Stage	RQ
3	Designing Stage	DG
4	Implementation Stage	IS
5	Testing and Integration Stage	TIS
6	Deployment Stage	DS
7	Security	SC
8	Threshold /Cut-off Value	TH
9	Functional Objectives	FO

Based on the suggested framework, it is essential for a SDO to integrate security throughout the entire SDLC process, as illustrated in Equation 1.

$$SDO = f(RQ\ DG\ IS\ TIS\ DS) \qquad (1)$$

Because the desired approach strives to increase confidentiality, the main purpose is represented as depicted in Equation 2.

$$FO = Max(SC) \qquad (2)$$

Adhering to the suggested guidelines outlined in the proposed framework can improve the requirement plan effectively. Consequently, it is possible to represent the requirement phase in the form of a model.

$$Rs = \sum_{a=1}^{n} Aa \qquad (3)$$

The requirement phase's security relies on adhering to a set of best practices, denoted as $A_1, A_2, \ldots A_n$. To assess the phase's security, it's essential to assign weights, $C_1, C_2, \ldots C_n$, to each best practice and establish a threshold value for RQ. The measurement of RQ can be calculated using the formula specified in Equation 4.

$$\frac{1}{n}(A_1\, G_1 + A_2 C_2 + \cdots A_n\, C_n \geq TH\ (Rs) \qquad (4)$$

For a successful requirement phase, Equation (4) needs to hold true, with the specific value of TH being project-dependent and determined by the project's characteristics. Similarly, adhering to optimal design practices, as presented in Equation. (5), can lead to the attainment of a secure design.

$$DG = \sum_{b=1}^{n} B_b \qquad (5)$$

The recommended guidelines for ensuring the security of software design are denoted as B_1, B_2, B_3, …Bn. To assess the safety of the software architecture, the firm has to ascertain the importance of TH according to the project's characteristics and allocate weights to each C_1, C_2, C_n, and so forth. Subsequently, the security of the design can be gauged by applying the formula specified in Equation (6).

$$\frac{1}{n}(B_1 G_1 + B_2 C_2 + \cdots B_n C_n \geq TH\ (DG) \qquad (6)$$

To effectively complete the secure software design step, Equation. (6) must be met. The organization moves on to the coding step after finishing the requirements and design stages.

$$I = \sum_{c=1}^{n} Dc \qquad (7)$$

In the coding process, it is essential, taking into account a set of secure coding practices labeled as D_1, D_2, D_3,..Dn.

$$\frac{1}{n}(D_1 G_1 + D_2 C_2 + \cdots D_n C_n \geq TH\ (I) \qquad (8)$$

The cumulative score of coding practices should exceed a predetermined security threshold. After ensuring secure coding, we proceed to Integration and testing stage. It is paramount the software importance since any defects or issues must be resolved here before delivering the software to the customer upon its completion.

$$TIS = \sum_{d=1}^{\mathrm{n}} E_d \qquad (9)$$

The security best practices, labeled as E_1, E_2, E_3,...En must be integrated into TIS for insurance of its security.

$$\frac{1}{\mathrm{n}}(E_1 G_1 + E_2 C_2 + \cdots E_n C_n \geq TH\ (TIS) \qquad (10)$$

The culmination of effective testing and integration practices should yield a security level surpassing the predetermined threshold value in order to ensure the success of the final phase. In this last stage, products are deployed to their immediate working environment accommodating any user-requested changes. It is imperative to meticulously plan this phase to prevent any user dissatisfaction or operational inconsistencies. The suggested model advocates adherence to a set of optimal practices during deployment, as detailed in Equation (11).

$$DS = \sum_{e=1}^{n} Fe \qquad (11)$$

It is critical to consider the best practices represented by F_1, F_2, F_3, and so forth during the Software Development Life Cycle deployment phase. The equation given in twelve is used to assess the security of this phase.

$$\frac{1}{n}(F_1 G_1 + F_2 C_2 + \cdots F_n C_n \geq TH\ (DS) \qquad (12)$$

The cumulative security level, determined by integrating each phase, should surpass the predetermined threshold value to ensure a securely executed deployment phase. Once all SDLC phases prioritize security and integrate best practices accordingly, organizations should evaluate the project's overall security against a project-specific threshold, which can be calculated using,

$$CUMSC = \frac{1}{n}\left(AC + BC + DC + EC + FC\right) \qquad (13)$$

The cumulative security should exceed the specified cumulative threshold value.

$$CUM(SC) \geq CUM(TH) \qquad (14)$$

After implementing the recommended guidelines outlined in the proposed framework, when the organization attains a collective level of security for its software development, the resulting software becomes impervious to security vulnerabilities and threats.

