IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 22, 2021, accepted November 29, 2021, date of publication January 5, 2022, date of current version January 14, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3140181

Systematic Literature Review on Security Risks and Its Practices in Secure Software Development

RAFIQ AHMAD KHAN[1], SIFFAT ULLAH KHAN[1], HABIB ULLAH KHAN[2], AND MUHAMMAD ILYAS[1]

[1]Software Engineering Research Group, Department of Computer Science & IT, University of Malakand, Chakdara 18800, Pakistan
[2]Department of Accounting and Information Systems, College of Business and Economics, Qatar University, Doha, Qatar

Corresponding authors: Habib Ullah Khan (habib.khan@qu.edu.qa) and Rafiq Ahmad Khan (rafiqahmadk@gmail.com)

This work was supported in part by the Qatar National Library, Doha, Qatar; in part by the Qatar University Internal Grant under Grant QUHI-CBE-21/22-1; and in part by the Department of Computer Science & IT, University of Malakand, Pakistan.

ABSTRACT Security is one of the most critical aspects of software quality. Software security refers to the process of creating and developing software that assures the integrity, confidentiality, and availability of its code, data, and services. Software development organizations treat security as an afterthought issue, and as a result, they continue to face security threats. Incorporating security at any level of the Software Development Life Cycle (SDLC) has become an urgent requirement. Several methodologies, strategies, and models have been proposed and developed to address software security, but only a few of them give reliable evidence for creating secure software applications. Software security issues, on the other hand, have not been adequately addressed, and integrating security procedures into the SDLC remains a challenge. The major purpose of this paper is to learn about software security risks and practices so that secure software development methods can be better designed. A systematic literature review (SLR) was performed to classify important studies to achieve this goal. Based on the inclusion, exclusion, and quality assessment criteria, a total of 121 studies were chosen. This study identified 145 security risks and 424 best practices that help software development organizations to manage the security in each phase of the SDLC. To pursue secure SDLC, this study prescribed different security activities, which should be followed in each phase of the SDLC. Successful integration of these activities minimizing effort, time, and budget while delivering secure software applications. The findings of this study assist software development organizations in improving the security level of their software products and also enhancing their security efficiency. This will raise the developer's awareness of secure development practices as well.

INDEX TERMS Software security, SDLC, security risks and practices, secure software development, secure software engineering, systematic literature review.

I. INTRODUCTION

Secure Software Engineering (SSE) has become a significant paradigm in the development of secure software for the software industry in recent years as security problems in the SDLC are difficult to address. Information and Communication Technology (ICT) has undeniably changed human lives, communications, the digital economy, socialization, and entertainment. Similarly, the market for internet-enabled applications is increasingly increasing. Therefore, there is an ever-growing demand for trusted software applications. Software security is the key to the software's success, especially in today's fast-paced and technology-oriented world. Software and technology have become such an inseparable part of our lives that it's virtually impossible to imagine a sector that doesn't employ them in its day-to-day operations. The world in every aspect has been modernized by an immense use of software systems. Software security ensures that the CIA (Confidentiality, Integrity, and Availability) of data and services are not compromised [1], [2]. This can only be done if the security is considered during all SDLC phases [1], [2].

To incorporate security into the software engineering paradigm, it should be considered from the start of the SDLC [3], [4]. Secure software engineering (SSE) is the process of designing, building, and testing software so that it becomes secure, this includes secure SDLC processes

The associate editor coordinating the review of this manuscript and approving it for publication was Luca Cassano.

and secure software development (SSD) methods [5]–[7]. Most businesses view security as a post-development process [8]. Security isn't considered at some point in the pre-development phase [9]. A simple error sometimes can end up causing millions of dollars of losses in today's business process. But unfortunately, many software development companies do not follow best practices to incorporate in SDLC [10], [11]. This negligence includes lack of awareness, fear of time and cost overrun, teams, are always in a hurry, use of third-party components and, lack of qualified professionals, etc. Rapid developments in information and communication technologies (ICTs) have made software security a key concern, such as the Internet of Things (IOT) and the Internet of Every Things, the advancement of Internet-based software systems, cloud computing, social networking, and location-based services; also besides, new business paradigms, versatile customers' requirements, rapid advancement in ICTs, and new regulations are constantly making a software application evolve accordingly [12], [13]. As software development becomes more complex, distributed, and concurrent, security issues have an ever-greater influence on software quality [14]. Insecure software harms an organization's reputations with customers, partners, and investors; it increases costs, as companies are forced to repair unreliable applications; and it delays other development efforts as limited resources are assigned to address current software deficiencies [14]. The majority of software programs are designed and deployed without attention to protection desires [15], [16]. Hidden attacking risks within or outside the organization are emerging day-by-day, results in huge financial loss, as well as confidentiality and credibility losses by putting the availability and integrity of organizational data at risk [17], [18]. Various approaches to software quality have been developed, such as CMMI, ''Microsoft Software Development Life Cycle (MS-SDL),'' ''Misuse case modeling,'' ''Abuse case modeling,'' ''Knowledge Acquisition for Automated Specification,'' ''System Security Engineering-Capability Maturity Model (SSE-CMM),'' ''OWASP,'' and ''Secure Tropos Methodology'' [19]. However, there exists no explicit solution for incorporating security into all phases of SDLC.

One of the critical reasons for widespread vulnerabilities is not making security a key priority [2]. Even diligent businesses use the ''fix and penetrate'' technique in which security is accessed after completing the project [2]. The drawback of this is that the application users do not apply these patches. Further, attackers might plan and penetrate new vulnerabilities [20]. Traditional security mechanisms mainly focus on network systems, and they spent a huge amount of money to make their network secure. These mechanisms include IDS (Intrusion detection system), firewalls, encryption, antivirus, and antispyware [5], [21]. Building secure software means building software that functions properly even under malicious attacks [22]. This requires addressing the security challenges through the whole SDLC, especially in the early stages during the design phase [23]. This reduces the risk of overlooking critical security requirements or introducing security flaws throughout the implementation process.

SDLC is a process for producing high-quality, low-cost applications in the shortest amount of time. It offers a well-structured step flow that assists enterprises in easily produce high-quality, well-tested, and ready-to-use production of software. The common phases of SDLC include requirement, design, coding, testing, deployment, and maintenance [24]. All these phases are dependent on each other are of equal importance. If security is not incorporated during all phases of SDLC then the resultant product will not be vulnerable to security threats. This is only possible if a secure SDLC process is followed, secure SDLC ensures that security-related activities are an integral part of the overall development effort [20], [22], [25].

Researchers in the literature [26]–[30] have introduced and practitioners in the software industry have adopted a wide variety of software security practices, approaches, and methods. In addition, several companies have created maturity models and frameworks to assess the degree of maturity of their software security practices. On the other hand, none of these models or structures are specifically committed to recognizing security risks/threats and their practices in the SDLC. As a result, they fall short of covering all aspects and activities of a secure SDLC. Because of the importance of a secure SDLC, it's critical to recognize the security threats that vendor organizations face while developing secure applications, as well as risk mitigation strategies. This will enable software development vendors to assess their maturity and assurance levels, as well as improve their secure SDLC performance. It will also raise the level of awareness among software engineers.

Therefore, to assess and find out security threats and their practices in SDLC phases, we have studied the existing literature on finding software security threats/risks in SDLC and highlighted the security practices that need to be incorporated in SDLC phases to strengthen the security of the software development process.

The remainder of this paper is structured as follows: Section II provides context information and related work. Research methodology is presented in Section III. The results of this study are presented in Section IV. A conclusion and future studies are presented in Section V. Finally, Section VI discusses the study limitations.

II. BACKGROUND

Software security is a hot subject both in academia and industry, as it has made an important contribution to this research field over the last two decades. Secure software is software that cannot be accessed, updated, or targeted by an unauthorized user. Software that has no vulnerabilities is considered highly stable, whereas software that has at least one vulnerability is considered vulnerable [20], [31].

A. SOFTWARE SECURITY BASIC CONCEPTS

This section describes some of the security terminologies [26], [31], [32] used in this paper:

- "**Software security** is the idea of engineering software that continues to function correctly under malicious attack [4], [33]."
- "**Software security** is the process of discussing an application to discover risks and vulnerabilities of the application and its data [34]."
- **Asset:** "is anything that has value to the organization, its business operations and their continuity, including information resources that support the organization's mission."
- **Vulnerability:** "A weakness in the design, operation, implementation or any process in the system which expose the system to a threat defined it as a weakness of an asset or group of assets that can be exploited by one or more attacker."
- **Threat:** "A possible danger that may result in harm to systems and organization."
- **Attack:** "An actual event done by a person; attacker to harm as an asset of the software through exploiting a vulnerability."
- **Risk:** "A potential for loss, damage, or destruction of an asset as a result of a threat exploiting a vulnerability."
- **Software Security Requirement:** "is a non-functional requirement that elicits a control, constraint, safeguard or countermeasure to avoid or remove security vulnerabilities from requirements, design or code."
- **Confidentiality:** "means to disclose information to people or programs that are authorized to have access to that information."
- **Integrity**: "assures that a system performs its intended function in an unimpaired manner, free from deliberate or inadvertent unauthorized manipulation of the system."
- **Availability:** "assures that systems work promptly, and service is not denied to authorized users."
- **Process:** "is an instance of a computer program that is being executed."
- **Secure Software Process:** "is a set of activities used to develop and deliver a secure software solution."

B. SECURE SDLC PROCESSES

Software security is threatened at different points during SDLC phases, both through inadvertent and malicious acts by insiders and outsiders with no association with the company. The most efficient technique to eliminate software bugs/vulnerabilities is to incorporate security and other non-functional standards into all phases of the SDLC. Over the years, there has been a lot of research into "high integrity," and researchers and practitioners have worked hard to construct secure software systems. Despite all of the efforts, software that offers high standards of security integrity is uncommon. Even when security is a specified, requirement and security design is given to the implementation team as input, there is no assurance that the result will be safe [35].

This section discusses the different methodologies for incorporating security into the SDLC phases, as well as the security practices that are commonly used in these methodologies:

McGraw [36], [37] recommends seven touchpoint operations ("Abuse cases, Security requirements, Architectural risk analysis, code review and repair, Penetration testing, and security operations") for creating secure software, all of which are connected to software development artifacts. Microsoft developed the Microsoft Trustworthy Computing Security Development Lifecycle [38] adds a set of security practices to each step of its software development process, as follows: during requirement phase, the security feature requirements are defined based on the customer demands, in the design phase the MS SDL suggests a set of activities to be performed such as threat modeling for security risk identification, identifying components that are critical to security or needs special attention during testing, in the implementation phase, use of static analysis code-scanning tools and code reviews, after completing implementation, the complete software is tested focusing on the security-critical components of the software during the testing phase, a final code review of new as well as legacy code is used during the verification phase, and finally, during the release phase, a Final Security Review is conducted by the Central Microsoft Security team.

TSP Secure (Team Software Process for Secure Software Development) [39] is developed specifically for software teams to help them create a high-performance team and prepare their work to produce the best results. The TSP Secure focuses directly on the security of software in three ways: planning, development and management, and training for developers about security-related aspects and other team members. In the initial phase; planning, the team identifies security risks, security requirements, secure design, code review, use of static analysis tools, unit tests, and Fuzz testing, and produces a detailed plan to be used in the development phase during a series of meetings. Next, the plan is executed, and the team ensures that all the security activities are taken place. Secure Software Development Process Model (S2D-ProM) [40] is a strategy-oriented process model that offers guidance and support to developers and software engineers at all level, from beginners to experts, to build secure software. Niazi *et al.* [2], conducted a systematic literature review (SLR) to pinpoint the required practices for developing secure software. This paper also amended Somerville's requirement engineering practices. After identifying best requirement practices, a framework for secure requirement engineering named Requirements Engineering Security Maturity Model (RESMM) was developed. Questionnaires and case studies were used to test the suggested framework. The findings demonstrate that the proposed framework is practical and adaptable.

Comprehensive, Lightweight Application Security Process (CLASP) [41] is a straightforward process that

consists of 24 high-level security activities that can be completely or partially integrated into software during the SDLC. In CLASP threat modeling and risk analysis is performed during requirement and design phase. In the design and implementation phase, it suggests secure design guidelines and secure coding standards [42]. Inspections, static code analysis, and security testing are performed in the assurance phase [43]. Correctness by Construction is a technique for developing high integrity software [44]. The following are the seven main principles of Correctness by Construction [44]: expect requirements to change, know why you are testing, eliminate errors before testing, write software that is easy to verify, develop incrementally, some aspects of software development are just plain hard, the software is not useful by itself. AEGIS (Appropriate and Effective Guidance for Information Security) first evaluating device assets and their relationships, then moves on to risk analysis, which defines weaknesses, threats, and risks [45]. According to Subedi *et al.* [46], security protection is not considered in the overall system development lifecycle due to which a lot of security breaches occur. This paper presents a secure paradigm that is an extension of security development practices in agile methodology to overcome this problem in web application development.

The Secure Software Development Model (SSDM) security training provides stakeholders in software development with adequate security education [47]. During the requirements process of SSDM, a threat model is used to identify and their capabilities. The security specification must be specialized by specifying the guidelines for achieving security. Penetration monitoring is the only SSD operation in the security assurance process that checks the software's ability to avoid the attack. Security Quality Requirements Engineering (SQUARE) methodology allows for elicitation, classification, and prioritization of security specifications for information technology systems and applications [48]. Al-Matouq *et al.* [20], conducted a Multi-vocal literature review to identify the best practices for designing secure software. Based on identified best practices, a framework Secure Software Design Maturity Model (SSDMM) was developed. The framework was evaluated using case studies, and the results show that SSDMM helps measure the maturity level of software development organizations.

- It is obvious from the above discussion that incorporating security in different phases of SDLC is inevitable for quality software. There exist various studies that discuss the importance of incorporating security in SDLC, however, still there exists space for further research in this area. To address the security risks at all stages of SDLC, there is a dire need to identify security risks and introduce secure specialized practices in SDLC. Therefore, to assess and measure security threats and vulnerabilities in SDLC phases, we have conducted a systematic literature review in this paper to identify security risks in SDLC and highlighted the security best practices that need to be incorporated in SDLC phases to make the development process more secure.

III. RESEARCH METHODOLOGY

A systematic literature review (SLR) was selected as the research method for this study. "An SLR is a type of secondary study in which primary studies are examined impartially and iteratively to define, interpret, and discuss evidence relevant to the research questions" [49]. According to Kitchenham [49], [50], an SLR has three main phases: planning, conducting and reviewing the review, as shown in Table 1. Researchers have used the SLR process in several domains [2], [51]–[55].

The authors of this work completed all three phases of the SLR. Inter-rate reliability analyses were undertaken during the initial and final selection phases of the SLR to reduce inter-person bias. The findings of the inter-rater reliability review are discussed in Section 3.2. We followed all of the processes in the three phases of the SLR, as stated in Table 1.

TABLE 1. SLR phases.

