IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 2, 2022, accepted May 19, 2022, date of publication May 26, 2022, date of current version June 7, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3178301

Security Assurance Model of Software Development for Global Software Development Vendors

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

[1]Software Engineering Research Group, Department of Computer Science and IT, University of Malakand, Chakdara 18800, Pakistan
[2]Department of Computer Science, Albaha University, Albaha 65799, Saudi Arabia

Corresponding author: Rafiq Ahmad Khan (rafiqahmadk@gmail.com)

This work was supported in part by the Department of Computer Science & IT, University of Malakand, Chakdara, Pakistan, and in part by the Department of Computer Science, Albaha University, Albaha, Saudi Arabia.

ABSTRACT The number of security attacks and the impact has grown considerably in the recent several years. As a result, new emerging software development models are required that assist in developing software that is secure by default. This article reviews the most widely used security software models. It proposes a new Security Assurance Model (SAM) for Software Development that is adaptable to all contemporary scenarios, emphasizing global software development (GSD) vendor companies. The SAM of Software Development was developed after studying 11 well-known development models and analyzing results obtained from a systematic literature review (SLR) and questionnaire survey. The SAM of Software Development consists of seven security assurance levels: Governance and Security Threat Analysis, Secure Requirement Analysis, Secure Design, Secure Coding, Secure Testing and Review, Secure Deployment, and Security Improvement. The security assurance levels of SAM of software development consist of 46 critical software security risks (CSSRs) and 388 practices for addressing these risks. The proposed SAM of Software Development was assessed based on a tool created by Motorola, which is used to evaluate the present state of a company's software processes and find areas for improvement. We conducted 3 case studies on software development companies, using data from real software projects to examine the results of a practical experiment in each company. The results of the case studies indicate that the proposed SAM of Software Development helps measure the security assurance level of an organization. In addition, it can potentially serve as a framework for researchers to develop new software security measures.

INDEX TERMS Secure software engineering, software development life cycle, global software development, systematic mapping study, systematic literature review, questionnaire survey, case study, security risks and practices.

I. INTRODUCTION

Security for software has become increasingly important since hacking and other attacks on computer systems have grown in popularity in the last few years. As a result, several researchers have examined security solutions as early as the requirement engineering phase. With the growth of the software business and the Internet, people are paying increasing attention to software system security. Managers and users of software systems may suffer enormous losses if the system's sensitive data is exposed or hacked and rendered inoperable.

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

To incorporate security into the software engineering paradigm, it should be considered from the start of the SDLC [1], [2]. Secure software engineering (SSE) is the process of designing, building, and testing software so that it becomes secure; this includes secure SDLC processes and secure software development (SSD) methods [3]–[5]. Most enterprises typically view security as a post-development procedure [6]. No consideration is given to security before development [7].

Khan [8] stated that as software development becomes more complex, distributed, and concurrent, security issues greatly influence software quality. Insecure software harms an organization's reputation 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 [8]. Most software programs are designed and deployed without attention to protection desires [9], [10].

Hidden attacking risks within or outside the organization emerge day by day, resulting in substantial financial loss and confidentiality and credibility losses by putting the availability and integrity of organizational data at risk [11], [12]. Numerous methodologies for assessing software quality have been developed, including the following: ''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'' [13].

The common phases of SDLC include requirement, design, coding, testing, deployment, and maintenance [14]. The final product will not be secure if security is not considered in all phases of the SDLC. 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 [15]–[17].

The literature [18]–[21] shows that the software industry has implemented numerous software security techniques, approaches, and solutions. There are a variety of maturity models available for evaluating software security processes, and many firms use these. However, our systematic mapping study [22], [23] revealed several limitations in the existing software security models. None of these is committed explicitly to identifying security risks and their practices in each phase of the SDLC. For global software development (GSD) vendor businesses, none of the existing models encompasses all components and activities of a secure SDLC. GSD vendor companies must be aware of the security threats while producing secure applications and risk mitigation techniques to ensure the SDLC's integrity. To increase their SDLC security, GSD vendors will measure their level of maturity and assurance. It will also make GSD engineers more aware of the issue.

This paper aims to develop a secure SDLC and Security Assurance Model (SAM) of Software Development for GSD vendor organizations to specify the requirements for secure software development better. As a result, GSD vendors will assess their level of security assurance and capacity to produce more secure software. To accomplish this, we investigate the following research questions (RQs):

RQ1: How can a secure SDLC be developed for GSD vendor companies that are both practical and robust?

RQ2: Is the proposed security assurance model capable of assisting GSD organizations in determining their security assurance to produce secure software?

This paper is organized as follows: Section II includes an overview of the relevant work and its summary. Section III goes into detail on how the study was conducted. Section IV presents a secure SDLC for GSD vendor businesses. Section V outlines the proposed SAM for software development and the security assurance levels. The validation of the model using case studies is presented in Section VI. Section VII highlights the limitations of the study. Section VIII concludes with a discussion of the findings and directions for future research.

II. LITERATURE REVIEW

A review of software security research and maturity models is presented in this section:

A. SOFTWARE SECURITY

Let's a look at some of the concepts about software security that have been discussed in the literature:

- ''Software security is the software's ability to resist, tolerate and recover from events that intentionally threaten its dependability [24]''.
- ''Software security is about building secure software: designing software to be secure, making sure that software is secure, and educating software developers, architects, and users about to build secure things [25]''.

B. SECURE SOFTWARE DEVELOPMENT PROCESS

There are many different ways to include security into the software development lifecycle (SDLC), and this section explains some of the most prevalent security techniques employed in these methodologies:

- McGraw [24], [26] 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 artefacts.
- Gupta *et al.* [27] developed Team Software Process for Secure Software Development (TSP) specifically for software teams to help them create a high-performance team and prepare their work to produce the best results.
- Flechas *et al.* [28] developed AEGIS (Appropriate and Effective Guidance for Information Security). It first evaluated device assets and their relationships, then moved on to risk analysis, defining weaknesses, threats, and risks.
- Subedi *et al.* [29] present a security paradigm that extends security development practices in agile methodology to overcome this problem in web application development.
- Sodiya [30] developed the Secure Software Development Model (SSDM), which provides training to stakeholders in software development with adequate security education.
- Al-Matouq *et al.* [15] designed a framework Secure Software Design Maturity Model (SSDMM), and the results show that SSDMM helps measure the maturity level of software development organizations.

C. SOFTWARE MATURITY MODELS

Different models for assessing the maturity of software products have been created and implemented:

- The Software Engineering Institute (SEI) at Carnegie Mellon University developed the Capability Maturity Model Integration (CMMI) [31] process model, which assists companies measure and improving their development processes while also delivering high-quality products.
- Alshayeb *et al.* [32], [33], proposed a framework to evaluate the maturity of software products called Technical-CMMI (T-CMMI).
- Eckert *et al.* [34] developed a model to measure the maturity level of Inner Source implementation, which is the process of adopting open-source software development practices for the internal development activities of an organization.
- Al-Qutaish and Abran [35] proposed the Software Product Quality Maturity Model (SPQMM), which measures the quality of a software product.
- The EuroScope consortium [36] developed a model to assess software product quality called the SCOPE Maturity Model (SMM).
- April *et al.* [37] proposed the Software Maintenance Maturity Model (SMmm), based on the CMMI, to assess and improve the quality of software maintenance activities.
- Da Silva and de Barros [38] presented an information security maturity model for software developers based on ISO 27001; it was evaluated by subject experts and utilized to measure the maturity level of several organizations.

D. SECURITY IN SDLC PHASES

This section presents some well-known secure software modeling processes that include security in software development phases:

- S. R. Ahmed [39] identified security activities that should be performed to build secure software and has shown how the security activities are related to usual activities in different phases of software development.
- Essafi *et al.* [40] developed the Secure Software Development Process Model (S2D-ProM), a strategy-oriented process model that offers guidance and support to developers and software engineers, from beginners to experts, to build secure software.
- Niazi *et al.* [41] conducted a systematic literature review (SLR) to pinpoint the required practices for developing secure software and identifying best requirement practices; a framework for secure requirement engineering named Requirements Engineering Security Maturity Model (RESMM) was developed.
- Manico [42], designed the Comprehensive, Lightweight Application Security Process (CLASP), which consists of 24 high-level security activities that can be entirely or partially integrated into software during the SDLC.
- The Building Security In Maturity Model (BSIMM) [43] quantifies numerous businesses' security activities and provides a common foundation for comparing their security endeavours. There are 119 activities in the BSIMM 10 software security framework. These activities are divided into twelve practices. Each practice's exercises are divided into three maturity levels.
- The Open Web Application Security Project (OWASP) created the Software Assurance Maturity Paradigm (SAMM) [42], which is a non-commercial model and is an open platform that aids software companies in developing and implementing software security policies. It contains resources that can help a business assess its software security practices, develop a balanced software security assurance program, and show improvement programs.

E. SUMMARY

Recent software security research has concentrated on offering technical solutions for security concerns; however, a limited study examines security policies and processes in the SDLC's various phases. Little research has been conducted to increase organizational maturity and assurance in secure software development.

Despite the importance of identifying software security risks and practices for addressing these risks in all the phases of SDLC, there is no maturity, readiness, assurance, or secure model for GSD vendor organizations that has been proposed. Furthermore, multiple methodologies and strategies have been established by various researchers; however, industry developers have not adopted them. Existing industry maturity, readiness, and assurance frameworks do not consider the security procedures and techniques described in the literature.

GSD vendor organizations need to be more aware of the security risks during the software development process. They need to read up on the literature and follow industry best practices. This will help that the project goes well and is secure.

III. RESEARCH METHODOLOGY

We used a seven-step methodology to meet the study's goals; we also gathered information from the academic community and the software business, as depicted in Figure 1.

A. SYSTEMATIC MAPPING STUDY (SMS)

The acquired data is more reliable because of this evidence-based methodology. The following sections go over each of these steps in-depth:

SMS is a literature study that focuses on selecting and combining all high-quality research related to a specific topic and provides a complete summary of current texts applicable to specific map queries [44]. Additionally, it finds gaps in existing subject areas and forecasts future research trends [45]–[47].