4.3 Evaluation of Guidelines via Research Study

Integrity is a vital component of every technology, though its significance can vary from one project to another. While the traditional security method known as Penetrate and Patch can be costlier and intricate, integrating security throughout the entire Software Developmental stages is of utmost importance. Various approaches have been devised for integrating security into Software development, but challenges persist. To address the challenges, a Secure-DevOps strategy emphasizes starting security measures from the project's inception to the point the developed software deployed becomes fully operational.
ABC management adheres to this framework and customizes parameters based on the specific project's characteristics. They employ a conceptual mapping method in assessing the implementation of Secure-DevOps across the developmental processes of software. The concept involves identifying key practices and enlisting the input of the production crew to outline them according to their comprehension. This method is particularly useful for evaluating qualitative aspects(Nina et al., 2021).

Table 4. Security Measures Proposed by Secure-DevOps based on a case study.

X	Security Measures Proposed by Secure-DevOps	Y	Z
	REQUIREMENT PHASE		
x_1	All parties involved must reach a consensus on the definitions of requirements.	0.85	0.90
x_2	Determine assets that are both crucial and susceptible	0.95	0.93
x_3	Determine Dependencies in Requirements.	0.96	0.91
x_4	Determine Security Risk	0.95	0.93
x_5	Create relevant Artifacts	0.93	0.95
x_6	Recognize potential hazards	0.95	0.95
x_7	Generate Security Requirement	0.90	0.85
x_8	Prioritize and categorize requirements	0.95	0.90
x_9	Conduct a thorough examination of the requirements.	0.95	0.85
x_{10}	Revise the Requirement Repository.	0.95	0.90
	DESIGNING PHASE		
x_1	Utilize economy policy to maintain a design that is as straightforward as possible.	0.95	0.90
x_2	Utilize the principles of False-Safe Default to ensure that any activity's failure prevents unsafe operations.	0.95	0.85
x_3	Implement an Access Control Mechanism to ensure that each item undergoes authorization verification	0.90	0.85
x_4	Grant the minimum necessary permissions.	0.85	0.80

x_5	Use Least Common Method for limiting access to shared resources	0.85	0.75
x_6	Designs that are psychologically acceptable inherently include fundamental security measures.	0.85	0.90
x_7	Multilevel security is included in Defense in Depth	0.95	0.95
x_8	To validate the design, a design review should be undertaken.	0.95	0.95
	IMPLEMENTATION PHASE		
x_1	Conduct Secured programming, adhering to secured coding criteria and implementing recommended approaches.	0.95	0.85
x_2	Adhere to the Secure Coding Practices and utilize the associated checklists.	0.85	0.75
x_3	Adhere to the General Coding Practices	0.85	0.80
x_4	Engage in Pair Programming when it is feasible.	0.80	0.55
	TESTING AND INTEGRATING PHASE		
x_1	Conduct testing and integration in a secure manner.	0.85	0.85
x_2	Test cases should be created based on requirement phase results	0.95	0.95
x_3	Conduct testing that assesses the functionality	0.95	0.98
x_4	Conduct testing for nonfunctional aspects	0.85	0.90
x_5	Conduct Integration Testing	0.95	0.98
	DEPLOYMENT PHASE		
x_1	Process for Managing Document Changes	0.95	0.85
x_2	Adhere to the process of managing changes.	0.90	0.85
x_3	Planning to support Resources	0.85	0.85

In Table 4, X is identified as the optimal security procedure, Y represents the significance of executing this security measure, and Z indicates the outcome derived from implementing these measures.
Regarding the security measures suggested by Secure-DevOps for the Software Development Lifecycle, a specific threshold value is allocated to each phase of the SDLC. This value will be utilized to assess the effectiveness of the security measures implemented throughout the entire software development process, as recommended by Secure-DevOps. Table 5 captures the allocated threshold value for each stage of the SDLC process.