Phases	Steps
Planning	• Research Questions • Data Sources • Inclusion and Exclusion Criteria • Search Strings • Quality Criteria for Study Selection
Conducting	• Primary Study Selection • Data Extraction • Data Synthesis
Reporting	• Documenting the extracted results

A. PHASE 1: PLANNING THE REVIEW

1) RESEARCH QUESTIONS

The current study conducted an SLR to identify security threats/risks in SDLC and highlighted the security best practices that need to be incorporated in SDLC phases to make the development process more secure. The following research questions were answered in this study:

RQ1: What are the security risks that vendor firms should avoid while designing secure software applications, according to the literature?

RQ2: What are the best practices for vendor firms to follow when designing secure software applications, as identified in the literature?

2) DATA SOURCES

In this study, the data is gathered by an automated search. The automated search technique uses an optimized search string to find the most relevant literature [56]. As a result of our research experience and the recommendations of Chen *et al.* [57], a total of six digital repositories were chosen. The following are the digital sources that were chosen:

- IEEE Xplore
- ACM Digital Library
- Sciencedirect

- Springer Link
- Wiley Online Library
- Google Scholar (Search Engine)

3) SEARCH STRING

We generate an efficient search string based on the submitted study questions for retrieving relevant literature from the selected digital sources. Zhang *et al.* [56] following the criteria of the search strings were developed using the main search words used in the research questions and their alternatives. To concatenate the keywords into search strings, we used the Boolean "OR" and "AND" operators. The following string was used to scan the digital repositories: (("Risk" OR "Threat" OR "Issue" OR "Challenge" OR "Practice" OR "Solution" OR "Mitigation") AND ("Software Security" OR "Secure Software" OR "Secure Software Engineering" OR "Software Privacy" OR "Software Development Life cycle" OR "SDLC" OR "Global Software Development")).

4) INCLUSION CRITERIA

For data inclusion, we adopted the following guidelines based on parameters used by other researchers [11], [32], [55], [58]–[61]:

- Articles related to the domain of Secure Software Engineering.
- Papers were published between 2000 and 2020.
- Papers must provide at least one risk or practice relevant to software development process security specifications, design, code, testing, and maintenance security.
- Papers were peer-reviewed in conferences and journals.

TABLE 2. Study quality assessment criteria.

QA Questions	Checklist Questions
QA1	Does the study discuss any security risk/threat of software development?
QA2	Does the study address the use of any secure software development practice?
QA3	Are the aims of the research clearly stated without ambiguity in the paper?
QA4	Are the data collection methods adequately defined?
QA5	Is the research useful for the software industry and research community?
QA6	Are the limitations of the study mentioned?

5) EXCLUSION CRITERIA

For data exclusion, we followed the guidelines based on parameters used by other researchers [11], [32], [55], [58]–[61]:

- Papers that don't deal with secure software development and aren't related to the research questions.
- Papers that do not describe software security risks and practices in detail.
- Publications are not peer-reviewed and do not conform to a complete book's abstract, an editorial, or a letter.
- Paper that is not in English.
- Duplicate papers were not considered.

6) STUDY QUALITY ASSESSMENT

The final selected publications' data extraction and quality assessment (QA) were done at the same time. We established a checklist to objectively and subjectively assess the primary studies that were chosen. The checklist was generated using the guidelines given in [61] (Table 2). We have designed seven questions on the QA checklist (QA1-QA7). For each question, the following assessment was made:

- We gave an article a score of 1 if it answered the checklist question fully.
- For a partial answer, we gave it a score of 0.5.
- If it did not cover the question on the defined checklist, we gave it a score of 0.

The quality evaluation aims to see how well selected primary studies can be used to answer study research questions. As a result, Appendix contains the score assigned to each primary study. The credibility, integrity, and relevance to answering the study questions were used to evaluate the quality of the studies.

B. PHASE 2: CONDUCTION THE REVIEW

1) PRIMARY STUDY SELECTION

The tollgate method suggested by Afzal *et al.* [62] was used to refine the research articles found during primary study collection. There are five steps to this method (see Table 3):

Phase 1: Using search terms to find related articles.

Phase 2: Inclusion and exclusion of articles based on titles and abstracts.

Phase 3: Inclusion and exclusion of articles based on introduction and conclusion section.

Phase 4: Inclusion and exclusion of articles based on the full-text reading.

Phase 5: Final collection of primary studies for inclusion in the SLR based on study quality assessment criteria.

Initially, the developed search string and on the base of inclusion and exclusion criteria was used to retrieve 12114 papers from the selected online databases. The tollgate method [62] yielded a shortlist of 121 papers for consideration in the primary study. Finally, the quality assessment requirements were applied to the shortlisted papers. Appendix contains a list of the primary studies that were chosen.

2) DATA EXTRACTION AND SYNTHESIS

The primary (first) author extracted all of the data in this publication by using the inclusion and exclusion criteria as well as the study quality assessment research questions. The other authors, on the other hand, evaluated the categories and subcategories of security risks, as well as their methods, by dispersing them throughout the SDLC phases. Inter-rater reliability analyses were used to eliminate inter-person bias. Three external reviewers from the Software Engineering Research Group (SERG UOM) randomly selected fifteen papers from the first phase of the tollgate process and applied the tollgate process selection phases (phases 2–5) as well as the quality assessment criteria. We calculated

TABLE 3. Selection of articles using tollgate approach.

S. No	Electronic Databases	Phase-1	Phase-2	Phase-3	Phase-4	Phase-5
1	IEEE Xplore	1227	80	47	38	37
2	Science Direct	1726	101	31	25	22
3	Springer Link	3873	72	37	19	17
4	ACM Digital Library	3928	175	29	14	13
5	Wiley Online Library	426	64	14	8	8
6	Google Scholar (Search Engine)	934	244	53	26	24
	Total	12114	737	211	130	121

the nonparametric Kendall's coefficient of concordance to examine inter-rater agreement among the reviewers (W) [63] values. The reviewer's rate W on a scale of 0 to 1, with 0 indicating complete disagreement and 1 indicating perfect agreement. W = 0.78 for fifteen publications chosen at random, showing a high level of agreement between the authors and the external reviewers. To acquire the results against the research questions, all of the collected data was arranged by rephrasing the security threats and practices according to the study questions.

C. PHASE 3: REPORTING THE REVIEW

1) QUALITY ASSURANCE OF PRIMARY SELECTED STUDIES

The overall score for each QA question was determined and is presented in Appendix. The QA score for each primary sample was determined using the seven QA questions in Appendix. According to the data, about 88 percent of the primary studies received a score of >57 percent on the QA questions, implying that the primary studies chosen are relevant to the study research questions.

IV. RESULTS

In this section, the results of the SLR study are discussed:

A. SOFTWARE SECURITY RISKS (ANSWER RQ1)

This section aims to present security risks/threats to assist software development organizations to avoid these risks when designing secure software development. We have obtained a list of 156 software security risks using the SLR methodology. The identified security risks, along with the frequencies are discussed in the following subsections:

1) LACK OF PROPER ATTENTION TO SECURITY ISSUES DURING THE REQUIREMENT ENGINEERING (RE) PHASE

The findings of this study show that security risks in the RE phase of the SDLC are a highly rated factor, in the development of secure software. It stands on the top (97.5%) amongst all the identified security risks. Existing literature on requirement security has highlighted different security risks that might occur if security is not incorporated from the beginning. Some common security risks that might occur during the requirement phase of SDLC are listed [2], [24], [64]–[67] in Table 4.

TABLE 4. Software security risks in requirement engineering phase.

S. No	Security Risks in Requirement Engineering Phase	Freq (N=121)
i.	Security requirements are often neglected or considered as a non-functional requirement	91
ii.	Lack of security requirements negotiation and management	56
iii.	Lack of security requirements validation	49
iv.	Improper risk assessment	34
v.	Lack of security risk analysis	33
vi.	Lack of experience, knowledge, guidance, and security training during security requirement documentation	26
vii.	Lack of developing threat modeling	25
viii.	Lack of security requirements elicitation activity	21
ix.	Improper security requirement identification and inception	20
x.	Lack of secure requirement documentation	15
xi.	Lack of security requirements prioritization, management and categorization	15
xii.	Lack of security document checklist	13
xiii.	Lack of shared understanding of requirement definitions	10
xiv.	Improper plan for secure requirement authentication, authorization and privacy	9
xv.	Lack of development of corresponding security requirements artifacts	6
xvi.	Lack of security requirements awareness in customers/users	5
xvii.	Improper security requriements mapping	3
xviii.	Insecure deserialization	3
xix.	Improper identification of security requirements dependencies	2
xx.	The software development company has little ability to implement security features in small increment	2
xxi.	Improper identification of critical and vulnerable assets	2
xxii.	Change of requirement breaks	1
xxiii.	Lack of security requirements repository updating	1
xxiv.	Requirement adjustments make it impossible to connect requirement specifications to security goals	1
xxv.	Lack of security prototype	1

2) LACK OF PROPER ATTENTION TO SECURITY ISSUES DURING THE DESIGN PHASE

The stages of the SDLC where the security aspect is considered, according to our findings, can differ from study to study. Design flaws are one of the most common sources of security threats in software systems [20], [68]. It has been observed, that in most cases, software bugs are found during the design process of the SDLC [69]. The design process of the SDLC serves as the foundation for designing a secure software system [70]. Reducing risks in this step can reduce the effort needed in subsequent phases [19], [32]. As it can be observed from the findings of this study, security risks are reported more frequently in the design phase of

SDLC. Table 5, presents some of the most common security problems that occur during software design [6], [20], [68], [71]–[73]:

TABLE 5. Software security risks in design phase.

S. No	Security Risks in Design Phase	Freq (N=121)
i.	Lack of developing threat modeling during the design phase	57
ii.	Lack of attention to follow security design principles	29
iii.	Lack of security design awareness, guidance, and training	27
iv.	Improper secure design documentation	23
v.	Lack of building and maintaining abuse case models and attack patterns	23
vi.	Improper security design review and its verification	23
vii.	Lack of developing data flow diagram	20
viii.	Improper conduction of design and architecture security review	20
ix.	Improper restriction to share resource access	19
x.	Lack of security design specification review	17
xi.	Lack of establishing security design requirements	16
xii.	Lack of implementation of security design decisions: (Cryptographic protocols, standards, services, frameworks, and mechanisms)	13
xiii.	Lack of defense in depth	12
xiv.	Lack of access control and traceability	10
xv.	Lack of use of security design patterns	8
xvi.	Lack of design data encryption and validation features	7
xvii.	Improper design audit logging features	5
xviii.	Failure to handle error	5
xix.	Improper evaluation of risks from third-party components	4
xx.	The software appears to have more bugs as it becomes more complex	4
xxi.	Race conditions	3
xxii.	Usage of vulnerable components and sensitive application details	2
xxiii.	Refactoring practices breaks security constraints	1
xxiv.	Lack of diversification and obfuscation	1
xxv.	Using components with known vulnerable	1

3) LACK OF SECURE DEVELOPMENT OR CODING

The selection of appropriate coding language and classification of modules is a challenging task. Each phase of the SDLC must include a variety of appropriate security protections, analyses, and countermeasures that result in more secure code being released [74], [75]. Table 6 presents, software security issues during the coding phase of SDLC [11], [74], [76]–[78]:

Improper authentication and authorization mechanisms refer to the erroneous implementation of authentication functions and access-control policies [79]. Authentication and authorization are critical components of basic security processes, and they are particularly important in the production of secure software [80]. Microsoft uses STRIDE to model threats to their systems; threats are defined by looking into

TABLE 6. Software security risks in coding phase.

S. No	Software Security Risks in Coding Phase	Freq (N=121)
i.	Tampering: is the unauthorized modification of data	54
ii.	SQL Injection	37
iii.	Cross Site Scripting, cross-site request forgery	35
iv.	Denial of Services: is the process of making a system or application unavailable	32
v.	Repudiation: is the ability of users (legitimate or otherwise) to deny that they performed specific actions or transactions	29
vi.	Information Disclosure: is the unwanted exposure of private data	29
vii.	Elevation of privilege: occurs when a user with limited privileges assumes the identity of a privileged user	28
viii.	Spoofing: An attempt to gain access to a system by using a fake identity	26
ix.	Password Conjecture: Lack of password complexity enforcement	26
x.	Buffer and Array Overflow	25
xi.	Weak encryption, insecure communication	14
xii.	Messy code, code bad smells, dead code	13
xiii.	Code, Command Injection	10
xiv.	Format string problems	9
xv.	Session-Id Vulnerable, Session-Id Theft	6
xvi.	Hacking	5
xvii.	Man in the middle: Man-In-The-Middle Attack: This attack intercepts communications between two components.	5
xviii.	Null Pointer Dereference	4
xix.	Insecure application programming, lack of security programming language	4
xx.	Replay Attack Flaws	4
xxi.	Software security, often, fail because their development is generally based on ad-hoc foundations or follow traditional development processes	3
xxii.	Sensitive information in source code	3
xxiii.	Unsafe threading	2
xxiv.	Bandwidth Usage	2
xxv.	Failure to Restrict URL Access	2
xxvi.	Lack of difference between the developer's roles and security reviewer role to have objective results	2
xxvii.	Invalidated Redirects and Forwards	2
xxviii.	Accessible Database	2
xxix.	HTTP application instead of HTTPS	2
xxx.	Phishing through framework	2
xxxi.	TCP response timestamp	2
xxxii.	Continuous code changes make completing the assuring activities difficult	1
xxxiii.	Insecure Direct Object References	1
xxxiv.	DNS Hijacking	1
xxxv.	Send fake seismic parameters	1
xxxvi.	Continuous changing of the development processes (to support lesson learned) conflicts with audit need to uniform stable processes	1
xxxvii.	Autocomplete attribute not enabled	1
xxxviii.	POST change requests for GET	1
xxxix.	POST directives with invalidated parameters	1
xl.	Default Server Page	1
xli.	Links Injections	1

TABLE 6. *(Continued.)* **Software security risks in coding phase.**

xlii.	SSL weak certificates	1
xliii.	BPEL State Deviation and Flooding Attacks	1
xliv.	Slicing attacks	1
xlv.	Cookie poisoning	1

TABLE 7. **Software security risks in testing phase.**

S. No	Software Security Risks in Testing Phase	Freq (N=121)
i.	Lack of Penetration Analysis Security Testing	32
ii.	Lack of Static and Dynamic Analysis Security Testing	30
iii.	Lack of final security review	26
iv.	Lack of Fuzz Testing	16
v.	Brute Force Attack	7
vi.	Lack of developing threat models: as it helps to develop test cases or test plans	7
vii.	Lack of Functional and Non Functional Testing	6
viii.	Lack of manually reviewing the code	6
ix.	Lack of Automatic Patch Generation	5
x.	Lack of Unit Testing	5
xi.	Various kinds of Attacks (viruses,) malware, Trojan Virus: A type of virus that is well known for causing issues and destruction to computers is a Trojan virus.	5
xii.	Lack of developing test plan to describe software testing scope and activities	3
xiii.	Illegal seismic parameters, incomplete/inconsistent parameter validation	3
xiv.	Invalid correct use of Security Testing Tools	3
xv.	Tests are, in general, insufficient to ensure the implementation of security requirements	1
xvi.	Tests do not cover in general, all vulnerabilities cases	1
xvii.	Security tests are in general difficult to automate	1
xviii.	Lack of Redundancy Analysis	1

the possibilities of spoofing identity, tampering with data, repudiation, information leakage, denial of services, and elevation in the given situation [71]. The present study identified 63 articles to discuss authentication and authorization are essential parts of security in the development of secure software. Spoofing, tampering, repudiation, information disclosure, denial of services, elevation of privilege and failure to restrict URL access are some of the most common security issues that hamper the process of secure authorization and authentication [31], [64], [71], [72], [81].