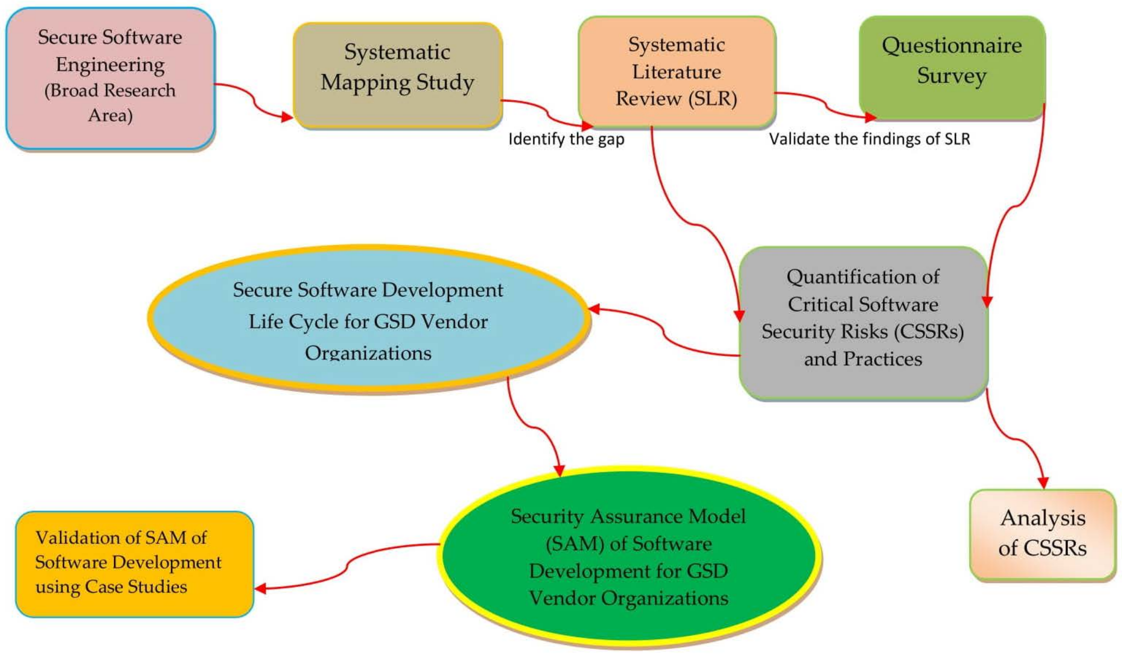

Secure Software Engineering (Broad Research Area)
Systematic Mapping Study
Identify the gap
Systematic Literature Review (SLR)
Validate the findings of SLR
Questionnaire Survey
Quantification of Critical Software Security Risks (CSSRs) and Practices
Secure Software Development Life Cycle for GSD Vendor Organizations
Analysis of CSSRs
Security Assurance Model (SAM) of Software Development for GSD Vendor Organizations
Validation of SAM of Software Development using Case Studies

FIGURE 1. Research methodology.

The primary purpose of the first step (conducting SMS) of this study is to know what is the state-of-the-art in secure software engineering (SSE) [22], [23]. The final selection was among 116 studies that met the inclusion and exclusion criteria [22], [23]. The SMS findings were categorized based on the quality assessment, software security processes/models/frameworks/methods, security stages of the SDLC and the publication venue, and SWOT analysis of the software security approaches [22], [23].

After reviewing the studies, we discovered 37 SSE paradigms, frameworks, and models [22], [23]. The findings indicate that the following SSE frameworks/models are the most frequently employed: ''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)'' and ''Secure Tropos Methodology'' [22], [23].

This study's findings suggest that this field is still in its infancy. It is necessary to conduct sufficient research, mainly focusing on empirically validated solutions, to identify software security risks and strategies for managing these risks at each stage of the software development lifecycle (SDLC).

B. SYSTEMATIC LITERATURE REVIEW (SLR)

''An SLR is a secondary study in which primary studies are examined impartially and iteratively to define, interpret, and discuss evidence relevant to the research questions'' [45]. SLR was used in the second phase of this study to classify the selected articles for identifying security risks and practices [48]. A total of 121 papers were selected via the tollgate technique [49] based on the inclusion, exclusion, and quality rating criteria. Khan *et al.* [48] identified 145 security risks and 424 best practices that help software development organizations to manage security throughout the SDLC.

C. QUESTIONNAIRE SURVEY

An online questionnaire survey using Google Docs was created to validate the SLR findings and discover other security risks and their practices. The questionnaire methodology is more effective than other observational methods because it allows for a larger population to be targeted for data gathering [50]–[58].

In the 3rd stage of this study, an online survey method is employed for data collection [59]. The following steps were used in the questionnaire survey:

1) DEVELOPMENT OF QUESTIONNAIRE SURVEY

The questionnaire primarily consists of closed-ended questions designed to extract specific information from experts. There are a few open-ended questions in the questionnaire to remove any other software security-related risks and practices that were not identified by the SLR. We employed a five-point Likert scale to obtain survey participants' observations regarding the software security risks and its practices listed in the closed-ended section, i.e., ''strongly agree, agree, neutral, disagree, and strongly disagree.

2) PILOT OF QUESTIONNAIRE SURVEY

To conduct the pilot assessment of the questionnaire survey, we chose experts working in the GSD environment (i.e. ''Software Engineering Research Group (SERG UOM) Pakistan'', ''King Fahd University of Petroleum and Minerals, Saudi Arabia'', and ''Qatar University, Doha, Qatar.'').

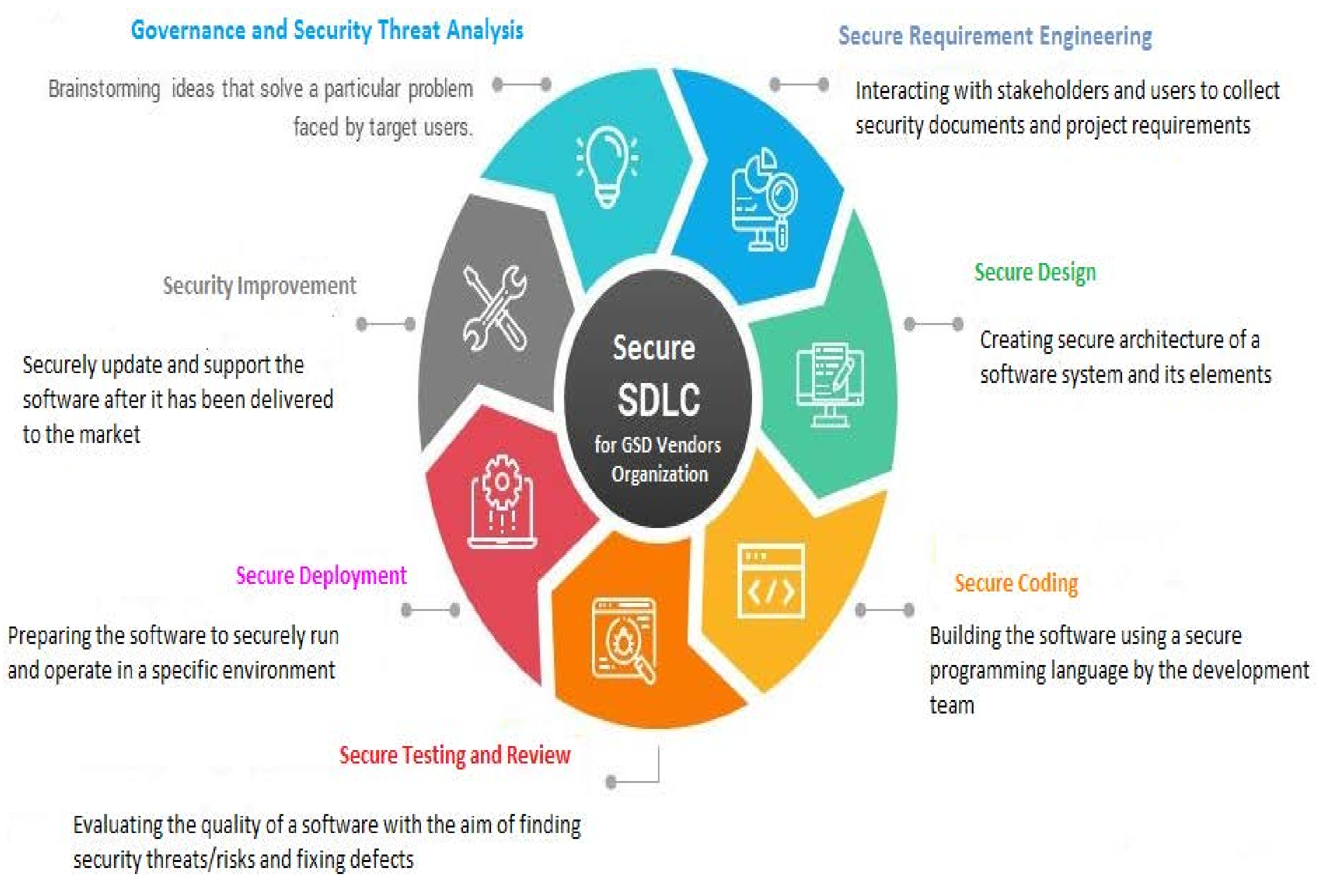

Governance and Security Threat Analysis
Brainstorming ideas that solve a particular problem faced by target users.
Secure Requirement Engineering
Interacting with stakeholders and users to collect security documents and project requirements
Secure Design
Creating secure architecture of a software system and its elements
Secure Coding
Building the software using a secure programming language by the development team
Secure Testing and Review
Evaluating the quality of a software with the aim of finding security threats/risks and fixing defects
Secure Deployment
Preparing the software to securely run and operate in a specific environment
Security Improvement
Securely update and support the software after it has been delivered to the market
Secure SDLC for GSD Vendors Organization

FIGURE 2. **Secure SDLC for GSD vendor organizations.**

This pilot assessment aims to address significant issues (in terms of statistical variables) and improve the survey questions' understandability. Experts suggest improving the questionnaire's design, such as adding questions to obtain more information about survey participants. The questionnaire survey was revised after taking into account the ideas and recommendations of the experts.

At the beginning of the survey, a statement on the researchers' ethical responsibility was also added to assure the participant's confidentiality. This remark reassured the participants that only the study team would access their information. It was stated that the research team would not share the data with anyone to reveal the identity of any participant or organization.

3) DATA COLLECTION SOURCES

As previously indicated, our target population was large and spread organizations across the globe. We decided to use unusual methods to collect responses from SSD professionals working in GSD. We used the snowball sampling technique to gather data from the experts [50]. Snowballing is a low-cost and straightforward strategy to reach a specific audience [50], [54], [60]. We used social media networks such as Facebook, LinkedIn, Research Gate, and email to contact the experts. The empirical study's data was collected online from June 01, 2021, to July 04, 2021, and the entire data gathering process took one month and four days. During the survey's implementation, 64 responses were collected.

All of the responses were manually reviewed. We excluded 14 responses because the expertise shared by these 14 persons was unrelated to GSD and SSD. For analysis, the final 50 survey results were taken into account. We ensure survey participants that the obtained data will only be used for research purposes and that their identities will never be disclosed to a third party.

For analysis, the final 50 survey results were taken into account. This study empirically validated 145 software security risks and 424 security practices that assist GSD vendor organizations in managing the security activities in the SDLC phases.

4) DATA ANALYSIS

The frequency analysis method was used to examine survey participant replies in this study. This approach works well for analyzing nominal and ordinal data over many variables or groups of variables [61]. Because the survey responses are nominal, we employed the chi-square ("liner-by-linear connection") technique to discover significant differences across the variables. Various research with similar data types has used the same analysis approach [60], [62].