Table 5. An allocated threshold value for the SDLC process

S/N	STAGE OF SDLC	ALLOCATED THRESHOLD VALUE
1	Requirement Stage	0.80
2	Designing Stage	0.70
3	Implementation Stage	0.60
4	Testing and Integration Stage	0.65
5	Deployment Stage	0.60
6	Cumulative Security	0.65

By integrating Secure-DevOps in the Requirement Stage,

$$SC(RQ) = \frac{1}{n}(A_1\,G_1 + \;A_2C_2 + \cdots A_n\,C_n\,) \qquad (15)$$

Substituting the values from Table 4 into the equation 15,

$$SC(RQ) = \frac{1}{10}\begin{pmatrix}(0.90)(0.85) + (0.93)(0.95) + (0.91)(0.96) + (0.93)(0.95) + (0.95)(0.93)\\(0.95)(0.95) + (0.85)(0.95) + (0.90)(0.95) + (0.85)(0.95) + (0.90)(0.95)\end{pmatrix}$$

$$= \frac{1}{10}\langle 8.5641\rangle \quad = \; 0.86$$

The collective worth of SC(RQ) amounts to 0.86, surpassing 0.80, indicating SCI(RQ) > TH(RQ). This illustrates that integrating Secure-DevOps approaches during the RQ stage enhances the reliability of security.

Subsequently, equation 16 below is used to evaluate the level of security during the design phase.

$$SC(DS)\;\frac{1}{n}(B_1\,G_1 + \;B_2C_2 + \cdots B_n\,C_n) \qquad (16)$$

Substituting the values from Table 4 into the equation above,

$$SC(DS) = \frac{1}{8}\begin{pmatrix}(0.90)(0.95) + (0.85)(0.95) + (0.85)(0.90) + (0.80)(0.85) +\\(0.75)(0.85) + (0.90)(0.85) + (0.95)(0.95) + (0.95)(0.95)\end{pmatrix}$$

$$= \frac{1}{8}\langle 6.315\rangle = \; 0.79$$

The design phase SC(DS)) is 0.79 exceeding 0.70. Meaning SC(DS)>TH(DS).

To assess the level of security at the implementation stage. Equation 17 will be utilized as shown below.

$$SC(I) = \;\frac{1}{n}(D_1\,G_1 + \;D_2C_2 + \cdots D_n\,C_n\,) \qquad (17)$$

Substituting the values from Table 4 into the equation above,

$$SC(I) = \;\frac{1}{4}((0.85)(0.95) + (0.75)(0.85) + (0.80)(0.85) + (0.55)(0.80)\,)$$

$$= \frac{1}{4}\langle 2.565\rangle = \; 0.64$$

The security value achieved during the Implementing phase is 0.64, slightly surpassing the 0.60 threshold. This underscores the importance of integrating Secure-DevOps into implementation phase, as outlined in the suggested framework for organizations.

The formula denoted in equation 18 is employed to compute the security metrics during the testing and integration stage.

$$SC(TIS) = \frac{1}{n}(E_1\,G_1 + \;E_2C_2 + \cdots E_n\,C_n \qquad (18)$$

Substituting the values from Table 4 into the equation above,

$$SC(TIS) = \;\frac{1}{5}((0.85)(0.85) + (0.95)(0.95) + (0.98)(0.95) + (0.90)(0.85) + (0.98)(0.95)\,)$$

$$= \frac{1}{5}\langle 4.235\rangle \; = \; 0.85$$

The implementation of security best practices according to the proposed framework during the testing and integration stage resulted in a security value of 0.85, surpassing the 0.65 threshold. This indicates that the suggested framework effectively enhances the security of the SDLC's testing and integration stage.

In the deployment stage, the security level is calculated using Equation 19 as shown below;

$$SC(DS) = \frac{1}{n}(F_1\, G_1 + \ F_2 C_2 + \cdots F_n\, C_n \qquad (19)$$

Substituting the values from Table 4 into the equation above,

$$SC(D) = \ \frac{1}{3}((0.85)(0.95) + (0.85)(0.90) + (0.85)(0.85))$$

$$= \frac{1}{3}\langle 2.295 \rangle \ = \ 0.77$$

The security value achieved during the deployment phase is 0.77, surpassing the 0.60 threshold.

By integrating the Secure-DevOps concept throughout the developmental phases of software, equation 20 is utilized.

$$CUMSC = \frac{1}{n}(AC + BC + DC + ED + FC) \qquad (20)$$

Substituting the Security values obtained after following the best security practice proposed by Secure-DevOps in all phases of the SDLC, the cumulative Security is evaluated as follows;

$$CUMSC = \ \frac{1}{5}(0.86 + 0.79 + 0.64 + 0'85 + 0.77)$$

$$= \frac{1}{5}\langle 3.91 \rangle \ = \ 0.78$$

The cumulative security value acquired is 0.78 surpassing the CUMTH of 0.65. The finding demonstrates the effectiveness of Secure-DevOps approach toward enhancement of confidentiality and reliability in developmental lifecycle of software from start to finish.