Incorrect input validation refers to the lack of or incorrect validation of input provided by a user via the application's user interface. Injection attacks take advantage of the lack of input validation controls to allow malicious inputs to be passed in, which can be used to obtain elevated rights, alter data, or crash a system [82]. Code injection attacks can breach data security, cause a loss of services, and harm thousands of users' systems [83]. This study identified; Cross-site scripting, Cross-site request forgery, format string problems, code and command injection, autocomplete attribute not enabled,

TABLE 8. **Software security risks in deployment phase.**

S. No	Software Security Risks in Deployement Phase	Freq (N=121)
i.	Lack of default software configuration	9
ii.	Logout incorrectly implemented	5
iii.	Improperly enabled services and ports	3
iv.	Ignoring security breaks	3
v.	Lack of output validation	3
vi.	Improper use of secure APIs	2
vii.	Lack of security feed back	2
viii.	Lack of response in planning and execution	2
ix.	Concurrent connections from different IPS	1
x.	SMB signature (Server Message Block) Not required	1
xi.	Lack of certification in final release and archive	1
xii.	Lack of ensures to specify customer expectations and requirements	1
xiii.	Improper code sign-off	1
xiv.	Improper data sanitization and safe disposal	1
xv.	Lack of threat models updating	1

TABLE 9. **Software security risks in maintenance phase.**

S. No	Software Security Risks in Maintenance Phase	Freq (N=121)
i.	Lack of security trust	18
ii.	Lack of proper methods to find out new threats in the system	15
iii.	Lack of finding the attack surface area for the new threats	13
iv.	Not developing security patches for the threats	12
v.	Improper configuration, vulnerability management and change control	8
vi.	Security activities increase the cost of the software	7
vii.	Timing attacks	5
viii.	Inability to run software updates or change usernames and passwords	5
ix.	Lack of Log Optimization	4
x.	Lack of educate the users in using the software application in a secure manner	4
xi.	Need to verify vulnerability correction	4
xii.	Lack of Maintenance of Software Security	2
xiii.	Lack of relationship management in bug reporting	1
xiv.	Lack of continuous monitoring and improvement of security assessment	1
xv.	Lack of collaboration with external organizations in promoting system security	1
xvi.	Lack of piloting innovative ideas, technologies and security tools to improve organizational security	1
xvii.	Lack of government assistance for proper rules for cybercrime	1

POST change requests for GET, POST directives with invalidated parameters, and accessible database are injection vulnerabilities from the literature [5], [11], [79], [83], [84].

The vulnerabilities in software systems include outdated software/firmware, default usernames and password, password conjuncture, and the inability to run software updates or change usernames and passwords, are leveraged to gain initial access to systems of corporate targets which then can be further exploited [6], [85], [86].

TABLE 10. Secure requirement engineering practices (SREP).

SREP1	**Develop Threat Modeling (Freq: 25)**
SREP1.1	Identify threat origin with the help of threat modeling in the requirement phase
SREP1.2	Follow STRIDE Threat Model
SREP1.3	Follow DREAD Threat Model
SREP1.4	Analyze the threats faced at the time of requirement development
SREP2	**Security Requirement Elicitation Practices (Freq: 31)**
SREP2.1	Elicit and categorize safety and security requirements
SREP2.2	Take into consideration organizational and political issues
SREP2.3	Use scenarios to elicit sensitive data and communication in terms of authentication, authorization, privacy, system maintenance security requirement
SREP2.4	Identify stakeholders
SREP2.5	Identify the operating environment of the system
SREP2.6	Use concerns related to business to motivate security requirement elicitation
SREP2.7	Identify information assets
SREP2.8	Identify functional and non-functional security requirements
SREP2.9	Search for domain constraints
SREP2.10	Record rationale for security requirement
SREP2.11	Gather security requirements from different and various views
SREP2.12	Use hypothetical cases to elicit security requirements
SREP2.13	Identify operational process
SREP2.14	Remove any ambiguous requirements
SREP2.15	Reuse security requirements
SREP2.16	Determine and consult stakeholders of the system
SREP2.17	Record security requirements sources
SREP2.18	Assess system security feasibility
SREP3	**Perform Secure Requirement Identification and Inception (Freq: 20)**
SREP3.1	All stakeholders, customers, clients need to be agreed on the requirement definition
SREP3.2	Illustrate the security needs with different perspectives, analyze them, priorities and then specify
SREP3.3	Identification of security goals
SREP3.4	Identify high-level functional security objectives, requirements
SREP3.5	Identification of potential attackers of the software
SREP3.6	Utilize brainstorming technique to aggregate identification security requirement
SREP3.7	Identify system stakeholders to improve identification security requirement
SREP3.8	Check that identification security requirement meets your standard
SREP3.9	Set forth the security objectives to address the needs identified
SREP3.10	For each security objective, security requirements are identified along with the functional and non-functional requirements
SREP3.11	Capture and define non-functional security requirements as attributes of the software
SREP4	**Perform Security Requirement Analysis and Negotiation (Freq: 28)**
SREP4.1	The main responsibility is to conduct product security risk analysis to ensure early identification of potential security requirements and constraints.
SREP4.2	Attack trees modeling is one of the techniques suggested to be used for analyzing security risks
SREP4.3	Analyze tradeoffs between cost and protection provided by security controls
SREP4.4	Security Risk Assessment: use DREAD model
SREP4.5	Identify security issues with STRIDE by classifying attacker goals
SREP4.6	Perform threat landscaping
SREP4.7	Data comprehensiveness
SREP4.8	Grouping of Requirements
SREP4.9	Write down the misuse cases for each secure requirement identified
SREP4.10	Define security of system boundaries
SREP4.11	Verify the misuse case strength in understanding possible attacks
SREP4.12	Make use of checklists to analyze security requirements
SREP4.13	Conflicts Resolution: Consider conflicts and how to resolve them
SREP4.14	Sort out security requirements through a multi-dimensional approach
SREP4.15	Identify priorities in security requirements
SREP4.16	Provide software to support negotiations
SREP4.17	Perform risk analysis to address the security issues in requirement development
SREP5	**Perform Secure Requirement Mapping (Freq: 3)**
SREP5.1	Map all the non-functional security requirements identified with functional requirement
SREP5.2	Make the mapping explicit, identify the use cases adapted to misuse cases
SREP5.3	Translate all the negative and non-actionable requirements to positive and actionable requirements
SREP6	**Security Requirement Documentation Practices (Freq: 15)**
SREP6.1	Incorporate security needs, objectives, and requirements in the final documentation
SREP6.2	Specify security policies, standards, and reference guidelines for security requirements
SREP6.3	Explain how to use the security document
SREP6.4	Make a business case for the system concerning security
SREP6.5	Define specialized security terms
SREP6.6	Help readers find information
SREP6.7	Make the document easy to change
SREP6.8	Include a summary of the security requirement
SREP6.9	Illustrate threat landscaping, risk likelihood, and mitigation strategy

TABLE 10. *(Continued.)* **Secure requirement engineering practices (SREP).**

SREP7	**Perform Secure Requirement Review, Verification & Validation (Freq: 25)**
SREP7.1	Review documentation against the objectives and needs
SREP7.2	Check the documentation against the security requirement documentation acceptance test parameters
SREP7.3	Perform Secure Requirements Review
SREP7.4	Software products should be certified according to security requirements
SREP7.5	Validate that software artifacts and processes no longer bear the unacceptable risk
SREP7.6	Identification of attackers interest and capabilities in the resources/assets of a piece of software
SREP7.7	A threshold of acceptable security can be defined by using, security index.
SREP7.8	Identify validation checklists
SREP7.9	Specify low-level security requirements to remove security errors
SREP7.10	Use multi-disciplinary teams to assess security requirements
SREP7.11	Use prototype to animate security requirements
SREP8	**Perform Secure Requirement Prioritization and Management (Freq: 21)**
SREP8.1	Perform Requirement Specification
SREP8.2	Identify policies for management of security requirements
SREP8.3	Specifically, define each security requirement
SREP8.4	Risk mitigation should be conducted in a coherent and a cost-effective manner
SREP8.5	Governance: Practice that helps organize, manage, and measure a software security initiative
SREP8.6	Evaluate and manage product security risks throughout the project
SREP8.7	Risk ranking to prioritize and determine the risks that should be avoided
SREP8.8	Preservation of confidentiality, Integrity, Availability, Usability, should be specified to mitigate identified threats
SREP8.9	Establish and manage the project secure development process
SREP8.10	Define and maintain traceability manual
SREP8.11	Identify view point
SREP8.12	Define policies for change management
SREP8.13	Identify global system security requirement
SREP8.14	Asset Rating
SREP8.15	Risk estimation
SREP8.16	Identify volatile security requirement
SREP8.17	Record rejected security requirement
SREP8.18	Vulnerability measurement
SREP8.19	Perform Requirement Elaboration
SREP8.20	Threat evaluation & prioritization
SREP9	**Plan for Secure Requirement Authentication, Authorization, and Privacy (Freq: 9)**
SREP9.1	Plan for conflicts and conflict resolution for authentication, authorization, immunity security, non-repudiation, and system maintenance requirement in terms of multiple accounts
SREP9.2	Define standard templates for describing authentication, authorization, immunity, privacy, integrity, non-repudiation, intrusion detection, and system maintenance security requirement
SREP9.3	Use simple and concise language to explain authentication, authorization, immunity, privacy, integrity, non-repudiation, intrusion detection, and system maintenance security requirement
SREP9.4	Check that authentication, authorization, immunity, privacy, integrity, non-repudiation, intrusion detection, and system maintenance security requirement meets your standard
SREP9.5	Define change management policies for authentication, authorization, immunity, privacy, integrity, non-repudiation security, and system maintenance requirement
SREP9.6	Use interaction matrices to find conflicts and overlaps in terms of intrusion detection security requirement
SREP9.7	Define the system boundaries in terms of privacy and system maintenance security requirements such as sensitive data and communication.
SREP9.8	Define operational processes to gain non-repudiation, integrity, immunity, intrusion detection, and security requirement
SREP10	**Assess Physical Protection, Survivability and Secure Auditing Requirement Risks (Freq: 8)**
SREP10.1	Assess physical protection, survivability, and secure auditing requirement risks
SREP10.2	Be sensitive to organizational and political considerations in gaining physical protection of security requirement
SREP10.3	Use checklists for secure auditing requirements
SREP10.4	Define the system's operation environment to gain survivability security requirement
SREP10.5	Institute accountability for security issues
SREP10.6	Assess system feasibility in terms of survivability security requirement
SREP11	**Methods used in security RE (Freq: 42)**
SREP11.1	UMLsec, SecureUML
SREP11.2	Secure Tropos
SREP11.3	Abuse Cases
SREP11.4	Structure Object-Oriented Formal Languages
SREP11.5	Machine learning Techniques
SREP11.6	Fuzz-Analytic Hierarchy Process
SREP11.7	Security Requirement Engineering Approach
SREP11.8	Problem Frames
SREP11.9	Tropos (i* framework)
SREP11.10	Create and describe Misuse Cases
SREP12	**Others (Freq: 52)**
SREP12.1	Assess Security and Privacy Risk
SREP12.2	Institute Security awareness program
SREP12.3	All security team members have adequate security training
SREP12.4	Establish an organization policy for security
SREP12.5	Proactive approaches or top-down approaches are also qualified as preventive, as they deal with security concerns since the

TABLE 10. *(Continued.)* **Secure requirement engineering practices (SREP).**

	requirement phase will drive the next development steps.
SREP12.6	It may be possible that more than one security mechanism can fulfill a security requirement.
SREP12.7	Update Requirement Repository
SREP12.8	Adopt international standards that fit your organization
SREP12.9	Identify security packages
SREP12.10	Identify user roles and resource capabilities
SREP12.11	Assess immunity security requirements in terms of undesirable programs
SREP12.12	Analyze and minimize the attack surface
SREP12.13	Identify requirement dependencies
SREP12.14	Develop correspond artifacts analysis which examines the possible risk
SREP12.15	Create quality gates/Bug bars
SREP12.16	Develop security guidelines (collection of practices, checklists, code style, security specification, security function, etc)

The majority of security attacks are possible due to implementation flaws such as improper input validation, improper authentication, and authorization mechanisms, improper session management, and other vulnerabilities (Session-Id vulnerable or theft, logout incorrectly implemented, lock failed attempts per browser session, peer-user session restriction, and log replay feature) that compromise the application's intended functionality [5], [79], [87].

In MITRE's Common Vulnerabilities Exposures database, the latest classification of common defects by type is provided in Common Vulnerability Enumeration [88], a list of registered vulnerabilities. As a consequence, the most common forms of security vulnerabilities are weak encryption, explicit password storage, insecure communication, and synchronization errors [88]. Invalidated redirects and forwards, improper use of secure APIs, weak encryption, insecure communication, man in the middle, and bandwidth usage are some of the most common security issues that hamper the communication and encryption processes [85], [88], [89].

Software security is concerned with protecting data, facilities, and applications from harm caused by various types of malware attacks (e.g., password sniffing, viruses, hijacking) that may be mounted by various types of attackers (e.g., hackers, crackers, domestic cyber-terrorists, industrial spies, international military, and so on) [87], [89]. This study identified some of the most common malware attacks (various kinds of viruses, malware, trojan virus, brute force attack, DNS hijacking, replay attack flaws, attacker denies services to the application by opening thousands of connections but does nothing with them, BPEL state deviation and flooding attacks, send fake seismic parameters, the bulleting is modified before and during sending, the bulletin is not delivered or delivered to the fake place, blocking of E-mail notification by a malicious user, the attacker shuts down the user's process, slicing attacks, and cookie poisoning) which affect the processes of secure software development [87], [89]–[91].

4) LACK OF PROPER ATTENTION TO SECURITY TESTING ANALYSIS

The testing phase of the SDLC aims to make sure that all the system components provide their required functionality alone and as part of the whole system. Software testing is the most time-consuming, complicated, and costly process of the SDLC [92]. This phase is an important component of improving the efficiency of software development projects [32]. While it is an essential part of software development, rigorous testing is not always a focus of software engineering education [93]. As a result of this shortcoming, software developers often regard software testing as a liability, lowering overall software quality. Threat modeling is a systematic method for identifying threats that may compromise security, and it is considered a well-known accepted practice by the software testing industry [94]. This phase aims to find possible bugs and errors in the system and remove them. The present study identified 64 papers to discuss software security risks during software testing phase of SDLC. Some common security risks involved in this phase are as follows [5], [22], [95]–[98]:

5) SOFTWARE SECURITY RISKS IN DEPLOYMENT PHASE

Developing secure software systems involves many challenging problems, e.g., designing authentication protocols, improper configuration management, building strong cryptosystems, devising effective trust models and security policies [99]. Configuration management is an important component in the secure maintenance and operation phase [100]. This study identified (see Table 8), some of the common software security risks which affect deployment phase of the SDLC in the development of secure software [5], [78], [99]–[102].