TABLE 1. **CSSRs and its practices in the governance and security threat analysis phase of the secure SDLC.**

Secure SDLC Phase	Critical Software Security Risks (CSSRs)	Practices for CSSRs
Governance and Security Threats Analysis	**CSSR1:** Lack of security document checklist	**R1P1:** Make use of checklists to analyze security requirements **R1P2:** Develop security guidelines (collection of practices, checklists, code style, specification, security function, etc.) **R1P3:** Check that the identification security requirement meets your standard **R1P4:** Check the documentation against the security requirements documentation acceptance test parameters **R1P5:** Check that authentication, authorization, immunity, privacy, integrity, non-repudiation, intrusion detection, and system maintenance security requirement meets your standard **R1P6:** Use checklists for secure auditing requirements
	CSSR2: Lack of experience, knowledge, guidance, and security training during security requirements documentation	**R2P1:** All security team members have adequate security training **R2P2:** Develop security guidelines (collection of practices, checklists, code style, security specification, security function, etc.) **R2P3:** Specify security policies, standards, and reference guidelines for security requirements **R2P4:** Identify system stakeholders to improve the knowledge of identification of security requirements **R2P5:** Explain how to use the security document
	CSSR3: Lack of shared understanding of security requirements definitions	**R3P1:** All stakeholders, customers, clients need to be agreed on the security requirements definition **R3P2:** Capture and define non-functional security requirements as attributes of the software **R3P3:** Define the system boundaries regarding privacy and system maintenance security requirements such as sensitive data and communication. **R3P4:** Define specialized security terms **R3P5:** A acceptable security threshold can be defined using a security index. **R3P6:** Specifically define each security requirement **R3P7:** Define policies for change management **R3P8:** Define standard templates for describing authentication, authorization, immunity, privacy, integrity, non-repudiation, intrusion detection, and system maintenance security requirements **R3P9:** Define the system's operation environment to gain survivability security requirements **R3P10:** Define change management policies for authentication, authorization, immunity, privacy, integrity, non-repudiation security, and system maintenance requirements **R3P11:** Define operational processes to gain non-repudiation, integrity, immunity, intrusion detection, and security requirements

D. QUANTIFICATION OF SECURITY RISKS INTO CRITICAL SOFTWARE SECURITY RISKS (CSSRs) USING FREQUENCY AND RANKING ORDER

To identify the critical software security risks (CSSRs), we used the criterion of 10% occurrences in both SLR and survey findings. Other researchers have used a similar approach [62]–[64]. However, secure software development professionals and researchers might set their criteria for determining the importance of the stated software security risk. Based on this criterion, we identified 46 critical software security risks in the SDLC phases. The CSSRs and practices for addressing these risks are presented in Section IV. We also analyzed these CSSRs on GSD Experts' views concerning the impact of software security risks on secure software development (SSD) based on survey respondents' location, GSD vendor organization size, and respondents' experience level in SSD projects *(Accepted for publication in Security in Communication Journal).*

E. CASE STUDY

A case study is a suitable research methodology and an effective validation tool in Software Engineering [65], [66]. In the last stage of this research, we have conducted three case studies in software development companies to validate our projected model, "Security Assurance Model (SAM) of Software Development," for GSD vendor organizations. We conducted focus groups with case study applicants to get feedback on our newly engineered model.

IV. SECURE SDLC FOR GSD VENDOR ORGANIZATIONS

The CSSRs and their practices collected via the SLR and questionnaire survey were categorized into different SDLC phases and used to develop the secure SDLC for GSD vendor organizations. The structure of this SDLC is shown in Figure 2.

It consists of seven stages: Governance and Security Threat Analysis, Secure Requirement Analysis, Secure Design, Secure Coding, Secure Testing and Review, Secure Deployment, and Security Improvement. Each of these phases contains CSSRs and practices for addressing these risks. The following sections briefly describe each stage:

A. GOVERNANCE AND SECURITY THREAT ANALYSIS

To give strategic direction, ensure that objectives are met, ensure risk management is acceptable, and check resource usage is responsible, security governance is the collection of roles and procedures exercised by senior management. Effective security governance will allow your firm to efficiently coordinate its security efforts when it is carried out correctly.

Table 1 presents the CSSRs and their practices in the "Governance and Security Threat Analysis" phase of the secure SDLC for GSD vendor organizations [14], [17], [23],

TABLE 2. CSSRs and its practices in the secure requirements engineering phase of the secure SDLC.

Secure SDLC Phase	Critical Software Security Risks (CSSRs)	Practices for CSSRs
Secure Requirements Engineering	**CSSR4:** Security requirements are often neglected or considered non-functional requirements	**R4P1:** Identify functional and non-functional security requirements **R4P2:** Identify high-level functional security objectives, requirements **R4P3:** For each security objective, security requirements are identified along with the functional and non-functional requirements **R4P4:** Capture and define non-functional security requirements as attributes of the software **R4P5:** Map all the non-functional security requirements identified with functional requirements **R4P6:** Develop security guidelines (collection of practices, checklists, code style, specification, security function, etc.)
	CSSR5: Lack of security requirements, review, assessment, analysis, verification, validation	**R5P1:** Review documentation against the objectives and needs **R5P2:** Check the documentation against the security requirements documentation acceptance test parameters **R5P3:** Perform Secure Requirements Review **R5P4:** Software products should be certified according to security requirements **R5P5:** Validate that software artefacts and processes no longer bear the unacceptable risk **R5P6:** Identification of attackers' interest and capabilities in the resources/assets of a piece of software **R5P7:** A threshold of acceptable security can be defined by using the security index **R5P8:** Identify validation checklists **R5P9:** Specify low-level security requirements to remove security errors **R5P10:** Use multi-disciplinary teams to assess security requirements
	CSSR6: Lack of assessment of physical protection, survivability, and secure auditing requirement risks	**R6P1:** Assess physical protection, survivability, and secure auditing requirement risks **R6P2:** Be sensitive to organizational and political considerations in gaining physical protection of security requirements **R6P3:** Use checklists for secure auditing requirements **R6P4:** Define the system's operation environment to gain survivability security requirements **R6P5:** Institute accountability for security issues **R6P6:** Assess system feasibility in terms of survivability security requirements
	CSSR7: Improper plan for Security Requirements Analysis and negotiation	**R7P1:** The primary responsibility is to conduct product security risk analysis to identify potential security requirements and constraints early. **R7P2:** Attack trees modeling is one of the techniques suggested to be used for analyzing security risks **R7P3:** Analyze tradeoffs between cost and protection provided by security controls **R7P4:** Identify security issues with STRIDE by classifying attacker goals **R7P5:** Security Risk Assessment: use DREAD model **R7P6:** Perform threat landscaping **R7P7:** Data comprehensiveness **R7P8:** Grouping of Requirements **R7P9:** Write down the misuse cases for each secure requirement identified **R7P10:** Define security of system boundaries **R7P11:** Perform risk analysis to address the security issues in requirements development **R7P12:** Make use of checklists to analyze security requirements **R7P13:** Conflicts Resolution: Consider conflicts and how to resolve them **R7P14:** Sort out security requirements through a multi-dimensional approach **R7P15:** Identify priorities in security requirements **R7P16:** Provide software to support negotiations **R7P17:** Verify the misuse case strength in understanding possible attacks
	CSSR8: Lack of developing threat modeling	**R8P1:** Identify threat origin with the help of threat modeling in the requirement phase **R8P2:** Follow STRIDE Threat Model for developing threat modeling **R8P3:** Follow DREAD Threat Model for developing threat modeling **R8P4:** Analyze the threats faced at the time of requirements development **R8P5:** Illustrate threat landscaping, risk likelihood, and mitigation strategy
	CSSR9: Lack of security requirements elicitation activity	**R9P1:** Elicit and categorize safety and security requirements **R9P2:** Take into consideration organizational and political issues **R9P3:** Use scenarios to elicit sensitive data and communication in terms of authentication, authorization, privacy, system maintenance, security requirements **R9P4:** Identify stakeholders **R9P5:** Identify the operating environment of the system **R9P6:** Use concerns related to business to motivate security requirements elicitation **R9P7:** Identify information assets **R9P8:** Identify functional and non-functional security requirements **R9P9:** Search for domain constraints **R9P10:** Record rationale for security requirements **R9P11:** Gather security requirements from various views **R9P12:** Use hypothetical cases to elicit security requirements **R9P13:** Remove any ambiguous requirements **R9P14:** Identify operational process **R9P15:** Reuse security requirements **R9P16:** Determine and consult stakeholders of the system **R9P17:** Record security requirements sources **R9P18:** Assess system security feasibility

TABLE 2. ***(Continued.)*** **CSSRs and its practices in the secure requirements engineering phase of the secure SDLC.**

	CSSR10: Improper security requirements identification and inception	**R10P1:** All stakeholders, customers, clients need to be agreed on the requirements definition **R10P2:** Illustrate the security needs with different perspectives, analyze them, priorities and then specify **R10P3:** Identify high-level functional security objectives, requirements **R10P4:** Identification of security goals **R10P5:** Utilize brainstorming technique to aggregate identification security requirements **R10P6:** Identification of potential attackers of the software **R10P7:** Check that the identification security requirement meets your standard **R10P8:** Set forth the security objectives to address the needs identified **R10P9:** For each security objective, security requirements are identified along with the functional and non-functional requirements **R10P10:** Capture and define non-functional security requirements as attributes of the software **R10P11:** Identify system stakeholders to improve identification of security requirements
	CSSR11: Lack of secure requirements documentation	**R11P1:** Incorporate security needs, objectives, and requirements in the final documentation **R11P2:** Explain how to use the security document **R11P3:** Specify security policies, standards, and reference guidelines for security requirements **R11P4:** Make a business case for the system concerning security **R11P5:** Define specialized security terms **R11P6:** Help readers find information **R11P7:** Make the document easy to change **R11P8:** Illustrate threat landscaping, risk likelihood, and mitigation strategy **R11P9:** Include a summary of the security requirements
	CSSR12: Lack of security requirements prioritization, management, and categorization	**R12P1:** Incorporate security needs, objectives, and requirements in the final documentation **R12P2:** Explain how to use the security document **R12P3:** Specify security policies, standards, and reference guidelines for security requirements **R12P4:** Make a business case for the system concerning security **R12P5:** Define specialized security terms **R12P6:** Help readers find information **R12P7:** Make the document easy to change **R12P8:** Illustrate threat landscaping, risk likelihood, and mitigation strategy **R12P9:** Include a summary of the security requirements **R12P10:** Perform Requirements Specification **R12P11:** Identify policies for management of security requirements **R12P12:** Specifically define each security requirement **R12P13:** Risk mitigation should be conducted in a coherent and a cost-effective manner **R12P14:** Governance: Practice that helps organize, manage, and measure a software security initiative **R12P15:** Evaluate and manage product security risks throughout the project **R12P16:** Risk ranking to prioritize and determine the risks that should be avoided **R12P17:** Define and maintain traceability manual **R12P18:** Establish and manage the project's secure development process **R12P19:** Preservation of confidentiality, Integrity, Availability, Usability should be specified to mitigate identified threats **R12P20:** Identify viewpoint **R12P21:** Define policies for change management **R12P22:** Identify global system security requirements **R12P23:** Asset Rating **R12P24:** Risk estimation **R12P25:** Identify volatile security requirements **R12P26:** Record rejected security requirements **R12P27:** Vulnerability measurement **R12P28:** Perform Requirements Elaboration **R12P29:** Threat evaluation & prioritization
	CSSR13: Improper plan for secure requirements authentication, authorization, and privacy	**R13P1:** Plan for conflicts and conflict resolution for authentication, authorization, immunity security, non-repudiation, and system maintenance requirements in terms of multiple accounts **R13P2:** Define standard templates for describing authentication, authorization, immunity, privacy, integrity, non-repudiation, intrusion detection, and system maintenance security requirements **R13P3:** Use concise and straightforward language to explain authentication, authorization, immunity, privacy, integrity, non-repudiation, intrusion detection, and system maintenance security requirements **R13P4:** Check that authentication, authorization, immunity, privacy, integrity, non-repudiation, intrusion detection, and system maintenance security requirement meets your standard **R13P5:** Define change management policies for authentication, authorization, immunity, privacy, integrity, non-repudiation security, and system maintenance requirements **R13P6:** Use interaction matrices to find conflicts and overlaps in terms of intrusion detection security requirements **R13P7:** Define the system boundaries in terms of privacy and system maintenance security requirements such as sensitive data and communication **R13P8:** Define operational processes to gain non-repudiation, integrity, immunity, intrusion detection, and security requirements

TABLE 3. CSSRs and its practices in the secure design phase of the secure SDLC.