5. Conclusion

Detecting and addressing bugs and flaws early in the production phase is not only more cost-effective but also less complicated compared to identifying them at a later stage. Historically, many software projects failed due to negligence toward security. Evaluating a software's security post-development not only consumes time and adds complexity but also escalates the overall project time and costs. To prevent complications and project failures in the future, a framework based on Secure-DevOps best practices was initiated. The study's findings demonstrate that incorporating security best practices of Secure-DevOps into various phases of the SDLC leads to improved software security.

REFERNCES

1. Akbar, M. A., Smolander, K., Mahmood, S., & Alsanad, A. (2022). Toward successful DevSecOps in software development organizations: A decision-making framework. *Information and Software Technology*, *147*, 106894. https://doi.org/10.1016/j.infsof.2022.106894
2. Alenezi, M., & Almuairfi, S. (2019). Security Risks in the Software Development Lifecycle. *International Journal of Recent Technology and Engineering*, *8*, 7048–7055. https://doi.org/10.35940/ijrte.C5374.098319
3. Alferidah, D. K., & Jhanjhi, N. (2020). *A Review on Security and Privacy Issues and Challenges in Internet of Things*.
4. Al-Matouq, H., Mahmood, S., Alshayeb, M., & Niazi, M. (2020). A Maturity Model for Secure Software Design: A Multivocal Study. *IEEE Access*, *8*, 215758–215776. IEEE Access. https://doi.org/10.1109/ACCESS.2020.3040220
5. Alnaseef, F., Niazi, M., Mahmood, S., Alshayeb, M., & Ahmad, I. (2023). Towards a successful secure software acquisition. *Information and Software Technology*, *164*, 107315.

https://doi.org/10.1016/j.infsof.2023.107315

6. Alvi, A. K., & Zulkernine, M. (2021). A security pattern detection framework for building more secure software. *Journal of Systems and Software*, *171*, 110838. https://doi.org/10.1016/j.jss.2020.110838
7. Arizon-Peretz, R., Hadar, I., & Luria, G. (2022). The Importance of Security Is in the Eye of the Beholder: Cultural, Organizational, and Personal Factors Affecting the Implementation of Security by Design. *IEEE Transactions on Software Engineering*, *48*(11), 4433–4446. IEEE Transactions on Software Engineering. https://doi.org/10.1109/TSE.2021.3119721
8. Assal, H., & Chiasson, S. (n.d.). *Security in the Software Development Lifecycle*.
9. Bodin, N., & Golberg, H. K. B. (2021). *Software Security Culture in Development Teams: An Empirical Study* [Master thesis, NTNU]. https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2827053
10. Cruz Zapata, B., Fernández-Alemán, J. L., Toval, A., & Idri, A. (2018). Reusable Software Usability Specifications for mHealth Applications. *Journal of Medical Systems*, *42*(3), 45. https://doi.org/10.1007/s10916-018-0902-0
11. Dapshima, B. A. (2024). Constraints that Hinders Secure Software Implementation and Development Processes. *International Journal for Research in Applied Science and Engineering Technology*, *12*(6), 2400–2404. https://doi.org/10.22214/ijraset.2024.63494
12. Dapshima, B. A., Essa, Y. C., & Chaturvedi, Dr. S. (2023). Fault Detection and Protection of Power Transformer Using Fuzzy Logic. *International Journal for Research in Applied Science and Engineering Technology*, *11*(1), 1816–1824. https://doi.org/10.22214/ijraset.2023.48748
13. Ferdiansyah, D., Isnanto, R., & Suseno, J. E. (2023). Organizational indicators on startup software for implementing secure software development lifecycle (SSDL): A systematic literature review. *AIP Conference Proceedings*, *2683*(1), 050010. https://doi.org/10.1063/5.0125388
14. Goutam, A., & Tiwari, V. (2019). Vulnerability Assessment and Penetration Testing to Enhance the Security of Web Application. *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, 601–605. https://doi.org/10.1109/ISCON47742.2019.9036175
15. Hamid, M. A., Hafeez, Y., Hamid, B., Humayun, M., & Jhanjhi, N. Z. (2020). Towards an effective approach for architectural knowledge management considering global software development. *International Journal of Grid and Utility Computing*, *11*(6), 780–791. https://doi.org/10.1504/IJGUC.2020.110908
16. Humayun, M., Jhanjhi, N., Fahhad Almufareh, M., & Ibrahim Khalil, M. (2022). Security Threat and Vulnerability Assessment and Measurement in Secure Software Development. *Computers, Materials & Continua*, *71*(3), 5039–5059. https://doi.org/10.32604/cmc.2022.019289
17. Jayalath, L. M., C Dharshana, K. A., & Rathnayake, R. M. T. P. (2020). Towards Secure Software Engineering. *South Asian Research Journal of Engineering and Technology*, *2*(6), 45–53. https://doi.org/10.36346/sarjet.2020.v02i06.001
18. 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. IEEE Access. https://doi.org/10.1109/ACCESS.2021.3052311
19. Kritikos, K., Magoutis, K., Papoutsakis, M., & Ioannidis, S. (2019). A survey on vulnerability assessment tools and databases for cloud-based web applications. *Array*, *3–4*, 100011. https://doi.org/10.1016/j.array.2019.100011
20. Kudriavtseva, A., & Gadyatskaya, O. (2023). *Secure Software Development Methodologies: A Multivocal Literature Review* (arXiv:2211.16987). arXiv. http://arxiv.org/abs/2211.16987