6) SOFTWARE SECURITY RISKS IN THE MAINTENANCE PHASE

Vulnerability-oriented architectural research provides a systematic and thorough approach to evaluating a wide variety of possible vulnerabilities, but it is time-consuming and costly [91]. For estimating the severity and cost of security threats, Table 9 presents, some maintenance and stakeholder considerations may be considered [78], [91], [103].

Software development iterations are of limited time, often few weeks, which makes fitting security activities (e.g., security requirement elicitation) challenging because they are often time-consuming'' [65]. Furthermore, defining security policies takes time and raises the cost of software development [65]. Some of the common issues due to

TABLE 11. Secure design practices (SDP).

SDP1	**Develop Threat Modeling**	**(Freq:57)**
SDP1.1	Enumerate threats and prioritize the threat based on the potential impact	
SDP1.2	Analyze and Minimize Attack Surface	
SDP1.3	Verify whether the threat is mitigated with a security control	
SDP1.4	Identify areas that could be of interest to attackers	
SDP1.5	There could be multiple design decisions to mitigate any threat	
SDP1.6	Secure design decisions to remove threats can be prioritized based on a cost/benefit analysis	
SDP1.7	Secure design decisions must be identified for threats that violate any of the high-level security requirements.	
SDP1.8	Risk analysis should be performed on the identified threats to calculate the potential damage	
SDP1.9	Use threat weighting or ranking during threat modeling	
SDP2	**Secure Design Documentations**	**(Freq:23)**
SDP2.1	Develop Test plan	
SDP2.2	Document each identified threat along with its description, risk, defensive technique, and risk management strategy	
SDP2.3	Document secure design	
SDP2.4	Remove unimportant features	
SDP2.5	Identify design attributes	
SDP2.6	Use security diagram classes	
SDP2.7	Remember that hiding secrets are hard	
SDP2.8	Avoiding logs from external data	
SDP2.9	Map security requirements with cryptographic services (Authentication, Confidentiality, Integrity, and Non-Repudiation)	
SDP2.10	Identify environmental and device security constraints	
SDP2.11	A threshold of acceptable security can be defined by using, security index.	
SDP2.12	Perform cost/benefit analysis (CBA) & Security planning (based on risks & CBA)	
SDP3	**Follow Security Design Principles for Secure Software Development**	**(Freq:29)**
SDP3.1	Least Privilege	
SDP3.2	Implement defense-in-depth policy which includes multilevel security	
SDP3.3	Keep your design as simple as you can by applying economy of mechanism policy	
SDP3.4	Correctness by Construction (CbyC)	
SDP3.5	Fail Securely: The system does not disclose any data that should not be disclosed ordinarily at system failure	
SDP3.6	Apply false-safe default principles to make sure that the failure of any activity will prevent unsafe operation	
SDP3.7	Separation of Privilege	
SDP3.8	Reluctance to Trust	
SDP3.9	Use a Positive Security	
SDP3.10	Establish Secure Defaults	
SDP3.11	Never assumes that your secrets are safe	
SDP3.12	Securing the Weakest Link	
SDP3.13	Proactive not Reactive	
SDP3.14	Privacy as the Default	
SDP3.15	Privacy Embedded into Design	
SDP3.16	Full Functionality	
SDP3.17	End-to-End Security	
SDP3.18	Visibility, Usability and Transparency	
SDP3.19	Detect Intrusion	
SDP3.20	Implement Sandboxing	
SDP3.21	Follow psychological acceptability principle of design to automatically incorporate basic security	
SDP3.22	Follow the least common mechanism to restrict shared resource access	
SDP3.23	Respect for User Privacy	
SDP4	**Secure Design Review and Verification**	**(Freq:23)**
SDP4.1	Revise or review design implementation	
SDP4.2	External review of the design	
SDP4.3	Establish secure design requirements	
SDP4.4	Plan and implement secure supplier and third-party component selection	
SDP4.5	The design must be inspected (multiple times if required) for identifying and removing software errors	
SDP4.6	Remember that backward compatibility will always give your grief	
SDP4.7	The expert also needs to verify the interface and mediator between product management and development	
SDP4.8	Create a team to identify new attacks	
SDP4.9	Identify potential attackers and develop attacker profiles	
SDP4.10	Perform comparative security assessments of different integrating options	
SDP4.11	Use automation to stimulate attacks	
SDP4.12	Establish a process for architectural analysis	
SDP5	**Others**	**(Freq:24)**
SDP5.1	Implement security design decisions: (Security Cryptographic protocols, standards, services, and mechanisms)	
SDP5.2	Use of security patterns	
SDP5.3	Apply access control mechanism to make sure for authorization	
SDP5.4	Do not mix code and data	
SDP5.5	Secure data transition by invoking secure data transfer protocols	

TABLE 11. *(Continued.)* **Secure design practices (SDP).**

SDP5.6	Interface with third parties who feed the software supply chain
SDP5.7	Identify time-consuming factors in the secure development process,
SDP5.8	Learn from mistake
SDP5.9	Security design awareness and knowledge training

time pressure in the secure software development process are [65], [83], [101]:

i. Organizations compromise security activities to accommodate the accelerated releasing schedule
ii. Timing attacks
iii. Insufficient time for the teams to get used to the security activities
iv. The pressure to deliver to tight deadlines.

B. PRACTICES FOR DEVELOPING SECURE SOFTWARE (ANSWER RQ2)

In all phases of the Software Development Life Cycle (SDLC), the focus on secure software development has gradually grown over the last two decades. To produce secure software, security awareness, guidelines, principles, and practices are very important during all the stages of SDLC. The purpose of this section is to describe software security practices to help software development firms better specify the criteria for secure software development. To answer RQ2, we must go through the following subsections:

1) BEST PRACTICES FOR SECURE REQUIREMENT ENGINEERING (SRE)

The requirement stage in the SDLC is the primary stage where the initial plan for software is made. It necessitates a set of initial specifications, which are collected from a variety of sources. Various methods such as brainstorming, group sessions, and interviews are used to gather requirements. Secure requirement engineering (SRE) is different; the aim is to provide complete security by implementing basic security functions, such as confidentiality, integrity, and availability [25]. SRE is usually done during the first stage of the SDLC, and the success of this phase leads towards a better software product. Further, handling security in this phase assists software development organizations to save rework and additional costs. SRE has proved to be a difficult task over time. The main activities involved in this stage are security requirements identification and inception, documentation, elicitation, analysis and negotiation, mapping, verification and validation, prioritization and management, authentication, and authorization [2], [64], [104]. Various researchers and industry practitioners have emphasized the importance of considering SRE from the start of the secure software development process. We list down (See Table 10) the commonly used best practices for handling security issues during the requirement stage of SDLC [2], [22], [24], [59], [64], [67], [90], [104]–[106].

2) BEST PRACTICES FOR DESIGNING SECURE SOFTWARE

The design phase is one of the most creative stages of the SDLC, which is one of the reasons it is important from the viewpoint of security [32], [69]. 50%, software defects are identified and detected during the design stage of the SDLC [32], [69]. The security design architecture specifies design methods such a strongly typed programming, least privilege, develop threat modeling, analyze and minimize attack surface [14]. The software developer must consider security best practices during design to complete this phase in a manner that is appropriate and secure. Table 11 presents some of the most widely used design security practices, these should be followed when designing secure software [14], [22], [31], [32], [68], [69], [71]–[73], [105], [107].

3) BEST PRACTICES FOR IMPLEMENTING SECURE CODE

80 percent of system penetration is due to coding errors in commercial software. This is surely a matter of national security. Increased bugs, security issues, and costs are all associated with bad code. Good code pays off in the long run [14]. Due to time-to-market pressures, software developers are passed to meet the deadline, lack security expertise, and fail to follow secure code guidelines. Furthermore, they make the mistake of assuming that perimeter security is sufficient to protect applications. Security code reviews, which can be conducted while the code is being checked for functionality, whether manual or automated, are required to verify the fundamental tenets of software security [22], [108]. The programmer must be aware of implementation-level vulnerabilities when writing secure code [14]. Programmers can use the documentation and guidelines created in earlier stages to help them write secure code. Table 12 shows prescriptive actions to increase security during the coding phase of SDLC [5], [14], [22], [98], [105], [109]–[111].

4) BEST PRACTICES FOR SECURE SOFTWARE TESTING

Software testing is the most time-consuming, complex, and costly phase of the SDLC [92]. This phase aims to identify and fix any bugs or errors in the system. ‘‘To detect potential attacks and the consequences of successful attacks, security testers typically use misuse cases, threat models, and design documents’’ [14]. Following the completion of security testing, test documents containing security test cases and a prioritized list of vulnerabilities resulting from automated and manual dynamic analysis are created [14]. Table 13 shows

TABLE 12. Secure coding practices (SCP).

S. No	Security Practices in Coding/Implementation Phase	Freq
SCP1	Perform Code Review	28
SCP2	Provide Security Knowledge and Training to Software Developers	24
SCP3	Implement static code analysis	18
SCP4	Apply secure coding standards such as CERT, MISRA, and AUTOSAR	15
SCP5	Use approved Security tools for Implementation	11
SCP6	Conduct source code assessment process	9
SCP7	Expert need to ensure that secure coding practices are followed and conducts code analysis to identify security vulnerabilities	6
SCP8	Validate input and output to mitigate common vulnerabilities	5
SCP9	Deprecate unsafe functions	5
SCP10	Develop complex Encryption methods	5
SCP11	Implement dynamic code analysis	5
SCP12	Using secure programming language that is safe to increase security in the development	4
SCP13	Refactoring can improve the security of an application by removing code bad smell	3
SCP14	Code-level hardening is a way that prevents vulnerabilities	3
SCP15	Secure code writing	3
SCP16	Secure the weakest link	3
SCP17	Perform security certification and accreditation of target system	3
SCP18	Eliminate weak cryptography	3
SCP19	Develop proper error/exception handling along with respective error message	3
SCP20	Develop threat modeling: It helps to do threat analysis into secure code review	2
SCP21	All temporary files of the cookies folder should be deleted	2
SCP22	All legitimate users must have the privileges and minimum access needed	2
SCP23	Avoid race conditions	2
SCP24	Code must be inspected to identify software and security errors	2
SCP25	Implement Diversification and Obfuscation	2
SCP26	Use of established security algorithms	2
SCP27	Choose a proper and hard to guess location for temporary files and applying an access control mechanism	2
SCP28	Define the acceptance level of vulnerabilities within the coding	2
SCP29	Review of complex functions	1
SCP30	Provide data protection services	1
SCP31	Handle data and errors safely	1
SCP32	Find security issues early	1
SCP33	Identify and Access Management	1
SCP34	Integrate security analysis into the source management process	1
SCP35	Minimize use of unsafe string and buffer functions	1
SCP36	Use robust integer operations for dynamic memory allocations and arrays offsets	1
SCP37	Maintain legacy code	1
SCP38	Heed compiler warnings	1
SCP39	Mask the problem, by applying filters to either block or modify user input	1
SCP40	Determine and execute remediation strategies	1
SCP41	Use Logging and Tracing	1
SCP42	Avoid weak or ambiguous variables	1
SCP43	Prevent execution of illegitimates code	1
SCP44	Remove debugging code and flags in code	1
SCP45	Do not make any rigid boundaries in the development; make sure to keep it flexible to be able to face any security challenges in the present and future	1
SCP46	Perform sanity check before invoking any pointer	1
SCP47	**Practices for Spoofing**	5
SCP47.1	Restriction of access	
SCP47.2	Customers must use a strong password or use a multi-login mechanism	
SCP47.3	Declaration of accessible IP addresses by using .htaccess file	
SCP47.4	Do not store secrets in plain text	
SCP47.5	Protect secret data	
SCP47.6	Acquisition of log	
SCP47.7	Recording of user ID, Date, type of operation, name of AP at time of operation execution	
SCP48	**Security Practices for Tampering**	7
SCP48.1	Acquisition of log	
SCP48.2	Appropriate authorization	
SCP48.3	Apply Hashes, message authentication codes	
SCP48.4	Incorporate Digital Signatures	
SCP48.5	Communication connections between system components must be ensured using protocols that provide confidentially	
SCP48.6	Recording of user ID, Date, type of operation, name of AP at time of operation execution	
SCP49	**Security Practices Non Repudiation**	5
SCP49.1	Every activity related to important and sensitive data must be recorded	
SCP49.2	Incorporate Digital Signatures, timestamps, audit trails	

TABLE 12. *(Continued.)* **Secure coding practices (SCP).**

SCP49.3	Ensure that the sender of a message does not deny having sent the message and the receiver does not deny receiving the message	
SCP50	**Security Practices for Information Disclosure**	2
SCP50.1	Ensure that only selected accounts can access important data	
SCP50.2	Implement encryption mechanism and protect secrets	
SCP51	**Security Practices for Denial of Services**	4
SCP51.1	Restrict the number of accesses per hour	
SCP51.2	Appropriate authentication and authorization	
SCP51.3	Appropriate filtering, throttling, quality of service	
SCP52	**Security Practices for Man-in-the-middle**	4
SCP52.1	Encryption of communication using cryptography (Mozilla guideline-secure transmission)	
SCP52.2	Acquisition of public key signed by Certificate Authority	
SCP52.3	Intrusion detection system	
SCP52.4	Exchange of public keys using a secure channel	
SCP53	**Security Practices for Illegal OR Unauthorized Access**	27
SCP53.1	Physical security techniques, such as lock doors, alarms, and monitoring of targets, should be implemented	
SCP53.2	Hashing of passwords (OWASP password storage cheat sheet)	
SCP53.3	Confidentiality: Protect data or services from unauthorized access	
SCP53.4	Integrity: Avoid unauthorized manipulation of data or services	
SCP53.5	Availability: Assure that the system work promptly, and services are not denied to an authorized user	
SCP53.6	Authentication: Identify the actors involved in a transaction and verify that they are who claims to be	
SCP53.7	Use a firewall, VPN, and SSL techniques	
SCP53.8	Hashing of password using "Auth" component in CakePHP	
SCP54	**Security Practices for Password Conjecture**	8
SCP54.1	Delete all default account credentials that may be put in by-product vendor	
SCP54.2	Strengthen the password	
SCP55	**Security Practices for SQL Injection**	5
SCP55.1	Use of parameterized queries or stored procedures (OWASP SQL injection prevention cheat sheet)	
SCP55.2	Input sanitization: User inputs are sanitized to ensure that they contain no dangerous code.	
SCP55.3	Security privileges. Setting security privileges on the database to the least required. For example, the delete rights to a database for end-users are seldom required.	
SCP55.4	Disabling literals. SQL injection can be avoided if the database engine supports a feature called disabling literals, where text and number literals are not allowed as part of SQL statements.	
SCP55.5	Avoid string concatenation for dynamic SQL statements	
SCP55.6	Check the query if they exist in query pool only they are permitted	
SCP55.7	You can replace the single quote with double-quotes. This blocks the SQL insertion attack	
SCP55.8	Use of Prepared Statement and/or "ORM"	
SCP56	**Security Practices for Brute for Attack**	8
SCP56.1	Implementation of password throttling mechanism (OWASP Authentication Cheat Sheet-prevent brute force attack)	
SCP56.2	Implement Account lockout procedure	
SCP56.3	Implement count number of login trial and set a flag that shows account lock to "True" if the number exceeds the threshold	
SCP56.4	Establishment of strong password policy and check of its observance (OWASP Authentication Cheat Sheet-password complexity, use multi-factor authentication)	
SCP56.5	Enhancement of specification regarding password	
SCP56.6	Checking of input with validation function in CakePHP	
SCP56.7	Use of strong input validation (OWASP SQL injection prevention cheat sheet)	
SCP56.8	The user of the error handler in CakePHP (core.php)	
SCP56.9	Recycle of password	
SCP57	**Security Practices for Cross Cite Scripting, and CSRF**	8
SCP57.1	Design libraries and templates that minimized unfiltered input. (OWASP XSS Prevention Cheat Sheet)	
SCP57.2	DNS rebinding	
SCP57.3	Using ESAPI safety mechanism can eliminate detected XSS vulnerabilities in web application	
SCP57.4	HTMLPurifier eliminates XSS vulnerabilities	
SCP57.5	Check input with validation function in CakePHP	
SCP57.6	Use a cryptographic token to associate the request with specific action. The token can be regenerated at every request. (OWASP CSRF Prevention Cheat Sheet; encrypted token)	
SCP57.7	Use of optional HTTP Referrer header	
SCP57.8	Confirm action every time concerning potentially sensitive data is invoked	
SCP57.9	Normalize filtering of all inputs including those not expected to have any scripting content.	
SCP57.10	"Sanitizing" is one way to prevent XSS attacks	
SCP57.11	Use of h() function and security component in CakePHP	
SCP58	**Security Practices for Session ID Hijacking**	4
SCP58.1	Always invalidate session ID after user logout	
SCP58.2	Use of destroy method in CakePHP session component	
SCP58.3	Setup of session time out for session IDs	
SCP58.4	Protect communication between client and server	
SCP58.5	Encrypt session data associated with session ID	