Secure SDLC Phase	Critical Software Security Risks (CSSRs)	Practices for CSSRs
Secure Design	**CSSR14:** Lack of developing threat modeling during the design phase	**R14P1:** Analyze and Minimize Attack Surface **R14P2:** Enumerate threats and prioritize the threat based on the potential impact **R14P3:** Verify whether the threat is mitigated with a security control **R14P4:** Identify areas that could be of interest to attackers **R14P5:** There could be multiple design decisions to mitigate any threat **R14P6:** Secure design decisions to remove threats can be prioritized based on a cost/benefit analysis **R14P7:** Secure design decisions must be identified for threats that violate any of the high-level security requirements. **R14P8:** Risk analysis should be performed on the identified threats to calculate the potential damage
	CSSR15: Lack of attention to following security design principles	**R15P1:** Implement Least Privilege **R15P2:** Implement a defence-in-depth policy, which includes multilevel security **R15P3:** Keep your design as simple as you can by applying the economy of mechanism policy **R15P4:** Correctness by Construction (CbyC) **R15P5:** Fail Securely: The system does not disclose any data that should not be disclosed ordinarily at system failure **R15P6:** Apply false-safe default principles to make sure that the failure of any activity will prevent unsafe operation **R15P7:** Separation of Privilege **R15P8:** Reluctance to Trust **R15P9:** Use a Positive Security **R15P10:** Establish Secure Defaults **R15P11:** Never assumes that your secrets are safe **R15P12:** Securing the Weakest Link **R15P13:** Proactive, not Reactive **R15P14:** Privacy as the Default **R15P15:** Privacy Embedded into Design **R15P16:** Full Functionality **R15P17:** End-to-End Security **R15P18:** Visibility, Usability, and Transparency **R15P19:** Intrusion Detect System (IDS): An IDS is a monitoring system that detects suspicious activities and generates alerts when they are detected. Based upon these alerts, a security operations center (SOC) analyst or incident responder can investigate the issue and take the appropriate actions to remediate the threat. **R15P20:** Follow the psychological acceptability principle of design to incorporate basic security automatically **R15P21:** Follow the least common mechanism to restrict shared resource access **R15P22:** Respect for User Privacy
	CSSR16: Lack of security design awareness, guidance, and training	**R16P1:** All security team members have adequate security training in secure design **R16P2:** Develop security design awareness guidelines (collection of practices, checklists, code style, security specification, security function, etc.) **R16P3:** Specify security policies, standards, and reference guidelines for security design awareness **R16P4:** Identify system stakeholders to improve the knowledge of identification of security design
	CSSR17: Improper secure design documentation	**R17P1:** Develop a Test plan **R17P2:** Document each identified threat along with its description, risk, defensive technique, and risk management strategy **R17P3:** Secure document design **R17P4:** Remove unimportant features **R17P5:** Identify design attributes **R17P6:** Use security diagram classes **R17P7:** Remember that hiding secrets are hard **R17P8:** Avoiding logs from external data **R17P9:** Map security requirements with cryptographic services (Authentication, Confidentiality, Integrity, and Non-Repudiation) **R17P10:** Identify environmental and device security constraints **R17P11:** A threshold of acceptable security can be defined by using a security index. **R17P12:** Perform cost/benefit analysis (CBA) & Security planning (based on risks & CBA)
	CSSR18: Lack of building and maintaining abuse case models and attack patterns	**R18P1:** Write down the misuse cases for each secure requirements **R18P2:** Verify the misuse case strength in understanding possible attacks **R18P3:** Make the mapping explicit, identify the use cases adapted to misuse cases
	CSSR19: Improper security design review and its verification	**R19P1:** Revise or review design implementation **R19P2:** External examination of the design

TABLE 3. ***(Continued.)*** **CSSRs and its practices in the secure design phase of the secure SDLC.**

		R19P3: Establish secure design requirements **R19P4:** Plan and implement secure supplier and third-party component selection **R19P5:** The design must be inspected (multiple times if required) to identify and remove software errors **R19P6:** Remember that backward compatibility will always give your grief **R19P7:** The expert also needs to verify the interface and mediator between product management and development
	CSSR20: Lack of developing data flow diagram	**R20P1:** Establish security improvement goals **R20P2:** Map out the process Value Stream. **R20P3:** Collect time and waste data. **R20P4:** Conduct Kaizen event (5 days). Understand the current work processes (Sub Process Map and & Going to the Gemba) Identified, analyzed wastes, and prioritized. improvement focus **R20P5:** Install solutions.
	CSSR21: Lack of establishing security design requirements	**R21P1:** Establish secure design requirements **R21P2:** Secure design decisions must be identified for threats that violate any of the high-level security requirements **R21P3:** Map security requirements with cryptographic services (Authentication, Confidentiality, Integrity, and Non-Repudiation)
	CSSR22: Improper restriction to share resource access	**R22P1:** Develop complex Encryption methods **R22P2:** Perform security certification and accreditation of target system **R22P3:** All legitimate users must have the privileges and minimum access needed **R22P4:** Choose a proper and hard to guess location for temporary files and apply an access control mechanism **R22P5:** Provide data protection services **R22P6:** Mask the problem by applying filters to either block or modify user input **R22P7:** Use Logging and Tracing
	CSSR23: Lack of implementation of security design decisions: (Cryptographic protocols, standards, services, frameworks, and mechanisms	**R23P1:** Implement security design decisions: (Security Cryptographic protocols, standards, services, and mechanisms) **R23P2:** Use of security patterns **R23P3:** Guide those involved in designing or enhancing security design decisions
	CSSR24: Lack of defence in depth	**R24P1:** Identify security threats, Characterize risks and Maintain asset inventory **R24P2:** Implement standard security recommendations, policies, and procedures **R24P3:** Implement physical security (field electronics locked down, control center access controls, remote site video, access controls, and remove barriers) **R24P4:** Implement security monitoring (intrusion detection systems, security audit logging, security incident, and event monitoring)

[41], [67]–[72]. In the following tables, R1P1: Means Practice 1 for CSSR1, R1P2: Means Practice 2 for CSSR1, and so on…

B. SECURE REQUIREMENTS ENGINEERING

Software development begins with a phase known as requirements analysis. During the requirements phase, the goal is to define and communicate software requirements specific to the customer's needs. Additionally, security requirements must be defined in addition to software requirements. Security requirements arise from a variety of places and at various periods. Security needs are more complicated to identify than functional requirements since they are not as visible as functional requirements.

Table 2 presents the CSSRs and their practices in the "Secure Requirement Engineering" phase of the secure SDLC for GSD vendor organizations [14], [17], [23], [41], [67]–[72].

C. SECURE DESIGN

Creating a software architecture is the first step in the design process. The critical components of software and their interactions are identified by software architecture. Software design documents are then used to explain the security architecture in further depth.

Security design is constantly revised in light of this information to incorporate security. Table 3 depicts the security operations during the design [8], [17], [22], [72]–[79].

D. SECURE CODING

The implementation phase serves a dual purpose in terms of security. The first task is to prevent software security issues by writing secure code. The second step is to look for software flaws using an automated security analysis tool. Automated tools are used to examine the source code created by the development team. These tools are run by computers and look for common security flaws. Manual reviews follow the automatic static analysis. Finally, the software is ready for testing. Table 4 presents the CSSRs and their practices in the "Secure Coding" phase of the secure SDLC for GSD vendor organizations [3], [8], [17], [72], [80]–[83].

E. SECURE TESTING AND REVIEW

Test planning is the first step in the testing phase. The testing team begins planning tests during the implementation phase because test planning does not necessitate access to

TABLE 4. CSSRs and its practices in the secure coding phase of the secure SDLC.