21. Miller, B. P., & Heymann, E. (2018). Tutorial: Secure Coding Practices, Automated Assessment Tools and the SWAMP. *2018 IEEE Cybersecurity Development (SecDev)*, 124–125. https://doi.org/10.1109/SecDev.2018.00025
22. Nina, H., Pow-Sang, J. A., & Villavicencio, M. (2021). Systematic Mapping of the Literature on Secure Software Development. *IEEE Access*, *9*, 36852–36867. IEEE Access. https://doi.org/10.1109/ACCESS.2021.3062388
23. Rai, A. (2020). A Review of Information Security: Issues and Techniques. *International Journal for Research in Applied Science and Engineering Technology*, *8*(5), 953–960. https://doi.org/10.22214/ijraset.2020.5150
24. Ruggieri, M., Hsu, T.-T., & Ali, M. L. (2019). Security Considerations for the Development of Secure Software Systems. *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 1187–1193. https://doi.org/10.1109/UEMCON47517.2019.8993081
25. Saleem, H., & Naveed, M. (2020). SoK: Anatomy of Data Breaches. *Proceedings on Privacy Enhancing Technologies*, *2020*(4), 153–174. https://doi.org/10.2478/popets-2020-0067
26. Sibi Chakkaravarthy, S., Sangeetha, D., & Vaidehi, V. (2019). A Survey on malware analysis and mitigation techniques. *Computer Science Review*, *32*, 1–23. https://doi.org/10.1016/j.cosrev.2019.01.002
27. Sotos Martínez, E., Villanueva, N. M., & Orellana, L. A. (2022). A Survey on the State of the Art of Vulnerability Assessment Techniques. In J. J. Gude Prego, J. G. de la Puerta, P. García Bringas, H. Quintián, & E. Corchado (Eds.), *14th International Conference on Computational Intelligence in Security for Information Systems and 12th International Conference on European Transnational Educational (CISIS 2021 and ICEUTE 2021)* (pp. 203–213). Springer International Publishing. https://doi.org/10.1007/978-3-030-87872-6_20
28. Sugiantoro, B., Anshari, M., & Sudrajat, D. (2020). Developing Framework for Web Based e-Commerce: Secure-SDLC. *Journal of Physics: Conference Series*, *1566*(1), 012020. https://doi.org/10.1088/1742-6596/1566/1/012020
29. Tung, Y.-H., Lo, S.-C., Shih, J.-F., & Lin, H.-F. (2016). An integrated security testing framework for Secure Software Development Life Cycle. *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 1–4. https://doi.org/10.1109/APNOMS.2016.7737238
30. Wen, S.-F., & Katt, B. (2019). Learning Software Security in Context: An Evaluation in Open Source Software Development Environment. *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 1–10. https://doi.org/10.1145/3339252.3340336
31. Yungaicela-Naula, N. M., Vargas-Rosales, C., Pérez-Díaz, J. A., & Zareei, M. (2022). Towards security automation in Software Defined Networks. *Computer Communications*, *183*, 64–82. https://doi.org/10.1016/j.comcom.2021.11.014