TABLE 12. *(Continued.)* **Secure coding practices (SCP).**

SCP58.6	Use multifactor authentication (OWASP authentication cheat sheet- use multi-factor authentication)	
SCP58.7	Use of session_id() function in CakePHP for creation of session ID	
SCP58.8	Regenerate and destruct session identifiers when there is a change in the level of privilege	
SCP58.9	After some time server should forcefully terminate the user's session	
SCP58.10	Use Secure Socket Layer (SSL) of client and IP address	
SCP58.11	Use strict session management mechanism that only accepts locally-generated session identifier	
SCP58.12	Use of safe session management mechanism	
SCP58.13	Pass session ID via hidden tag	
SCP58.14	Use of session identifiers that are difficult to guess or conduct brute force	
SCP58.15	Do not code to send session ID with GET method	
SCP59	**Security Practices for Avoiding Buffer Overflow and Format String Vulnerabilities**	6
SCP59.1	Choosing a type-safe programming language can avoid many buffer overflow problems.	
SCP59.2	Correct use of safe libraries of an unsafe language can prevent most buffer overflow vulnerabilities.	
SCP59.3	Stack-smashing protection can detect the most common buffer overflow by checking that the stack has not been altered when a function returns.	
SCP59.4	Executable space protection can prevent the execution of code on the stack or the heap.	
SCP59.5	Normalize strings before validating them	
SCP59.6	Deep packet inspection attempts to block packets that have the signature of a known attack or have a long series of no-operation instruction	

prescriptive actions to increase security during the testing phase of SDLC [5], [22], [25], [95]–[98], [105], [112], [113].

5) BEST PRACTICES FOR DEPLOYING SECURE SOFTWARE

After the software is deployed into its operational environment, it is important to monitor responses to flaws and vulnerabilities of the system to check for new evolved security patterns [66], [91]. After identifying new security patterns, the same should bed included in the requirement stage for further security improvements in subsequent releases [66], [91]. Static analysis and peer review are two useful procedures for mitigating or minimizing newly discovered vulnerabilities [14]. Final security reviews and audits are performed during the secure deployment phase [14]. At this phase, customer satisfaction is also very important. Table 14 presents prescriptive actions to increase security during the deployment phase of SDLC [5], [14], [98], [105], [114], [115].

6) BEST PRACTICES FOR MAINTAINING SECURE SOFTWARE

Before deploying software, administrators must first understand the software's security stance. Some of the identified faults that were not addressed previously will be revisited, prioritized, and corrected after deployment. New threats are tracked during this phase. The software can never be 100 percent secure, and new threats emerge regularly phase [14]. As a result, efforts must be made to secure the software. The maintenance team should keep track of new threats that the system encounters to address them promptly and prevent security breaches [83], [116]. Table 15 presents prescriptive actions to increase security during the maintenance phase of SDLC [14], [65], [105], [114], [117], [118].

V. CONCLUSION AND FUTURE WORK

The above discussion has highlighted the brief details of SDLC phases along with the security issues and their mitigation practices. Software security is now a primary need for secure software development (SSD) at every phase of the SDLC. We conclude from the preceding discussion that securing software systems in the post-development phases is insufficient, and better ways and means of securing software systems are urgently required. To summarize, software security is an important feature that should be given top priority. Many software projects have failed in the past due to a lack of attention on the security factor. Testing software for security after it has been developed is not only time-consuming and difficult, but it also adds to the project's complexity and more cost. Secure software engineering (SSE) believes that software security is a critical factor that should be assessed early in the SDLC process [119]. To build and deploy a secure software system, we need to integrate security features into our application development life cycle and adapt the latest SSE practices [3], [4].

Backward compatibility will be harmed if security is added after deployment since it will change functionality and/or application interfaces. Because adding security takes more time and money than doing it from the beginning, it is less likely to be done effectively or with care. In view of the necessity of incorporating security into the development lifecycle, the authors of this study endeavored to establish a methodology that addresses security across the SDLC.

The purpose of this article is to identify security issues and to give a set of development techniques, guidelines, activities, principles, and rules to help developers create more secure software. In the light of the importance of software security, we conducted a thorough systematic literature review and identified 145 security risks and 424 security practices for managing security in the SDLC to integrate security into the overall development cycle. This study is prescriptive and can provide software developers with simple security guidelines at each stage of the SDLC. It covers a six-phase SDLC and the prescriptive activities that must be completed at each stage.

TABLE 13. Secure testing practices (STP).

S. No	Secure Testing Practices (STP)	Freq
STP1	Perform Penetration Testing	32
STP2	Perform Static Analysis Security Testing	30
STP3	Perform Fuzz Testing	16
STP4	Perform Dynamic Analysis Security Testing	14
STP5	Perform Vulnerability Scanning	9
STP6	Develop threat models: It helps to developing test cases or test plans	7
STP7	Perform Functional and Non Functional Testing	6
STP8	Use manually reviewing the code	6
STP9	Perform Unit Testing	5
STP10	Perform Automatic Patch generation	5
STP11	Perform Integration Testing	4
STP12	Develop test plan to describe software testing scope and activities	3
STP13	Validate correct use of security testing tools	3
STP14	Conduct Attack Surface Review	3
STP15	Perform risk based security testing	2
STP16	Rank the areas of the program where an exploit would be easiest	2
STP17	Verify security attributes of resources	2
STP18	Implement reverse engineering and software disassembling	2
STP19	Test security audit and review	2
STP20	The experts need to evaluate the security mitigations effectively	2
STP21	Use advanced Security Testing Tools, such as: Fortify SCA, Checkmarx code analysis, HP Web inspect, Acunetix wen, IBM AppScan	2
STP22	Perform Alpha and Beta Testing	1
STP23	Perform security regression testing	1
STP24	Perform run-time verification	1
STP25	Perform system testing	1
STP26	Identify the areas closest to the attack surface	1
STP27	Identify resource-driven security tests	1
STP28	Design test cases to attack software successfully	1
STP29	Prioritize the test cases	1
STP30	Consider all assumptions and business processes	1
STP31	Load and operate the software in a test environment and test against each of the test cases designed	1
STP32	Test states and state preservation	1
STP33	Test cases should be developed based on functional and security requirements	1
STP34	Document security test cases	1
STP35	Receive permission to perform security testing	1
STP36	Extreme testing should be conducted	1
STP37	Review unfixed security bugs	1
STP38	Apply secure code documentation	1

The software development stages are Requirement, Design, Coding, Testing, Deployment, and Maintenance.

The findings of this study show that security risks in the RE phase of the SDLC are a highly rated factor, in the development of secure software. It stands on the top (97.5%) amongst all the identified security risks. We conclude that many software security issues stem from insufficient or incorrect identification, documentation, analysis, mapping, prioritization, specification, and availability of security requirements. The importance of identifying non-functional security requirements should be stressed more because it aids in the reduction or elimination of software vulnerabilities [2], [61], [100]. Misuse cases are similar to use cases in that they specify what a system should not do, and they are a great way to get security requirements [97], [100]–[102].

Section IV-A shows that software security risks were highlighted in the design phase of the SDLC in 64 percent of the studies in our SLR. This is because design-level flaws are the most common sources of security risks in software systems [32]. We conclude that ''lack of developing threat modeling,'' ''improper secure design documentation,'' and ''lack of attention to follow security design principles'' are the three topmost security issues in the design phase. To mitigate the risks in the design phase, Table 5 presents that ''enumerate threats and prioritize the threat based on the potential impact,'' ''follow least privilege design principle,'' ''implement a defense-in-depth policy which includes multilevel security,'' ''revise or review design implementation,'' and ''implement security design decisions: (security cryptographic protocols, standards, services, and mechanisms)'' are the most highlighted security practices in the design phase.

The antivirus, intrusion detection mechanisms, and firewalls are not enough to reduce the risk in the coding phase of the SDLC. It needs further various suitable security defenses, practices, analysis, and countermeasures that result in further secure the released code [74], [75]. The findings of this study show that ''software security often fail because their development is generally based on ad-hoc foundations or follow

TABLE 14. Secure deployment practices (SDeP).

S. No	Secure Deployment Practices (SDeP)	Freq
SDeP1	Perform static analysis	30
SDeP2	Perform final security review to find any remaining security flaws	12
SDeP3	Perform Security Assessment and Secure Configuration	8
SDeP4	Establish a plan to review to reduce the time and resources	6
SDeP6	Analyze the overall state of the software	6
SDeP7	Verify that whether security practices have been followed during the software development	6
SDeP8	Certify release and archive	5
SDeP9	Implement Security release checklists	3
SDeP10	Configure the monitoring and logging	3
SDeP11	Identify security breaks	3
SDeP12	Verify output validation	3
SDeP13	Ensure that work products meet their specified security requirements	3
SDeP14	Demonstrate that the product fulfills the security expectations when placed in the intended operational environment	3
SDeP15	Perform Code Integrating and Handling	2
SDeP16	Perform Security Feed back	2
SDeP17	Response Planning & Execution	2
SDeP18	Upgrade the new version by fixing all identified flaws	2
SDeP19	Implement global security policy	2
SDeP20	Perform Data sanitization & Safe Disposal	1
SDeP21	Perform Code sign-off	1
SDeP22	Upload debugging symbols to central server	1
SDeP23	Sign identified targets	1
SDeP24	Obtain code signing credentials	1
SDeP25	Perform Threat Models Updating	1
SDeP26	Release Preparation	1
SDeP27	Document Technical Stack: Document the components used to build, test, deploy, and operate the software	1

traditional development processes," "lack of using secure coding practices," "lack of security awareness, training," "messy code, code bad smells, dead code," and "buffer and array overflow" are the topmost security risks in the coding phase. To mitigate these risks, software development organizations need to "perform code review," "provide security knowledge and training to software developers," "implement static code analysis" and "secure code writing" during the secure design phase.

Section IV-A portrays that software security was considered in the testing phase of the SDLC in 53 percent of the studies. The security testing approach is one of the most important, efficient, and widely used methods for improving software security, as it is used to detect vulnerabilities and ensure security functionality. Threat modeling is a systematic method for identifying threats that may compromise security, and it is considered a well-known accepted practice by the software testing industry [94]. We conclude that "lack of static, dynamic, penetration, and vulnerability analysis security testing," "lack of secure test cases" and "lack of security test documentations" are the topmost security risks in the testing phase of the SDLC. Table 13 presents that "perform penetration, static and dynamic analysis, fuzz testing, and vulnerability scanning testing," "develop threat models: it helps to develop test cases or test plans" and "use manually reviewing the code" are the most highlighted security practices for secure testing.

The deployment stage deals with release and change management. The software is installed in its real environment at this stage. It may appear easy, but integrating software into an existing environment can be difficult. Patches are developed to address the flaws, but the software remains vulnerable to a variety of security threats. In this stage, it is important to monitor responses to flaws and vulnerabilities of the system to check for newly evolved security threats. "After identifying new security risks, the same should be included in the requirement stage for further security improvements in subsequent releases" [66], [91]. Static analysis and peer review are two useful procedures for mitigating or minimizing newly discovered vulnerabilities [14]. Final security reviews and audits are performed during the secure deployment phase [14].

Similarly, we aimed to discover any security-related risks and their practices in the software maintenance phase of the SDLC. The maintenance team should keep track of new threats that the system encounters. We conclude that "perform static analysis," "perform final security review to find any remaining security flaws," "perform security assessment and secure configuration," "establish a plan to review to reduce the time and resources" and "analyze the overall state of the software" are the most cited practices for the secure maintenance of software.

Based on the foregoing discussion, we conclude that securing software systems in the post-development phases is insufficient and that better ways and means of securing software systems in the early stages are urgently required. To incorporate security in the overall SDLC, we have done a detailed literature review and identified 145 security risks and 424 best practices that help software development organizations to manage the security in each phase of the SDLC.

TABLE 15. Secure maintenance practices (SMP).