Secure SDLC Phase	Critical Software Security Risks (CSSRs)	Practices for CSSRs
Secure Coding	**CSSR25:** Tampering: is the unauthorized modification of data	**R25P1:**Acquisition of log **R25P2:**Appropriate authorization **R25P3:**Apply Hashes, message authentication codes **R25P4:**Incorporate Digital Signatures **R25P5:**Communication connections between system components must be ensured using protocols that provide confidentially **R25P6:**Recording of user ID, Date, type of operation, name of AP at time of operation execution
	CSSR26: SQL Injection	**R26P1:** Use of parameterized queries or stored procedures (OWASP) **R26P2:** SQL injection prevention cheat sheet **R26P3:** Input sanitization: User inputs are sanitized to ensure that they contain no dangerous code **R26P4:** 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 needed. **R26P5:** Disabling literals. SQL injection can be avoided if the database engine supports disabling literals, where text and number literals are not allowed as part of SQL statements. **R26P6:** Avoid string concatenation for dynamic SQL statements **R26P7:** Check the query if they exist in the query pool. Only they are permitted **R26P8:** You can replace the single quote with double-quotes. This blocks the SQL insertion attack **R26P9:** Use of prepared Statement and "ORM"
	CSSR27: Cross-Site Scripting, cross-site request forgery	**R27P1:** Design libraries and templates that minimize unfiltered input (OWASP XSS Prevention Cheat Sheet) **R27P2:** DNS rebinding **R27P3:** Using ESAPI safety mechanism can eliminate detected XSS vulnerabilities in web application **R27P4:** HTML Purifier eliminates XSS vulnerabilities **R27P5:** Check input with validation function in CakePHP **R27P6:** Use a cryptographic token to associate a request with specific action. The token can be regenerated at every request. (OWASP CSRF Prevention Cheat Sheet; encrypted token) **R27P7:** Confirm action every time potentially sensitive data is invoked **R27P8:** Use of the optional HTTP Referrer header **R27P9:** Normalize all inputs, including those not expected to have any scripting content. **R27P10:** "Sanitizing" is one way to prevent XSS attacks **R27P11:** Use of h() function and security component in CakePHP
	CSSR28: Denial of Services: is the process of making a system or application unavailable	**R28P1:**Restrict number of accesses per hour **R28P2:**Appropriate authentication and authorization **R28P3:**Appropriate filtering, throttling, quality of service
	CSSR29: Repudiation: is the ability of users (legitimate or otherwise) to deny that they performed specific actions or transactions	**R29P1:** Every activity related to essential and sensitive data must be recorded **R29P2:** Incorporate Digital Signatures, timestamps, audit trails **R29P3:** Ensure that the sender of a message does not deny having sent the message, and the receiver does not deny receiving the message
	CSSR30: Information Disclosure: is the unwanted exposure of private data	**R30P1:** Ensure that only selected accounts can access essential data **R30P2:** Implement an encryption mechanism and protect secrets
	CSSR31: Elevation of privilege: occurs when a user with limited privileges assumes the identity of a privileged user	**R31P1:** Physical security techniques, such as locking doors, alarms, and monitoring targets, should be implemented **R31P2:** Hashing of password (OWASP password storage cheat sheet) **R31P3:** Confidentiality: Protect data or services from unauthorized access **R31P4:** Integrity: Avoid unauthorized manipulation of data or services **R31P5:** Availability: Assure that the system works promptly and services are not denied to an authorized user **R31P6:** Authentication: Identify the actors involved in a transaction and verify that they are who claim to be **R31P7:** Use a firewall, VPN, and SSL techniques **R31P8:** Hashing of password using "Auth" component in CakePHP
	CSSR32: Spoofing: An attempt to gain access to a system by using a fake identity	**R32P1:** Declaration of accessible IP addresses by using .htaccess file **R32P2:** Customers must use a strong password or use a multi-login mechanism **R32P3:** Restriction of access **R32P4:** Do not store secrets in plain text **R32P5:** Protect secret data
	CSSR33: Password Conjecture: Lack of password complexity enforcement	**R33P1:** Delete all default account credentials that product vendor may put in **R33P2:** Strengthen the password **R33P3:** Implementation of password throttling mechanism (OWASP Authentication Cheat Sheet-prevent brute force attack) **R33P4:** Implement the Account lockout procedure **R33P5:** Establishment of a firm password policy and check of its observance (OWASP

TABLE 4. *(Continued.)* **CSSRs and its practices in the secure coding phase of the secure SDLC.**

		Authentication Cheat Sheet-password complexity, use multi-factor authentication) **R33P6:** Implement count number of login trial and set a flag that shows account lock to "True" if the number exceeds the threshold **R33P7:** Enhancement of specification regarding password **R33P8:** Checking of input with validation function in CakePHP **R33P9:** Use of solid input validation (OWASP SQL injection prevention cheat sheet) **R33P10:** User of error handler in CakePHP (core.php) **R33P11:** Recycle password
	CSSR34: Buffer and Array Overflow	**R34P1:** Choosing a type-safe programming language can avoid many buffer overflow problems. **R34P2:** Correct use of safe libraries of an unsafe language can prevent most buffer overflow vulnerabilities. **R34P3:** Normalize strings before validating them **R34P4:** Executable space protection can prevent code execution on the stack or the heap. **R34P5:** Stack-smashing protection can detect the most common buffer overflow by checking that the stack has not been altered when a function returns. **R34P6:** Deep packet inspection attempts to block packets that have the signature of a known attack or have a long series of no-operation instruction
	CSSR35: Weak encryption, insecure communication	**R35P1:** Encryption of communication using cryptography (Mozilla guideline-secure transmission) **R35P2:** Acquisition of public key signed by Certificate Authority **R35P3:** Intrusion detection system **R35P4:** Exchange of public keys using a secure channel **R35P5:** Develop complex Encryption methods **R35P6:** Secure the weakest link **R35P7:** Eliminate weak cryptography
	CSSR36: Messy code, code bad smells, dead code	**R36P1:** Prevent execution of illegitimate code **R36P2:** Remove debugging code and flags in code **R36P3:** Apply Hashes, message authentication codes **R36P4:** Input sanitization: User inputs are sanitized to ensure that they contain no dangerous code **R36P5:** Do not code to send session ID with GET method **R36P6:** Executable space protection can prevent the execution of code on the stack or the heap **R36P7:** Perform Code Review **R36P8:** Implement static code analysis **R36P9:** Apply secure coding standards such as CERT, MISRA, and AUTOSAR **R36P10:** Expert need to ensure that secure coding practices are followed and conducts code analysis to identify security vulnerabilities **R36P11:** Implement dynamic code analysis **R36P12:** Remove debugging code and flags in code **R36P13:** Refactoring can improve the security of an application by removing code bad smell **R36P14:** Code-level hardening is a way that prevents vulnerabilities **R36P15:** Secure code writing **R36P16:** Develop threat modeling: It helps to do threat analysis into **R36P17:** Secure code review **R36P18:** Code must be inspected to identify software and security errors

program code. While the developers are working on code, the testing team prepares for the testing activity later. The security testing teams develop test cases based on design papers, threat models, and misuse situations. Security test cases aim to attack software successfully. Any modification to the software design must be disclosed to the testing team during the planning phase of the test. As a result, the testing team may create test cases that aren't relevant. Based on the nature of the software and its threats, the severity of bugs is adjusted. A minor bug in software code that isn't very important to how it works may only take a few minutes to fix, but it can significantly affect how secure it is. A discussion of the revised severities takes place with the development team. Table 5 shows the security operations conducted during the testing phase [3], [17], [80], [84]–[86].

F. SECURE DEPLOYMENT

Release and deployment are two distinct processes in the project's lifecycle. We still regard security activities as a single phase, even though there isn't much work to be done in this phase. A security assessment is performed before the release of the software. The evaluation's purpose is to find remaining security weaknesses. The study concludes with a review report. The development team fixes the Security flaws found in a review report. A security audit is carried out following the evaluation. Based on the audit report, the management decides on the software release. The software is now available for deployment when it has been released. Security activities are documented in Table 6 [3], [87]–[91] during the release and deployment phases.

G. SECURITY IMPROVEMENT

The software becomes commercial after its release and deployment. Some known security flaws may have been left out while the software was made. The software was released with some noncritical security flaws, but they were not dangerous. In the future, the defects will be fixed at some point. As a result, a patch is created to improve the discovered

TABLE 5. CSSRs and its practices in the secure testing and review phase of the secure SDLC.

Secure SDLC Phase	Critical Software Security Risks (CSSRs)	Practices for CSSRs
Security Testing and Review	**CSSR37:** Lack of Penetration Security Testing Analysis	**R37P1:** Perform Penetration Testing **R37P2:** Validate correct use of security testing tools **R37P3: Planning and reconnaissance** **R37P4: Static analysis** – Inspecting an application's code to estimate how it behaves while running. These tools can scan the entirety of the code in a single pass. **R37P5: Dynamic analysis** – Inspecting an application's code in a running state. This is a more practical way of scanning, as it provides a real-time view of an application's performance. **R37P6: Gaining Access**: This stage uses web application attacks, such as cross-site scripting, SQL injection, and backdoors**,** to uncover a target's vulnerabilities. **R37P7: Maintaining access**: The goal of this stage is to see if the vulnerability can be used to achieve a persistent presence in the exploited system— long enough for a bad actor to gain in-depth access **R37P8: Risk analysis and reporting**: The results of the penetration test are then compiled into a report detailing
	CSSR38: Lack of Static and Dynamic Security Testing Analysis	**R38P1:** Use advanced Security Testing Tools, such as Fortify SCA, Checkmark code analysis, HP Web Inspect, Acunetix wen, IBM AppScan **R38P2:**Finalize the tool. Select a static analysis tool that can perform code reviews of applications written in the programming languages you use **R38P3**: Create the scanning infrastructure, and deploy the tool **R38P4:**Customize the tool. Fine-tune the tool to suit the needs of the organization **R38P5:**Prioritize and onboard applications. Once the tool is ready, onboard your applications. If you have a large number of applications, prioritize the high-risk applications to scan first **R38P6:**Analyze scan results **R38P7:**Provide governance and training
	CSSR39: Lack of final security review	**R39P1:** Understand the developers' approach: Before starting a secure code review, talk to the developers and understand their approaches to mechanisms like authentication and data validation. **R39P2:** Use multiple techniques: If possible, use manual and automated techniques for the review because each method will find things that the other doesn't. **R39P3:** Do not assess the level of risk **R39P4:** Focus on the big picture. When performing a manual review, resist trying to understand the details of every line of code **R39P5:** Follow up on review points **R39P6:** Stick to the intent of the review **R39P7:** Test security audit and final review **R39P8:** Review unfixed security bugs **R39P9:** Perform a final security review to find any remaining security flaws **R39P10:** Review the weakness of all such areas about new threats identified
	CSSR40: Lack of Fuzz Testing Analysis	**R40P1:**Rebuild the System from Source **R40P2:**Transform Fuzzing Driver from Unit Tests and Sample Code **R40P3:**Develop Automatic Toolchain to Support Complex Compilation and Build **R40P4:**Shallow Bugs Repair with Developers **R40P5:**Patch the Program and the Fuzzer to Detect Hiding Bugs

security weaknesses. The patch is then tested and released. The same steps are taken for new threats. Table 7 shows the security measures taken throughout the maintenance phase [89], [92], [93].

V. SECURITY ASSURANCE MODEL (SAM) OF SOFTWARE DEVELOPMENT FOR GSD VENDORS

This section discusses building a suggested Software Development Security Assurance Model (SAM) for GSD Vendor Organizations. The structure of SAM of software development and its dimensions, such as security assurance levels and evaluation method, are discussed in the following sections:

A. STRUCTURE OF THE SAM OF SOFTWARE DEVELOPMENT

We have developed SAM of Software Development for GSD Vendors by studying various models frameworks [4], [8], [13], [15], [16], [31], [35], [38], [39], [41], [43], [58], [68], [81], [94] and the results obtained from the SLR [48] and questionnaire survey. The process flow for SAM of software development is depicted in Figure 3.

B. SECURITY ASSURANCE LEVELS OF SAM OF SOFTWARE DEVELOPMENT

A security assurance level includes relevant specific security practices for a specific process area: Governance and Security Threat Analysis, Secure Requirement Engineering, Secure Designing, Secure Coding, Secure Testing and Review, Secure Deployment and Secure Improvement, that can improve the organization's software security processes associated with that process area.

Software development Organizations that want to understand better their software security practices need to compare them with Software Security Assurance (SAM) of Software

TABLE 6. CSSRs and its practices in the secure deployment phase of the secure SDLC.

Secure SDLC Phase	Critical Software Security Risks (CSSRs)	Practices for CSSRs
Secure Deployment	**CSSR41:** Lack of default software configuration	**R41P1:** Perform Security Assessment and Secure Configuration **R41P2:**Configure the monitoring and logging **R41P3:**Analyze the overall state of the software **R41P4:**Identify security breaks **R41P5:**Ensure that work products meet their specified security requirements **R41P6:**Document Technical Stack: Document the components used to build, test, deploy, and operate the software
	CSSR42: Logout incorrectly implemented	**R42P1:** Configure the monitoring and logging **R42P2:** Prevent the browsers from caching **R42P3:** The user must be able to log out of a single service (local logout) **R42P4:** Fixing and avoiding obvious bugs **R42P5:** Enhanced cookie management in browsers to enable group deletion, **R42P6:** Unified and standardized process for ending service sessions, **R42P7:** Clearer and uniform user interface for logout and browser-sent keep-alive messages **R42P8:** Polling mechanisms to check for the IP session
	CSSR43: Improperly enabled services and ports	**R43P1:**Associate Active Ports, Services, and Protocols to Asset Inventory **R43P2:**Ensure Only Approved Ports, Protocols, and Services Are Running **R43P3:**Perform Regular Automated Port Scans **R43P4:**Apply Host-Based Firewalls or Port-Filtering **R43P5:**Implement Application Firewalls **R43P6:** Identify When a Service is Necessary **R43P7:** Harden Operating Systems **R43P8:** Beware of User-installed Software **R43P9:** Reduce Attack Surface

Development best practices. Implementing SAM practices often starts with an initial Governance and Security Threat Analysis level. Generally, a Software Development business decides to be appraised for one or more reasons, including to:

- Evaluate how the organization's processes compare to SAM of Software Development's best security practices and determine areas of improvement
- Share information with customers or suppliers about how the organization compares to SAM of Software Development's best security practices
- Comply with contractual terms of customers

It's worth noting that while the goal of organizations is to reach level 7 (Security Improvement), the model is still applicable and beneficial for organizations that have achieved this security assurance level. Organizations at this level are primarily focused on maintenance and improvements, and they also have the flexibility to focus on innovation and respond to industry changes.

The CSSRs and practices for addressing these risks are grouped into seven different levels, which we call security assurance levels, e. g,

- **Level-1:** Governance and Security Threat Analysis
- **Level-2:** Secure Requirement Analysis
- **Level-3:** Secure Design
- **Level-4:** Secure Coding
- **Level-5:** Secure Testing and Review
- **Level-6:** Secure Deployment
- **Level-7:** Security Improvement

The security assurance levels of SAM of software development for GSD vendor organizations are illustrated in Figure 4. 46 CSSRs and 388 practices (see Table 1-7) were identified using SLR and questionnaire survey.

C. ASSESSMENT METHOD FOR SAM OF SOFTWARE DEVELOPMENT

The SAM of Software Development assessment process is based on the Motorola evaluation tool [95]. Motorola created this tool to evaluate the present state of a company's software processes and find areas for improvement [95]. Other researchers have widely used it to assess their proposed models' maturity, readiness, or assurance [15], [41], [58], [68], [96].

The following three measurement dimensions are used by the Motorola instrument [95]:

1. Approach: This aspect is concerned with the willingness and ability of the organization to implement a given practice.

2. Deployment: This dimension looks at how well practices are used in different project parts.

3. Results: This indicator measures the success of projects across domains.

Each dimension of practice is evaluated by providing a score value (0, 2, 4, 6, 8, or 10). The given score value relates to a specific dimension's performance level and can be assessed using the criteria listed in Table 8. To assess the practice under specific security assurance level, the following steps should be followed:

- **Step 1:** For each software security practice, compute an average of the three-dimensional scores, then round that result to the nearest whole integer.

TABLE 7. CSSRs and its practices in the security improvement phase of the secure SDLC.

Secure SDLC Phase	Critical Software Security Risks (CSSRs)	Practices for CSSRs
Security Improvement	**CSSR44:** Lack of security trust	**R44P1:** Make sure institutions are effective and deliver real benefits for people. **R44P2:** Develop future leaders who work for the greater good, not for themselves. **R44P3:** Strengthen accountability and transparency. **R44P4:** Engage citizens in solving community and societal challenges. **R44P5:** Strengthen social inclusion. **R44P6:** Establish real commitment. **R44P7:** A high-level view of how security trust will be maintained **R44P8: Identity and access management** — who has access to specific information and how identity is authenticated and authorized **R44P9: Confidentiality and sensitivity** —objective analysis of the confidentiality of particular data sets, applications, and other security trust elements **R44P10: Acceptable use** — the standards that you expect end-users, developers, and other authorized users to abide by
	CSSR45: Lack of proper methods to find out the attack surface for new threats in the system	**R45P1:** Antivirus software is designed to detect, remove and prevent malware infections on a device or network. **R45P2:** To prevent spyware, network administrators should require remote workers to access resources over a network through a virtual private network that includes a security scan component. **R45P3:** Security policy first **R45P4:** Don't neglect physical security **R45P5:** Screen new hires **R45P6:** Use strong authentication **R45P7:** Secure your desktops **R45P8:** Segment LANs **R45P9:** Plug information leaks **R45P10:** Investigate anomalous activities **R45P11:** Refocus perimeter tools and strategies **R45P12:** Monitor for misuse
	CSSR46: Not developing security patches for the threats	**R46P1:** Set clear expectations and hold teams accountable **R46P2:** Work collaboratively with technical teams to ensure a common language **R46P3:** Establish a disaster recovery process **R46P4:** Install a patch for the vulnerability, if available, to fix the issue **R46P5:** Implement compensating controls so the vulnerability is mitigated without being fully patched **R46P6:** Accept the risk posed by that vulnerability and do nothing **R46P7:** Establish asset management **R46P8:** Priorities the vulnerabilities **R46P9:** Remediate the vulnerabilities to reduce the risks **R46P10:** Measure the success of your vulnerability management program **R46P11:** Where possible, create a backup/archive and verify its integrity by deploying it on a standby system **R46P12:** Create a checklist/procedure for patch activities and deploy the patch on the standby system **R46P13:** Test the patched standby system for operational functionality and compatibility with other resident applications. **R46P14:** Swap the patched standby system into production and keep the previous unpatched production system as a standby for emergency patch regression **R46P15:** Closely monitor the patched production system for any issues not identified during testing **R46P16:** Patch the standby system (old production) after confidence is established with the production unit. Update related records and software configuration management plan.

- **Step 2:** Add Step 1 to the number of security practices for a specific critical software security risk (CSSR). Then divide by the total number of practices. This determines the security assurance of the overall score for a particular CSSR.
- **Step 3:** A security assurance score of less than seven is deemed weak, while a score of 7 or greater is regarded as strong.

D. VALIDATION OF THE SAM OF SOFTWARE DEVELOPMENT USING CASE STUDIES

A case study is considered an appropriate research method in software engineering and a highly effective tool for validation purposes [65], [66]. In the last stage of this research, we have conducted three case studies on software development companies to validate our projected model (SAM of Software Development). The SAM of Software Development assessment process is based on the Motorola evaluation tool [95]. To get feedback on our suggested model, we convened focus groups with the participants of the case studies. The key objectives of performing case studies in this study are as follows:

- Present that SAM of Software Development for GSD Vendor Organizations may be employed in a real-world setting.
- Know ease of learning of SAM of Software Development are ease of learning

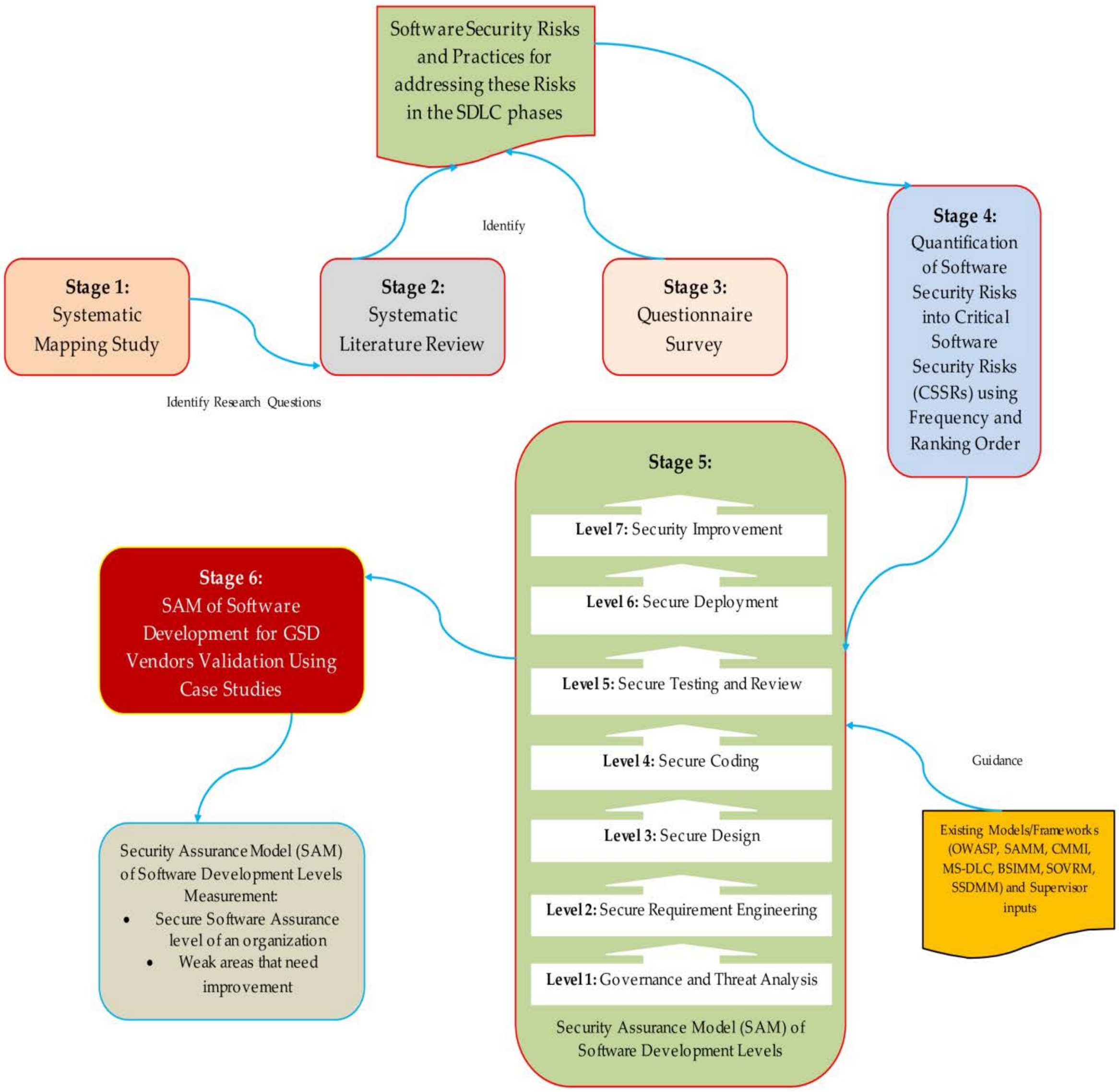

Software Security Risks and Practices for addressing these Risks in the SDLC phases
Identify
Stage 1: Systematic Mapping Study
Identify Research Questions
Stage 2: Systematic Literature Review
Stage 3: Questionnaire Survey
Stage 4: Quantification of Software Security Risks into Critical Software Security Risks (CSSRs) using Frequency and Ranking Order
Stage 5:
Level 7: Security Improvement
Level 6: Secure Deployment
Level 5: Secure Testing and Review
Level 4: Secure Coding
Level 3: Secure Design
Level 2: Secure Requirement Engineering
Level 1: Governance and Threat Analysis
Security Assurance Model (SAM) of Software Development Levels
Guidance
Existing Models/Frameworks (OWASP, SAMM, CMMI, MS-DLC, BSIMM, SOVRM, SSDMM) and Supervisor inputs
Stage 6: SAM of Software Development for GSD Vendors Validation Using Case Studies
Security Assurance Model (SAM) of Software Development Levels Measurement:
Secure Software Assurance level of an organization
Weak areas that need improvement

FIGURE 3. SAM of software development for GSD vendors.