S. No	Secure Maintenance Practices (SMP)	Freq
SMP1	Find out new threats to the system	15
SMP2	Prioritize the new threats and their potential impact	15
SMP3	Prepare the database for similar kinds of threats	14
SMP4	Mitigate the new threats identified	14
SMP5	Identify attack surface area for the new threats	13
SMP6	Review the weakness of all such areas about new threats identified	13
SMP7	Develop security patches for the threats	12
SMP8	Release the patches to protect software from security breaches	12
SMP9	Perform configuration, vulnerability management, and change control	8
SMP10	Create and Execute incident response plan	5
SMP11	Educate users in using the software application in a secure manner	4
SMP12	Verify vulnerability correction	4
SMP13	Use regularly update Anti-virus Software	4
SMP14	Establish and maintain a set of process assets and work environment standards for developing secure products	3
SMP15	Reactive approaches should be adopted in the maintenance phase. They deal with developing patches after attacks have been made on the product.	3
SMP16	Address deployment-time security issues (Security response execution)	2
SMP17	Keep software up to date on security patches	2
SMP18	Manage security issue disclosure process	1
SMP19	Operate enablement	1
SMP20	Build a security response center	1
SMP21	Decide on what to skip	1
SMP22	Provide means of communication for security issues	1
SMP23	Manage relationship with the bug reporter	1
SMP24	Create and test the fixation of security threats	1
SMP25	The maintenance of software is made easier and more manageable through the structured approach provided by the SecSDM.	1
SMP26	Intelligence: Practice for collecting corporate knowledge used in carrying out software security activities throughout the organization	1
SMP27	Create and release security bulletin/advisory (Content creation)	1
SMP28	Improve process based on lessons learned	1
SMP29	Perform continuous monitoring & periodic security assessment	1
SMP30	Establish and maintain a security roadmap for process improvement	1
SMP31	Establish and maintain collaborations with external organizations promoting system security	1
SMP32	Continuously improve the security process by piloting innovative ideas, new technologies, and tools to improve organizational capability related to security	1

The important activities to follow during the development lifecycle to build secure software were specified in this study. The specified actions are successfully incorporated into each phase of the SDLC, reducing effort, time, and budget while delivering secure software. This effort should aid software development companies in increasing the security level of their goods and improving their security performance. This will raise the developer's understanding of secure development methods as well.

In the future, we intend to develop a software security assurance model [19] for global software development (GSD) vendor organizations. This model will assist GSD vendors to determine their readiness for secure software development. We will develop the model using the results of this study, industrial survey, case study, supervisor inputs, and lessons learned from the existing studies ([2], [5], [20], [51], [64], [100]). The model will generate several assessment reports, including a list of security risks/threats and their practices that GSD vendor organizations will use in each phase of the SDLC. In the future, we aim to answer the following research questions (RQs) to achieve the above-mentioned objectives:

RQ1: According to the industrial survey, what are the security threats to the development of secure software products that GSD vendor organizations should avoid?

RQ2: What are the mitigation practices that GSD vendor organizations can use to create secure software products, as identified during the industrial survey?

RQ3: Is the proposed software security assurance model capable of assisting GSD vendor organizations in determining their readiness to develop secure software?

VI. THREATS TO VALIDITY

The study's validity is concerned with the reliability of its findings. The following are the limitations for this systematic literature review:

- **Construct Validity**

To broaden the scope of the study, we conducted a systematic search using a wide range of words in the sample. The study's keywords were included after thorough discussions and suggestions by the two authors to ensure the validity of the study and to include as much relevant literature as possible. Another threat to build validity was the use of digital libraries for the

TABLE 16. Selected studies and quality assessment score.

P-Id	Paper Titles	Year	QA1	QA2	QA3	QA4	QA5	QA6	QA7	Total Score (QA1-QA7)
1.	Driving Secure Software Development Experiences in a Diverse Product Environment	2012	0.5	1	0.5	0	1	1	0	4
2.	Case Base for Secure Software Development Using Software Security Knowledge Base	2015	1	1	0.5	0	1	1	0	4.5
3.	A Preventive Secure Software Development Model for a software factory: a case study	2020	1	1	1	1	1	1	0	6
4.	System Analysis and Design Using Secure Software Development Life Cycle Based On ISO 31000 and STRIDE. Case Study Mutiara Ban Workshop	2020	1	1	0.5	1	1	1	0	5.5
5.	Integrating Static Analysis into a Secure Software Development Process	2008	0.5	0.5	0.5	1	1	1	0	4.5
6.	Managing the Secure Software Development	2019	0.5	1	0.5	0	1	1	0	4
7.	An integrated security testing framework for Secure Software Development Life Cycle	2016	0.5	1	0	0.5	1	1	0	4
8.	Secure Software Development through Coding Conventions and Frameworks	2007	1	1	0	1	1	1	0	5
9.	Security in Software Engineering Requirement	2013	1	1	0	0	1	1	0	4
10.	S2D-ProM: A Strategy Oriented Process Model for Secure Software Development	2007	1	1	0.5	0	1	1	0	4.5
11.	Activity and Artifact Views of a Secure Software Development Process	2009	0.5	1	0.5	0	1	1	0	4
12.	Quantifying Security in Secure Software Development Phases	2008	0.5	1	0	0	1	1	0	3.5
13.	Adaption of a Secure Software Development Methodology for Secure Engineering Design	2020	1	1	1	1	1	1	0	6
14.	A Review Paper : Security Requirement Patterns for a Secure Software Development	2019	1	0.5	0	1	1	1	0	4.5
15.	A Knowledge Transfer Framework for Secure Coding Practices	2015	0.5	1	0	0.5	1	1	0	4
16.	On Selecting Appropriate Development Processes and Requirements Engineering Methods for Secure Software	2009	1	1	0	0.5	1	1	0	4.5
17.	The Impact of Software Security Practices on Development Effort: An Initial Survey	2019	1	1	1	1	1	1	1	7
18.	A Readiness Model for Security Requirements Engineering	2018	1	1	1	1	1	1	1	7
19.	Secure Software Developing Recommendations	2019	1	1	0	0	1	1	0	4
20.	Software Security in Practice	2011	0.5	1	0	0	1	1	0	3.5
21.	Security Considerations for the Development of Secure Software Systems	2019	1	1	0.5	0.5	1	1	0	5
22.	A Methodological Approach to Apply Security Tactics in Software Architecture Design	2013	0.5	1	0.5	1	1	1	0	5
23.	Static Analysis for Web Service Security – Tools & Techniques for a Secure Development Life Cycle	2015	1	0.5	0.5	0	1	1	0	4
24.	Literature Review of the Challenges of Developing Secure Software Using the Agile Approach	2015	1	1	1	1	1	1	1	7
25.	Rules of Thumb for Developing Secure Software: Analyzing and consolidating two proposed sets of rules	2008	0.5	1	0	0	1	1	0	3.5
26.	Automated Software Architecture Security Risk Analysis using Formalized Signatures	2013	1	0.5	0.5	0.5	1	1	0	4.5
27.	Security Guidelines: Requirements Engineering for Verifying Code Quality	2016	0.5	1	0.5	0.5	1	1	0	4.5
28.	Security-aware Software Development Life Cycle (SaSDLC) – Processes and Tools	2009	1	1	0	0	1	1	0	4
29.	A Comprehensive Pattern-Driven Security Methodology for Distributed Systems	2014	0.5	1	1	1	1	1	0	5.5
30.	Best Practices for Software Security: An Overview	2008	1	1	0.5	0	1	1	0	4.5
31.	An Integrated Approach to Security in Software Development Methodologies	2008	1	1	0.5	0.5	1	1	0	5
32.	How can the developer benefit from security modeling?	2007	0.5	0.5	0.5	1	1	1	0	4.5
33.	Embedding Security in Software Development Life Cycle (SDLC)	2016	0.5	1	1	0	1	1	0	4.5

TABLE 16. *(Continued.)* **Selected studies and quality assessment score.**

34.	Software development Life cycle model to improve maintainability of software applications	2014	1	1	1	0	1	1	0	5
35.	A Methodology for Enhancing Software Security During Development Processes	2018	1	1	0.5	0.5	1	1	0	5
36.	Integrating Risk assessment and Threat modeling within SDLC process	2016	1	1	0.5	1	1	1	0	5.5
37.	Towards a Secure Software Development Lifecycle with SQUARE+R	2012	1	0.5	0.5	0.5	1	1	0	4.5
38.	Strong security starts with software development	2020	1	1	0	0	1	1	0	4
39.	STORE: Security Threat Oriented Requirements Engineering Methodology	2018	1	1	1	1	1	1	0	6
40.	Countermeasure Graphs for software security risk assessment: An Action research	2013	1	1	1	1	1	1	1	7
41.	Securing Web Applications from Injection and Logic Vulnerabilities: Approaches and Challenges	2016	1	1	1	1	1	1	0	6
42.	On the secure software development process: CLASP, SDL and Touchpoints compared	2009	0.5	1	0.5	0	1	1	0	4
43.	Securing Web applications	2008	1	1	0	0	1	1	0	4
44.	Diversification and Obfuscation Techniques for Software Security: a Systematic Literature Review	2018	1	1	1	1	1	1	1	7
45.	Investigating Security Threats in Architectural Context: Experimental Evaluations of Misuse Case Maps	2015	0.5	1	1	1	1	1	1	6.5
46.	Secure software development: a prescriptive framework	2011	1	1	0	0	1	1	0	4
47.	Cross Site Scripting: Removing Approaches in Web Application	2017	0.5	1	0	0	1	1	0	3.5
48.	Exploring Software Security Approaches in Software Development Lifecycle: A Systematic Mapping Study	2016	1	1	1	1	1	1	1	7
49.	An aspect-oriented approach for the systematic security hardening of code	2008	1	1	1	0	1	1	0	5
50.	An empirical study to improve software security through the application of code refactoring	2018	0.5	1	1	1	1	1	1	6.5
51.	Developing a Novel Holistic Taxonomy of Security Requirements	2015	0.5	1	0.5	0	1	1	0	4
52.	Threat Analysis of Software Systems: A Systematic Literature Review	2018	0.5	1	1	1	1	1	1	6.5
53.	Applying Software Assurance and Cybersecurity NICE Job Tasks through Secure Software Engineering Labs	2019	0.5	1	0	1	1	1	0	4.5
54.	Toward effective adoption of secure software development practices	2018	1	1	0.5	1	1	1	1	6.5
55.	A maturity model for secure requirements engineering	2020	1	1	1	1	1	1	1	7
56.	Empirical Analysis of Web Attacks	2016	1	0	0	1	1	1	0	4
57.	Survey and analysis on Security Requirements Engineering	2012	1	1	0.5	0.5	1	1	0	5
58.	A systematic review of security requirements engineering	2010	1	1	1	1	1	1	1	7
59.	Engineering Secure Systems: Models, Patterns and Empirical Validation	2018	1	0.5	0.5	1	1	1	1	6
60.	The Study of the Effectiveness of the Secure Software Development Life-Cycle Models in IT Project Management	2019	0.5	1	0	0	1	1	0	3.5
61.	Towards the Integration of Security Practices in the Software Implementation Process of ISO/IEC 29110: A Mapping	2017	1	1	1	1	1	1	0	6
62.	Integrating security and privacy in software development	2020	1	1	0.5	0	1	1	1	5.5
63.	The ISDF Framework: Integrating Security Patterns and Best Practices	2009	0.5	1	0.5	0	1	1	0	4
64.	Time for Addressing Software Security Issues: Prediction Models and Impacting Factors	2016	1	0.5	1	1	1	1	1	6.5
65.	A framework for development of secure software	2013	1	1	1	1	1	1	0	6
66.	SecSDM: A Model for Integrating Security into the Software Development Life Cycle	2007	0.5	1	0	0	1	1	0	3.5
67.	Towards Incorporation of Software Security	2011	0.5	1	1	0.5	1	1	0	5

TABLE 16. *(Continued.)* **Selected studies and quality assessment score.**

	Testing Framework in Software Development									
68.	Critical Review on Software Testing: Security Perspective	2016	1	1	0.5	0.5	1	1	0	5
69.	Systematic review of web application security development model	2013	1	1	1	1	1	1	0	6
70.	Software Security Requirements Engineering: State of the Art	2015	1	1	0.5	0.5	1	1	0	5
71.	Identifying the implied: Findings from three differentiated replications on the use of security requirements templates	2016	0.5	0.5	1	1	1	1	1	5
72.	Effectiveness and performance analysis of model-oriented security requirements engineering to elicit security requirements: a systematic solution for developing secure software systems	2015	1	1	0.5	1	1	1	0	5.5
73.	A descriptive study of Microsoft's threat modeling technique	2013	1	1	1	1	1	1	1	7
74.	Sharing Lessons Learned: "Practiced" Practices for Software Security	2012	0.5	1	0.5	0	1	0.5	0	3.5
75.	The risks analysis like a practice of secure software development. A revision of models and methodologies	2006	1	1	0	0.5	1	1	0	4.5
76.	Knowledge-based security testing of web applications by logic programming	2017	1	0.5	1	0.5	1	1	1	6
77.	Mitigating Security Threats Using Tactics and Patterns: A Controlled Experiment	2016	1	1	0.5	0.5	1	1	0	5
78.	Research on Software Security Awareness: Problems and Prospects	2010	1	1	0.5	1	1	1	0	5.5
79.	MAC and UML for Secure Software Design	2004	0.5	1	0	0.5	1	0.5	0	3.5
80.	Bringing Security Home: A process for developing secure and usable systems	2004	1	1	0.5	1	1	1	0	5.5
81.	Layered Security Architecture for Threat Management using Multi-Agent System	2011	1	1	1	1	1	1	0	6
82.	Human Factors in Software Security Risk Management	2007	1	0.5	0.5	0	1	0.5	0	3.5
83.	Mitigation of SQL Injection Attacks using Threat Modeling	2014	1	1	0.5	0.5	1	1	0	5
84.	Managing Security in Software	2019	0.5	0.5	1	1	1	1	0	5
85.	TAM 2 : Automated Threat Analysis	2012	1	1	1	0.5	1	1	0	5.5
86.	The State of the Art on Secure Software Engineering: A Systematic Mapping Study	2020	1	1	1	1	1	1	1	7
87.	Threading Secure Coding Principles and Risk Analysis into the Undergraduate Computer Science and Information Systems Curriculum	2006	0.5	1	0.5	0.5	1	1	0	4.5
88.	Costing Secure Soft ware Development - A Systematic Mapping Study	2019	0.5	1	1	1	1	1	0	5.5
89.	ASIDE: IDE Support for Web Application Security	2011	1	0.5	0.5	0.5	1	1	0	4.5
90.	Interventions for long-term software security creating a lightweight program of assurance techniques for developers	2019	1	1	1	1	1	1	0	6
91.	Sensei: Enforcing secure coding guidelines in the integrated development environment	2019	0.5	1	0.5	1	1	1	0	5
92.	Threat-oriented security framework in risk management using multiagent system	2012	1	1	1	1	1	1	0	6
93.	Software Security	2008	0.5	1	0.5	0.5	1	0.5	0	3.5
94.	The practice of secure software development in SDLC: an investigation through existing model and a case study	2016	1	1	1	1	1	1	0	6
95.	Unified threat model for analyzing and evaluating software threats	2012	0.5	1	1	1	1	1	0	5.5
96.	A threat model-based approach to security testing	2012	1	1	0.5	0.5	1	1	1	6
97.	Assessing and improving the quality of security methodologies for distributed systems	2018	1	1	1	1	1	1	1	7
98.	A Survey on Design Methods for Secure Software Development	2017	1	1	1	1	1	1	0	6
99.	Strategies for Secure Software Development	2013	1	1	0	0	1	0.5	0	3.5
100.	A Study of the Evolution of Secure Software Development Architectures	2018	1	1	0.5	0.5	1	1	0	5
101.	A security specific knowledge modelling	2020	1	1	1	1	1	1	0	6