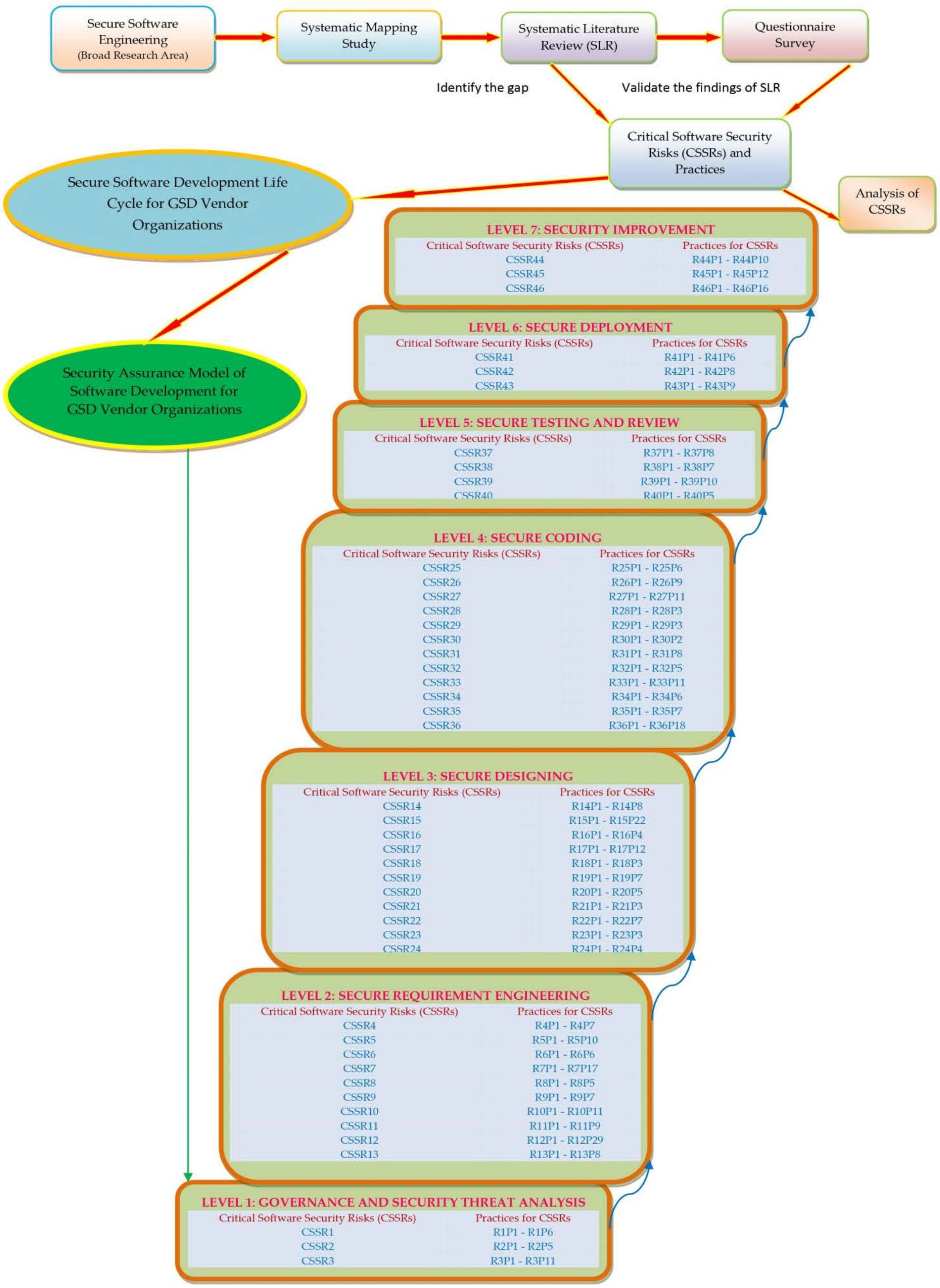
- Know user satisfaction of SAM of Software Development
- Know whether there are any modifications or improvements to the SAM of Software Development that the case study participants suggest.
- To what extent can case study participants improve the SAM of Software Development? Please also include the reasons behind your suggestions.
- Present the applicability and utility of SAM of Software Development.

We interacted with personnel software development companies to perform the case study, explaining the SAM of Software Development concept and inviting them to participate in our research. They were instructed to apply the Motorola assessment tool to analyze their software development security procedures using SAM of Software Development. They completed the examination at their place of business and emailed the results and their feedback.

We chose one large, one medium-sized, and one small-sized company for our case study to reduce the impact of their size.

1) COMPANY A

Iflexion (**https://www.iflexion.com/about**) provides full-cycle services in content management systems, portals, eCommerce, web-based solutions for enterprise and media content distribution, and social software worldwide. We have been providing software development and related IT services since we started in 1999. To deliver high-quality solutions, they combine the expertise of 850+ trained software professionals with tried-and-true techniques, business domain knowledge, and technical competence.


Secure Software Engineering (Broad Research Area)
Systematic Mapping Study
Systematic Literature Review (SLR)
Questionnaire Survey
Identify the gap
Validate the findings of SLR
Critical Software Security Risks (CSSRs) and Practices
Analysis of CSSRs
Secure Software Development Life Cycle for GSD Vendor Organizations
Security Assurance Model of Software Development for GSD Vendor Organizations

LEVEL 7: SECURITY IMPROVEMENT

Critical Software Security Risks (CSSRs)	Practices for CSSRs
CSSR44	R44P1 - R44P10
CSSR45	R45P1 - R45P12
CSSR46	R46P1 - R46P16

LEVEL 6: SECURE DEPLOYMENT

Critical Software Security Risks (CSSRs)	Practices for CSSRs
CSSR41	R41P1 - R41P6
CSSR42	R42P1 - R42P8
CSSR43	R43P1 - R43P9

LEVEL 5: SECURE TESTING AND REVIEW

Critical Software Security Risks (CSSRs)	Practices for CSSRs
CSSR37	R37P1 - R37P8
CSSR38	R38P1 - R38P7
CSSR39	R39P1 - R39P10
CSSR40	R40P1 - R40P5

LEVEL 4: SECURE CODING

Critical Software Security Risks (CSSRs)	Practices for CSSRs
CSSR25	R25P1 - R25P6
CSSR26	R26P1 - R26P9
CSSR27	R27P1 - R27P11
CSSR28	R28P1 - R28P3
CSSR29	R29P1 - R29P3
CSSR30	R30P1 - R30P2
CSSR31	R31P1 - R31P8
CSSR32	R32P1 - R32P5
CSSR33	R33P1 - R33P11
CSSR34	R34P1 - R34P6
CSSR35	R35P1 - R35P7
CSSR36	R36P1 - R36P18

LEVEL 3: SECURE DESIGNING

Critical Software Security Risks (CSSRs)	Practices for CSSRs
CSSR14	R14P1 - R14P8
CSSR15	R15P1 - R15P22
CSSR16	R16P1 - R16P4
CSSR17	R17P1 - R17P12
CSSR18	R18P1 - R18P3
CSSR19	R19P1 - R19P7
CSSR20	R20P1 - R20P5
CSSR21	R21P1 - R21P3
CSSR22	R22P1 - R22P7
CSSR23	R23P1 - R23P3
CSSR24	R24P1 - R24P4

LEVEL 2: SECURE REQUIREMENT ENGINEERING

Critical Software Security Risks (CSSRs)	Practices for CSSRs
CSSR4	R4P1 - R4P7
CSSR5	R5P1 - R5P10
CSSR6	R6P1 - R6P6
CSSR7	R7P1 - R7P17
CSSR8	R8P1 - R8P5
CSSR9	R9P1 - R9P7
CSSR10	R10P1 - R10P11
CSSR11	R11P1 - R11P9
CSSR12	R12P1 - R12P29
CSSR13	R13P1 - R13P8

LEVEL 1: GOVERNANCE AND SECURITY THREAT ANALYSIS

Critical Software Security Risks (CSSRs)	Practices for CSSRs
CSSR1	R1P1 - R1P6
CSSR2	R2P1 - R2P5
CSSR3	R3P1 - R3P11

FIGURE 4. Security assurance levels of SAM of software development for GSD vendors.

TABLE 8. **Motorola assessment tool.**

Motorola Assessment Instrument			
Key Activity Evaluation Dimensions			
Score	Approach (Score Range: 0, 2, 4, 6, 8, 10)	Deployment (Score Range: 0, 2, 4, 6, 8, 10)	Results (Score Range: 0, 2, 4, 6, 8, 10)
Poor (0)	No management recognition of need (OR) No organizational ability (OR) No organizational commitment (OR) Practice not evident	No part of the organization uses the practice (OR) No part of the organization shows interest	Ineffective
Weak (2)	Management begins to recognize used (OR) Support items for the practice start to be created (OR) A few parts of organization are able to implement the practice	Fragmented use (OR) Inconsistent use (OR) Deployed in some parts of the organization (OR) Limited to monitoring/ verification of use	Spotty results (OR) Inconsistent results (OR) Some evidence of effectiveness for some parts of the organization
Fair (4)	Wide but not complete commitment by management (OR) Road map for practice implementation defined (OR) Several supporting items for the practice in place	Less fragmented use (OR) Some consistency in use (OR) Deployed in some major parts of the organization (OR) Monitoring / Verification of use for several parts of the organization	Consistent and positive results for several parts of the organization (OR) Inconsistent results for other parts of the organization
Marginally Qualified (6)	Some management commitment; some management becomes proactive (OR) Practice implementation well under way across parts of the organization (OR) Supporting items in place	Deployed in many parts of the organization (OR) Mostly consistent use across many parts of the organization (OR) Monitoring / Verification of use for many parts of the organization	Positive measurable results in most parts of the organization (OR) Consistently positive results over time across many parts of the organization
Qualified (8)	Total management commitment (OR) Majority of management is proactive (OR) Practice established as an integral part of the process (OR) Supporting items encourage and facilitate the use of practice	Deployed in almost all parts of the organization (OR) Consistent use across almost all parts of the organization (OR) Monitoring / Verification of use for almost all parts of the organization	Positive measurable results in almost all parts of the organization (OR) Consistency positive results over time across almost all parts of the organization
Outstanding (10)	Management provides zealous leadership and commitment (OR) Organizational excellence in the practice recognized even outside the company	Pervasive and consistent deployed across all parts of the organization (OR) Consistent use over time across all parts of the organization (OR) Monitoring / Verification for all parts of the organization	Requirement exceeded (OR) Consistently world-class results (OR) Counsel sought by others