TABLE 16. *(Continued.)* **Selected studies and quality assessment score.**

	approach for secure software engineering									
102.	A Review of Factors Influencing Implementation of Secure Software Development Practices	2016	0.5	1	1	1	1	1	1	6.5
103.	Secure Software Development Practice Adoption Model: A Delphi Study	2018	0	1	0.5	0.5	1	0.5	0	3.5
104.	A Comparative Study of Software Requirements Tools For Secure Software Development	2010	0.5	1	1	0.5	1	0.5	0	4.5
105.	Synthesis of Secure Software Development Controls	2015	0	0.5	1	0.5	1	1	0	4
106.	Evaluation of Engineering Approaches in the Secure Software Development Life Cycle	2014	0.5	1	0.5	1	1	1	0	5
107.	Essential Activities for Secure Software Development	2020	1	0.5	1	1	1	1	0	5.5
108.	Secure Software Engineering: Learning from the Past to Address Future Challenges	2009	1	1	1	0.5	1	1	0	5.5
109.	A Case for the Economics of Secure Software Development	2016	0.5	1	0.5	0.5	1	1	0	4.5
110.	Teaching Secure Software Engineering: Writing Secure Code	2011	1	1	0.5	0	0.5	0.5	0	3.5
111.	Security Requirement Elicitation Phase of Secure Software Development Life Cycle	2013	1	1	1	0	1	1	0	5
112.	A framework to support alignment of secure software engineering with legal regulations	2011	0.5	1	1	1	1	1	0	5.5
113.	A New Model for Secure Software Development	2009	1	1	1	1	1	1	0	6
114.	Model Driven Architecture for Secure Software Development Life Cycle	2016	0.5	1	0.5	0	1	1	0	4
115.	Secure Software Engineering: Evaluation of Emerging Trends	2017	0.5	1	1	1	1	1	0	5.5
116.	A Secure Software Development Supported by Knowledge Management	2010	1	1	0.5	0	1	1	0	4.5
117.	Motorola Secure Software Development Model	2008	1	1	0.5	0.5	1	1	0	5
118.	Towards Secure Software Engineering	2020	1	1	0	0	1	1	0	4
119.	Significance of Security Metrics in Secure Software Development	2017	0.5	1	0	0	1	1	0	3.5
120.	Costing Secure Soft ware Development - A Systematic Mapping Study	2019	1	1	1	1	1	1	0	6
121.	Building and Validating a Scale for Secure Software Development Self-Efficacy	2020	0.5	1	1	1	1	1	1	6.5

studies. This threat was mitigated by identifying six digital libraries as key sources in such a domain.

- **Internal Validity**

Internal validity threats have been reduced to the point where all interested readers are encouraged to view the data extracted from the papers of the studies displayed without restrictions.

- **Conclusion Validity**

To minimize the threats, each step of the data collection, extraction, and analysis was checked through a systematic process and periodic reviews by the participating authors. The rationale for this move was that the same method has been used in the literature for similar studies.

- **External Validity**

External validity includes how much it is possible to generalize the outcomes of this study. To reduce this issue, the ratio of security risks and their practices have been included in this work.

APPENDIX

See Table 16.

ACKNOWLEDGMENT

The authors appreciate all of the critiques and ideas from the Software Engineering Research Group at the University of Malakand (SERG UOM).

REFERENCES

[1] M. Tatam, B. Shanmugam, S. Azam, and K. Kannoorpatti, "A review of threat modelling approaches for APT-style attacks," *Heliyon*, vol. 7, no. 1, Jan. 2021, Art. no. e05969.
[2] M. Niazi, A. M. Saeed, M. Alshayeb, S. Mahmood, and S. Zafar, "A maturity model for secure requirements engineering," *Comput. Secur.*, vol. 95, Aug. 2020, Art. no. 101852.
[3] M. Zhang, X. D. C. D. Carnavalet, L. Wang, and A. Ragab, "Large-scale empirical study of important features indicative of discovered vulnerabilities to assess application security," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 9, pp. 2315–2330, Sep. 2019.
[4] G. McGraw, "Six tech trends impacting software security," *Computer*, vol. 50, no. 5, pp. 100–102, May 2017.
[5] J. C. S. Nunez, A. C. Lindo, and P. G. Rodriguez, "A preventive secure software development model for a software factory: A case study," *IEEE Access*, vol. 8, pp. 77653–77665, 2020.

[6] S. Von Solms and L. A. Futcher, "Adaption of a secure software development methodology for secure engineering design," *IEEE Access*, vol. 8, pp. 125630–125637, 2020.
[7] M. Z. Gunduz and R. Das, "Cyber-security on smart grid: Threats and potential solutions," *Comput. Netw.*, vol. 169, Mar. 2020, Art. no. 107094.
[8] J. Li, Y. Zhang, X. Chen, and Y. Xiang, "Secure attribute-based data sharing for resource-limited users in cloud computing," *Comput. Secur.*, vol. 72, pp. 1–12, Jan. 2018.
[9] A. Sharma and M. P. Kumar, "Aspects of enhancing security in software development life cycle," *Adv. Comput. Sci. Technol.*, vol. 10, no. 2, pp. 203–210, 2017.
[10] W. Khreich, S. S. Murtaza, A. Hamou-Lhadj, and C. Talhi, "Combining heterogeneous anomaly detectors for improved software security," *J. Syst. Softw.*, vol. 137, pp. 415–429, Mar. 2018.
[11] S. Hosseinzadeh, S. Rauti, S. Laurén, and J.-M. Mäkelä, "Diversification and obfuscation techniques for software security: A systematic literature review," *Inf. Softw. Technol.*, vol. 104, pp. 72–93, Dec. 2018.
[12] E. K. Szczepaniuk, H. Szczepaniuk, T. Rokicki, and B. Klepacki, "Information security assessment in public administration," *Comput. Secur.*, vol. 90, Mar. 2020, Art. no. 101709.
[13] M. A. Akbar, A. Alsanad, S. Mahmood, and A. Alothaim, "A multicriteria decision making taxonomy of IoT security challenging factors," *IEEE Access*, vol. 9, pp. 128841–128861, 2021.
[14] R. Khan, "Secure software development: A prescriptive framework," *Comput. Fraud Secur.*, vol. 2011, no. 8, pp. 12–20, Aug. 2011.
[15] A. K. Srivastava and S. Kumar, "An effective computational technique for taxonomic position of security vulnerability in software development," *J. Comput. Sci.*, vol. 25, pp. 388–396, Mar. 2018.
[16] D. Mellado, C. Blanco, L. E. Sánchez, and E. Fernández-Medina, "A systematic review of security requirements engineering," *Comput. Standards Interfaces*, vol. 32, no. 4, pp. 153–165, 2010.
[17] I. Velásquez, A. Caro, and A. Rodríguez, "Authentication schemes and methods: A systematic literature review," *Inf. Softw. Technol.*, vol. 94, pp. 30–37, Feb. 2018.
[18] Y. Lee and G. Lee, "HW-CDI: Hard-wired control data integrity," *IEEE Access*, vol. 7, pp. 10811–10822, 2019.
[19] R. A. Khan and S. U. Khan, "A preliminary structure of software security assurance model," in *Proc. 13th Int. Conf. Global Softw. Eng.*, Gothenburg, Sweden, May 2018, pp. 137–140.
[20] H. Al-Matouq, S. Mahmood, M. Alshayeb, and M. Niazi, "A maturity model for secure software design: A multivocal study," *IEEE Access*, vol. 8, pp. 215758–215776, 2020.
[21] S. Moyo and E. Mnkandla, "A novel lightweight solo software development methodology with optimum security practices," *IEEE Access*, vol. 8, pp. 33735–33747, 2020.
[22] N. S. A. Karim, A. Albuolayan, T. Saba, and A. Rehman, "The practice of secure software development in SDLC: An investigation through existing model and a case study," *Secur. Commun. Netw.*, vol. 9, no. 18, pp. 5333–5345, Dec. 2016.
[23] S. R. Ahmed, "Secure software development—Identification of security activities and their integration in software development lifecycle," M.S. thesis, School Eng., Blekinge Inst. Technol., Ronneby, Sweden, 2007.
[24] S. Z. Hlaing and K. Ochimizu, "An integrated cost-effective security requirement engineering process in SDLC using FRAM," in *Proc. CSCI*, Dec. 2018, pp. 852–857.
[25] M. Khari and P. Kumar, "Embedding security in software development life cycle (SDLC)," in *Proc. 3rd Int. Conf. Comput. Sustain. Global Develop.*, Mar. 2016, pp. 2182–2186.
[26] N. M. Mohammed, M. Niazi, M. Alshayeb, and S. Mahmood, "Exploring software security approaches in software development lifecycle: A systematic mapping study," *Comput. Standards Interface*, vol. 50, pp. 107–115, Feb. 2017.
[27] P. Silva, R. Noël, M. Gallego, S. Matalonga, and H. Astudillo, "Software development initiatives to identify and mitigate security threats—A systematic mapping," in *Proc. CIbSE*, 2016, pp. 257–270.
[28] A. S. Guinea, G. Nain, and Y. L. Traon, "A systematic review on the engineering of software for ubiquitous systems," *J. Syst. Softw.*, vol. 118, pp. 251–276, Aug. 2016.
[29] C. Meshram, A. Alsanad, J. V. Tembhurne, S. W. Shende, K. W. Kalare, S. G. Meshram, M. A. Akbar, and A. Gumaei, "A provably secure lightweight subtree-based short signature scheme with fuzzy user data sharing for human-centered IoT," *IEEE Access*, vol. 9, pp. 3649–3659, 2021.
[30] S. Rafi, W. Yu, M. A. Akbar, A. Alsanad, and A. Gumaei, "Prioritization based taxonomy of DevOps security challenges using PROMETHEE," *IEEE Access*, vol. 8, pp. 105426–105446, 2020.
[31] A. Hudaib, M. Alshraideh, O. Surakhi, and M. Alkhanafseh, "A survey on design methods for secure software development," *Int. J. Comput. Techol.*, vol. 16, pp. 7047–7064, Dec. 2017.
[32] R. A. Khan, S. U. Khan, H. U. Khan, and M. Ilyas, "Systematic mapping study on security approaches in secure software engineering," *IEEE Access*, vol. 9, pp. 19139–19160, 2021.
[33] G. McGraw, "From the ground up: The DIMACS software security workshop," *IEEE Secur. Privacy*, vol. 1, no. 2, pp. 59–66, Mar. 2003.
[34] R. M. Parizi, K. Qian, H. Shahriar, F. Wu, and L. Tao, "Benchmark requirements for assessing software security vulnerability testing tools," in *Proc. 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2018, pp. 825–826.
[35] P. R. Khan, "Secure software development: A prescriptive framework," *Comput. Fraud Secur.*, vol. 2011, pp. 12–20, Jan. 2011.
[36] B. Potter and G. McGraw, "Software security testing," *IEEE Security Privacy*, vol. 2, no. 5, pp. 81–85, Sep. 2004.
[37] D. Verdon and G. McGraw, "Risk analysis in software design," *IEEE Security Privacy*, vol. 2, no. 4, pp. 79–84, Jul. 2004.
[38] S. Lipner, "The trustworthy computing security development lifecycle," in *Proc. 20th Annu. Comput. Secur. Appl. Conf.*, 2004, pp. 2–13.
[39] S. Gupta, M. Faisal, and M. Husain, "Secure software development process for embedded systems control," *Int. J. Eng. Sci. Emerg. Technol.*, vol. 4, pp. 133–143, Dec. 2012.
[40] M. Essafi, L. Jilani, and H. Ben Ghezala, "S2D-Prom: A strategy oriented process model for secure software development," in *Proc. Int. Conf. Softw. Eng. Adv.*, Aug. 2007, pp. 24–28.
[41] J. Manico, "OWASP," in *Proc. Appl. Secur. Verification Standard*, 2016, pp. 1–70.
[42] W. Li and T. Chiueh, "Automated format string attack prevention for Win32/X86 binaries," in *Proc. 23rd Annu. Comput. Secur. Appl. Conf.*, Dec. 2007, pp. 398–409.
[43] H. Peine, "Rules of thumb for developing secure software: Analyzing and consolidating two proposed sets of rules," in *Proc. 3rd Int. Conf. Availability, Rel. Secur.*, Mar. 2008, pp. 1204–1209.
[44] A. Hall and R. Chapman, "Correctness by construction: Developing a commercial secure system," *IEEE Softw.*, vol. 19, no. 1, pp. 18–25, Jan. 2002.
[45] I. Flechais, C. Mascolo, and M. A. Sasse, "Integrating security and usability into the requirements and design process," *Int. J. Electron. Secur. Digit. Forensic*, vol. 1, no. 1, pp. 12–26, 2007.
[46] B. Subedi, A. Alsadoon, P. W. C. Prasad, and A. Elchouemi, "Secure paradigm for web application development," in *Proc. 15th RoEduNet Conf., Netw. Educ. Res.*, Sep. 2016, pp. 1–6.
[47] A. S. Sodiya, S. A. Onashoga, and O. B. Ajayi, "Towards building secure software systems," *Issues Informing Sci. Inf. Technol.*, vol. 3, pp. 635–646, 2006.
[48] N. Mead and T. Stehney, "Security quality requirements engineering (SQUARE) methodology," *ACM SIGSOFT Softw. Eng. Notes*, vol. 30, pp. 1–7, Jul. 2005.
[49] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering—A systematic literature review," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 7–15, Jan. 2009.
[50] B. Kitchenham and C. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele Univ., Keele, U.K., Joint Rep. EBSE 2007-001, 2007.
[51] R. A. Khan, M. Y. Idris, S. U. Khan, M. Ilyas, S. Ali, A. U. Din, G. Murtaza, and A. W. Wahid, "An evaluation framework for communication and coordination processes in offshore software development outsourcing relationship: Using fuzzy methods," *IEEE Access*, vol. 7, pp. 112879–112906, 2019.
[52] F. de F. S. M. Russo and R. Camanho, "Criteria in AHP: A systematic review of literature," *Proc. Comput. Sci.*, vol. 55, pp. 1123–1132, Jul. 2015.
[53] S. U. Khan, M. Niazi, and R. Ahmad, "Factors influencing clients in the selection of offshore software outsourcing vendors: An exploratory study using a systematic literature review," *J. Syst. Softw.*, vol. 84, pp. 686–699, Apr. 2011.
[54] M. Staples and M. Niazi, "Systematic review of organizational motivations for adopting CMM-based SPI," *Inf. Softw. Technol.*, vol. 50, nos. 7–8, pp. 605–620, Jun. 2008.

[55] A. A. Khan, J. Keung, M. Niazi, S. Hussain, and A. Ahmad, "Systematic literature review and empirical investigation of barriers to process improvement in global software development: Client-vendor perspective," *Inf. Softw. Technol.*, vol. 87, pp. 180–205, Jul. 2017.
[56] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Inf. Softw. Technol.*, vol. 53, pp. 625–637, Jun. 2011.
[57] L. Chen, M. A. Babar, and H. N. Zhang, "Towards an evidence-based understanding of electronic data sources," in *Proc. Electron. Workshops Comput.*, Apr. 2010, pp. 135–138.
[58] S. Mahmood, S. Anwer, M. Niazi, M. Alshayeb, and I. Richardson, "Key factors that influence task allocation in global software development," *Inf. Softw. Technol.*, vol. 91, pp. 102–122, Nov. 2017.
[59] I. Keshta, M. Niazi, and M. Alshayeb, "Towards implementation of requirements management specific practices (SP1.3 and SP1.4) for Saudi Arabian small and medium sized software development organizations," *IEEE Access*, vol. 5, pp. 24162–24183, 2017.
[60] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," *Inf. Softw. Technol.*, vol. 55, pp. 2049–2075, Dec. 2013.
[61] M. Sulayman and E. Mendes, "A systematic literature review of software process improvement in small and medium web companies," in *Advances in Software Engineering*. Berlin, Germany: Springer, 2009, pp. 1–8.
[62] W. Afzal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties," *Inf. Softw. Technol.*, vol. 51, no. 6, pp. 957–976, 2009.
[63] V. Alexander and Y. M. Eun, *Analyzing Rater Agreement*. Oxfordshire, U.K.: Taylor & Francis, 2005.
[64] Y. Mufti, M. Niazi, M. Alshayeb, and S. Mahmood, "A readiness model for security requirements engineering," *IEEE Access*, vol. 6, pp. 28611–28631, 2018.
[65] H. Oueslati, M. M. Rahman, and L. B. Othmane, "Literature review of the challenges of developing secure software using the agile approach," in *Proc. 10th Int. Conf. Availability, Rel. Secur.*, Aug. 2015, pp. 540–547.
[66] S. Yahya, M. Kamalrudin, S. Sidek, M. Jaimun, J. Yusof, A. K. Hua, and P. Gani, "A review paper: Security requirement patterns for a secure software development," in *Proc. 1st Int. Conf. Artif. Intell. Data Sci. (AiDAS)*, Sep. 2019, pp. 146–151.
[67] P. Salini and S. Kanmani, "Survey and analysis on security requirements engineering," *Comput. Electr. Eng.*, vol. 38, no. 6, pp. 1785–1797, Nov. 2012.
[68] A. Van Den Berghe, R. Scandariato, K. Yskout, and W. Joosen, "Design notations for secure software: A systematic literature review," *Softw. Syst. Model.*, vol. 16, no. 3, pp. 809–831, Jul. 2017.
[69] V. Maheshwari and M. Prasana, "Integrating risk assessment and threat modeling within SDLC process," in *Proc. ICICT*, Aug. 2016, pp. 1–5.
[70] K. Khan, R. Ahmad, and I. Yazid, "Systematic mapping study protocol for secure software engineering," in *Proc. AIMC*, 2019, pp. 367–374.
[71] L. Y. Banowosari and B. A. Gifari, "System analysis and design using secure software development life cycle based on ISO 31000 and STRIDE. Case study mutiara ban workshop," in *Proc. 4th Int. Conf. Informat. Comput. (ICIC)*, Oct. 2019, pp. 1–6.
[72] G. Pedraza-Garcia, H. Astudillo, and D. Correal, "A methodological approach to apply security tactics in software architecture design," in *Proc. Colombian Conf. Commun. Comput.*, Jun. 2014, pp. 1–8.
[73] T. Doan, S. Demurjian, T. C. Ting, and A. Ketterl, "MAC and UML for secure software design," in *Proc. ACM workshop Formal methods Secur. Eng.*, Washington DC, USA, 2004, pp. 75–85.
[74] R. C. Seacord, *Secure Coding in C and C++*. Reading, MA, USA: Addison-Wesley, 2013.
[75] A. Mousa, M. Karabatak, and T. Mustafa, "Database security threats and challenges," in *Proc. 8th Int. Symp. Digit. Forensics Secur.*, Jun. 2020, pp. 1–5.
[76] D. Kleidermacher, "Integrating static analysis into a secure software development process," in *Proc. Conf. Technol. Homeland Secur.*, May 2008, pp. 367–371.
[77] H. Shirazi, "A new model for secure software development," *Int. J. Intell. Inf. Technol. Appl.*, vol. 3, pp. 136—143, Jan. 2009.
[78] D. Hein and H. Saiedian, "Secure software engineering: Learning from the past to address future challenges," *Inf. Secur. J., Global Perspective*, vol. 18, no. 1, pp. 8–25, Feb. 2009.
[79] G. Deepa and P. S. Thilagam, "Securing web applications from injection and logic vulnerabilities: Approaches and challenges," *Inf. Softw. Technol.*, vol. 74, pp. 160–180, Jun. 2016.
[80] J. Li, Y. K. Li, X. Chen, P. P. C. Lee, and W. Lou, "A hybrid cloud approach for secure authorized deduplication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1206–1216, Dec. 2015.
[81] A. Apvrille and M. Pourzandi, "Secure software development by example," *IEEE Security Privacy*, vol. 3, no. 4, pp. 10–17, Jul. 2005.
[82] M. Almorsy, J. Grundy, and A. S. Ibrahim, "Automated software architecture security risk analysis using formalized signatures," in *Proc. ICSE*, May 2013, pp. 662–671.
[83] R. Cope, "Strong security starts with software development," *Netw. Secur.*, vol. 2020, no. 7, pp. 6–9, Jul. 2020.
[84] D. Kaur and P. Kaur, "Empirical analysis of web attacks," *Proc. Comput. Sci.*, vol. 78, pp. 298–306, Jan. 2016.
[85] A. Hazeyama, M. Saito, N. Yoshioka, A. Kumagai, T. Kobashi, H. Washizaki, H. Kaiya, and T. Okubo, "Case base for secure software development using software security knowledge base," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, Jul. 2015, pp. 97–103.
[86] M. U. A. Khan and M. Zulkernine, "On selecting appropriate development processes and requirements engineering methods for secure software," in *Proc. 33rd Annu. Int. Comput. Softw. Appl. Conf.*, Jul. 2009, pp. 353–358.
[87] D. Baca and K. Petersen, "Countermeasure graphs for software security risk assessment: An action research," *J. Syst. Softw.*, vol. 86, no. 9, pp. 2411–2428, Sep. 2013.
[88] *CWE-Common Weakness Enumeration*. Accessed: Mar. 18, 2021. [Online]. Available: https://cwe.mitre.org/
[89] A. Masood and J. Java, "Static analysis for web service security—Tools & techniques for a secure development life cycle," in *Proc. HST*, Apr. 2015, pp. 1–6.
[90] W. S. Al-Shorafat, "Security in software engineering requirement," in *Proc. Int. Conf. Internet Technol. Secured Trans.*, Dec. 2013, pp. 666–673.
[91] G. Pedraza-Garcäa, R. Noël, S. Matalonga, H. Astudillo, and E. B. Fernandez, "Mitigating security threats using tactics and patterns: A controlled experiment," in *Proc. 10th Eur. Conf. Softw. Archit. Workshops*, Copenhagen, Denmark, 2016, p. 37.
[92] I. Rehman and S. Malik, "The impact of test case reduction and prioritization on software testing effectiveness," in *Proc. Int. Conf. Emerg. Technol.*, Oct. 2009, pp. 416–421.
[93] H. Mouratidis, P. Giorgini, and G. Manson, "When security meets software engineering: A case of modelling secure information systems," *Inf. Syst.*, vol. 30, no. 8, pp. 609–629, Dec. 2005.
[94] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security: From UML models to access control infrastructures," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, no. 1, pp. 39–91, 2006.
[95] H. Nina, J. A. Pow-Sang, and M. Villavicencio, "Systematic mapping of the literature on secure software development," *IEEE Access*, vol. 9, pp. 36852–36867, 2021.
[96] C. Camacho, S. Marczak, and T. Conte, "On the identification of best practices for improving the efficiency of testing activities in distributed software projects preliminary findings from an empirical study," in *Proc. IEEE 8th Int. Conf. Global Softw. Eng. Workshops*, Aug. 2013, pp. 1–4.
[97] A. Marback, H. Do, K. He, S. Kondamarri, and D. Xu, "A threat model-based approach to security testing," *Softw., Pract. Exper.*, vol. 43, no. 2, pp. 241–258, Feb. 2013.
[98] Y. Tung, S. Lo, J. Shih, and H. Lin, "An integrated security testing framework for secure software development life cycle," in *Proc. 18th Asia–Pacific Netw. Oper. Manage. Symp.*, Oct. 2016, pp. 1–4.
[99] A. R. S. Farhan and G. M. M. Mostafa, "A methodology for enhancing software security during development processes," in *Proc. 21st Saudi Comput. Soc. Nat. Comput. Conf.*, Apr. 208, pp. 1–6.
[100] A. Muhammad and A. Shafique, "Model driven architecture for secure software development life cycle," *Int. J. Comput. Sci. Inf. Secur.*, vol. 14, no. 6, pp. 649–661, 2016.
[101] V. Suburayan, "Software development life cycle model to improve maintainability of software applications," in *Proc. 4th Int. Conf. Adv. Comput. Commun.*, 2018, pp. 270–273.
[102] L. Catuogno, C. Galdi, and G. Persiano, "Secure dependency enforcement in package management systems," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 2, pp. 377–390, Oct. 2020.
[103] S. Islam and W. Dong, "Human factors in software security risk management," in *Proc. 1st Int. workshop Leadership Manage. Softw. Archit.*, Leipzig, Germany, 2008, pp. 13–16.

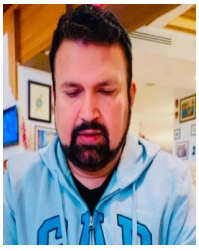
[104] M. Younas, M. A. Shah, D. N. A. Jawawi, M. K. Ishfaq, M. Awais, K. Wakil, and A. Mustafa, "Elicitation of nonfunctional requirements in agile development using cloud computing environment," *IEEE Access*, vol. 8, pp. 209153–209162, 2020.
[105] A.-U.-H. Yasar, D. Preuveneers, Y. Berbers, and G. Bhatti, "Best practices for software security: An overview," in *Proc. IEEE Int. Multitopic Conf.*, Dec. 2008, pp. 169–173.
[106] R. A. Khan, S. U. Khan, M. Ilyas, and M. Y. Idris, "The state of the art on secure software engineering: A systematic mapping study," in *Proc. Eval. Assessment Softw. Eng., Trondheim, Norway*, vol. 2020, pp. 487–492.
[107] L. B. Othmane, P. Angin, H. Weffers, and B. Bhargava, "Extending the agile development process to develop acceptably secure software," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 6, pp. 497–509, Nov. 2014.
[108] H. Mumtaz, M. Alshayeb, S. Mahmood, and M. Niazi, "An empirical study to improve software security through the application of code refactoring," *Inf. Softw. Technol.*, vol. 96, pp. 112–125, Apr. 2018.
[109] E. Venson, X. Guo, Z. Yan, and B. Boehm, "Costing secure software development: A systematic mapping study," in *Proc. 14th Int. Conf. Availability, Rel. Secur.*, Canterbury, CA, USA, 2019, p. 9.
[110] M. Sodanil, G. Quirchmayr, N. Porrawatpreyakorn, and A. M. Tjoa, "A knowledge transfer framework for secure coding practices," in *Proc. Int. Joint Conf. Comput. Sci. Softw. Eng.*, Jul. 2015, pp. 120–125.
[111] E. Venson, R. Alfayez, M. M. F. Gomes, R. M. C. Figueiredo, and B. Boehm, "The impact of software security practices on development effort: An initial survey," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Sep. 2019, pp. 1–12.
[112] P. Salini and S. Kanmani, "Effectiveness and performance analysis of model-oriented security requirements engineering to elicit security requirements: A systematic solution for developing secure software systems," *Int. J. Inf. Secur.*, vol. 15, no. 3, pp. 319–334, Jun. 2016.
[113] B. Musa Shuaibu, N. Md Norwawi, M. H. Selamat, and A. Al-Alwani, "Systematic review of web application security development model," *Artif. Intell. Rev.*, vol. 43, no. 2, pp. 259–276, Jan. 2015.
[114] B. De Win, R. Scandariato, K. Buyens, J. Grégoire, and W. Joosen, "On the secure software development process: CLASP, SDL and touchpoints compared," *Inf. Softw. Technol.*, vol. 51, no. 7, pp. 1152–1171, Jul. 2009.
[115] S. T. Siddiqui, "Significance of security metrics in secure software development," *Int. J. Appl. Inf. Syst.*, vol. 12, no. 6, pp. 10–15, Sep. 2017.
[116] R. E. Ahmed, "Software maintenance outsourcing: Issues and strategies," *Comput. Electr. Eng.*, vol. 32, no. 6, pp. 449–453, Nov. 2006.
[117] B. Chess and B. Arkin, "Software Security in Practice," *IEEE Security Privacy*, vol. 9, no. 2, pp. 89–92, Mar./Apr. 2011.
[118] S. Al-Amin, N. Ajmeri, H. Du, E. Z. Berglund, and M. P. Singh, "Toward effective adoption of secure software development practices," *Simul. Model. Pract. Theory*, vol. 85, pp. 33–46, Jun. 2018.
[119] L. Bracciale, P. Loreti, A. Detti, R. Paolillo, and N. B. Melazzi, "Lightweight named object: An ICN-based abstraction for IoT device programming and management," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5029–5039, Jun. 2019.

RAFIQ AHMAD KHAN received the M.Phil. degree in computer science with a specialization in software engineering from the University of Malakand, Khyber Pakhtunkhwa, Pakistan, under the research supervision of Dr. S. U. Khan, where he is currently pursuing the Ph.D. degree under the supervision of the same supervisor.

He has authored several articles in well-reputed international conferences and journals, including ICGSE and IEEE Access. His research interests include software security, empirical software engineering, systematic literature review, requirements engineering, green computing, software testing, agile software development, and global software engineering.

SIFFAT ULLAH KHAN received the Ph.D. degree in computer science from Keele University, U.K., in 2011.

He was the Head of the Department of Software Engineering, University of Malakand, Pakistan, for three years, where he was also the Chairperson of the Department of Computer Science and IT and is currently an Associate Professor in computer science. He is also the Founder and the Leader of the Software Engineering Research Group, University of Malakand. He has successfully supervised ten M.Phil. and four Ph.D. scholars. He has authored over 100 articles, so far, in well-reputed international conferences and journals. His research interests include software outsourcing, empirical software engineering, agile software development, systematic literature review, software metrics, cloud computing, requirements engineering, and green computing/IT. He received the Gold Medal (Dr. M. N. Azam Prize 2015) from the Pakistan Academy of Sciences in recognition of his research achievements in the field of computer (software).



HABIB ULLAH KHAN received the Ph.D. degree in management information systems from Leeds Beckett University, U.K. He is currently working as a Professor of information systems with the Department of Accounting and Information Systems, College of Business and Economics, Qatar University, Qatar. He has nearly 20 years of industry, teaching, and research experience. His research interests include IT adoption, social media, Internet addiction, mobile commerce, computer mediated communication, IT outsourcing, big data, and IT security.

MUHAMMAD ILYAS received the Ph.D. degree in computer science from the University of Malakand, Pakistan, where he is currently an Assistant Professor with the Department of Computer Science & IT. His research interests include software outsourcing, empirical software engineering, systematic literature review, cloud computing, requirements engineering, and green computing/IT.