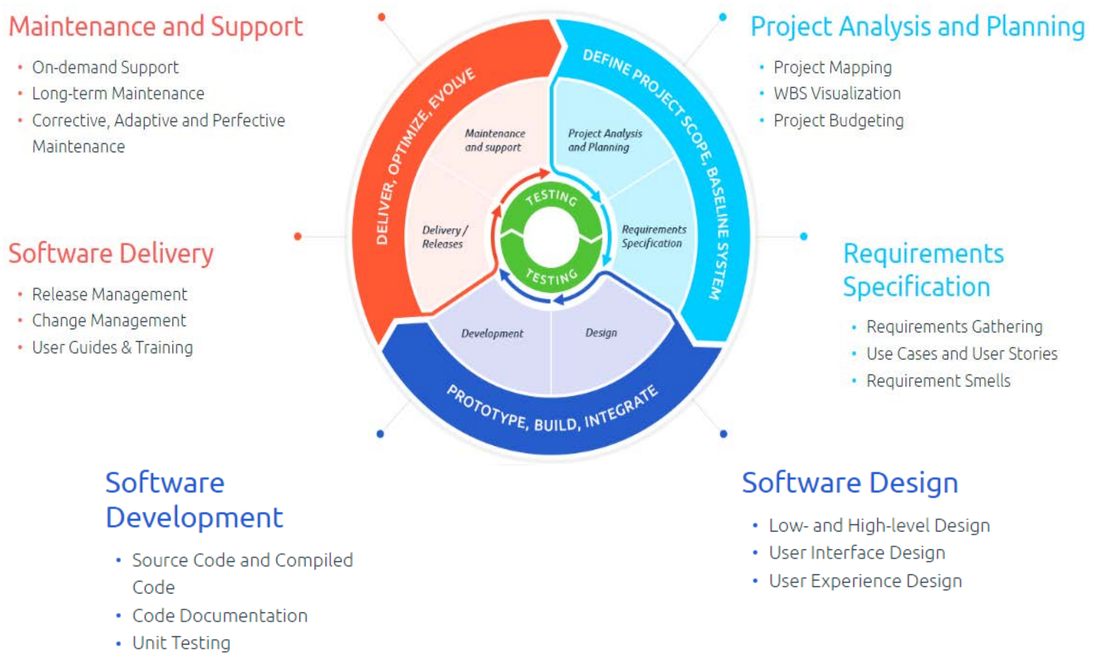


FIGURE 5. Iflexion covers the entire cycle of enterprise software development.

Iflexion US and UK Offices: IFlexion US (Head Office). 3900, S. Wadsworth Blvd., Denver, CO 80235. Iflexion UK. 3rd floor, 5-8 Dysart Street. Their clients and partners are dispersed in 30+ countries. They cover the entire cycle of enterprise software development, as depicted in Figure 5.

2) COMPANY B

Satsuma Droid Pvt Ltd (Islamabad, Pakistan) is dedicated to bringing your organization to the forefront by delivering cutting-edge IT services, Web Solutions, Mobile Solutions, and Digital Marketing Strategies. As a result, they are committed to helping you achieve long-term leadership in your industry by developing digital solutions specifically adapted to your needs.

- With mobile items, you can gain new clients for your firm. They create innovative iOS and Android apps that meet market standards.
- Use our Website design and development services to increase your company's online presence. They can also design interactive mobile web solutions to meet your needs.

3) COMPANY C

Company C (Peshawar, Pakistan) is a CMMI Level-3 software development firm based in Peshawar, Pakistan that offers Information Technology products and services that integrate applications and data within an enterprise and across the industry. The firm provides a complete spectrum of services ranging from Financial Systems, Enterprise Resource Planning (ERP), Monitoring & Evaluation (M&E) Systems to Business Process Re-Engineering (BPR), Mobile based Information Systems, and eGovernment Information and services automation.

Our team of highly experienced and qualified professionals provides practical solutions to organizations in both the Public and Private sectors. We build cohesive, flexible, and cost-effective IT solutions and guarantee their delivery on time and within budget. While ensuring timely and efficient delivery of services, they are not daunted by new technologies and always strive to stay ahead of the technological advances in other parts of the world.

Their services include an online web-based software application, Desktop, and Mobile development. They also design and develop embedded systems. They integrate different systems and implement an e-payments gateway. They think and innovate for you with new ideas to solve unsolved problems.

They combine Design Thinking, Agile Practices, Software Quality Processes, and Innovation Management to create and build digital transformation solutions. It offers five service pillars: Software Solution Design, Agile Software Development, Software Testing, Agile Team Allocation, and Professional Allocation.

E. ANALYSIS

The data collected during the case studies were analyzed in the following ways:

- To assess company security assurance level for CSSRs faced by GSD vendor organizations in secure software development.
- To calculate the score for each CSSR and practice of each company, see Tables 9, 10, and 11.

TABLE 9. **Assessment results of company A.**

SAM of Software Development Levels		Critical Software Security Risks (CSSRs)	Score	Status
S. No	Security Assurance Levels			
Level: 1	Governance and Security Threat Analysis	CSSR1	7	Strong
		CSSR2	8	Strong
		CSSR3	7	Strong
Level: 2	Secure Requirement Engineering	CSSR4	7	Strong
		CSSR5	8	Strong
		CSSR6	8	Strong
		CSSR7	8	Strong
		CSSR8	8	Strong
		CSSR9	8	Strong
		CSSR10	7	Strong
		CSSR11	7	Strong
		CSSR12	7	Strong
		CSSR13	7	Strong
Level: 3	Secure Design	CSSR14	7	Strong
		CSSR15	8	Strong
		CSSR16	7	Strong
		CSSR17	7	Strong
		CSSR18	7	Strong
		CSSR19	7	Strong
		CSSR20	7	Strong
		CSSR21	7	Strong
		CSSR22	7	Strong
		CSSR23	7	Strong
		CSSR24	7	Strong
Level: 4	Secure Coding	CSSR25	7	Strong
		CSSR26	7	Strong
		CSSR27	7	Strong
		CSSR28	7	Strong
		CSSR29	8	Strong
		CSSR30	7	Strong
		CSSR31	8	Strong
		CSSR32	8	Strong
		CSSR33	8	Strong
		CSSR34	8	Strong
		CSSR35	8	Strong
		CSSR36	7	Strong
Level: 5	Secure Testing and Review	CSSR37	8	Strong
		CSSR38	6	Weak
		CSSR39	7	Strong
		CSSR40	8	Strong
Level: 6	Secure Deployment	CSSR41	7	Strong
		CSSR42	8	Strong
		CSSR43	7	Strong
Level: 7	Security Improvement	CSSR44	8	Strong
		CSSR45	7	Strong
		CSSR46	7	Strong

- The CSSR will be recognized as "Strong" (strongly addressed) if the overall score is greater or equal to 7. Otherwise, "Weak" (weakly addressed)
- To examine whether SAM of Software Development needs any improvement, we conducted a focus group session with the participants to get feedback

The focus session aims:

- To determine whether or not SAM of Software Development can be used effectively in an organization to identify strong and weak CSSRs in the context of Secure Software Development.
- Identify whether SAM of Software Development is evident, easy to use, and specifically helpful in measuring and eliminating security risks in Secure Software Development.
- To assess the participant's satisfaction with the SAM of Software Development assessment outcomes.
- Verify that each CSSR's practices are easy to follow.
- Validate SAM of Software Development generalization and applicability for GSD vendor companies.
- They identify strong and weak software security activities in Secure Software Development using SAM software development.

F. ASSESSMENT RESULTS OF COMPANY A

We have used Motorola Assessment Tool [95] for the assessment results of each company. A score of 7 or above for each CSSR indicates a specific company had successfully addressed the risk. Any CSSR with a score of less than seven is considered a weakness. The Company-A participant has

TABLE 10. Assessment results of company B.

SAM of Software Development Levels		Critical Software Security Risks (CSSRs)	Score	Status
S. No	**Security Assurance Levels**			
Level: 1	Governance and Security Threat Analysis	CSSR1	7	Strong
		CSSR2	7	Strong
		CSSR3	7	Strong
Level: 2	Secure Requirement Engineering	CSSR4	7	Strong
		CSSR5	7	Strong
		CSSR6	7	Strong
		CSSR7	8	Strong
		CSSR8	8	Strong
		CSSR9	7	Strong
		CSSR10	7	Strong
		CSSR11	7	Strong
		CSSR12	7	Strong
		CSSR13	7	Strong
Level: 3	Secure Design	CSSR14	7	Strong
		CSSR15	7	Strong
		CSSR16	7	Strong
		CSSR17	6	Weak
		CSSR18	7	Strong
		CSSR19	7	Strong
		CSSR20	7	Strong
		CSSR21	5	Weak
		CSSR22	7	Strong
		CSSR23	7	Strong
		CSSR24	7	Strong
Level: 4	Secure Coding	CSSR25	8	Strong
		CSSR26	6	Weak
		CSSR27	8	Strong
		CSSR28	6	Weak
		CSSR29	6	Weak
		CSSR30	5	Weak
		CSSR31	7	Strong
		CSSR32	6	Weak
		CSSR33	8	Strong
		CSSR34	6	Weak
		CSSR35	6	Weak
		CSSR36	7	Weak
Level: 5	Secure Testing and Review	CSSR37	7	Strong
		CSSR38	7	Strong
		CSSR39	8	Strong
		CSSR40	5	Weak
Level: 6	Secure Deployment	CSSR41	8	Strong
		CSSR42	6	Weak
		CSSR43	7	Strong
Level: 7	Security Improvement	CSSR44	7	Strong
		CSSR45	6	Weak
		CSSR46	8	Strong

measured his security assurance for these CSSRs using SAM of Software Development. Table 9 presents the Company-A assessment results. The assessment will focus on the following points:

- Table 9 presents that Company-A stands at Level-4 "Security Coding" of the SAM of Software Development and comprehensively addresses the security risks of the previous three levels (Governance and Security Analysis, Secure Requirement Engineering, and Secure Desing) since almost it achieved a score greater than 7.
- Company-A is weak in addressing the CSSR38 "Messy code, code bad smells, dead code", since the score values are less than 7.

However, Company A should focus more on monitoring the success of practice implementation rather than organizational commitment. This will aid in creating secure software by enhancing its security assurance:

G. ASSESSMENT RESULTS OF COMPANY B

Table 10 presents the Company-B assessment results. The assessment will focus on the following points:

- Table 10 presents that Company-B stands at Level-3 "Secure Design" of the SAM of Software Development. It has fully addressed the security risks of the other two levels (Governance and Security Threat Analysis and Secure Requirement Engineering) since almost

TABLE 11. Assessment results of company C.

SAM of Software Development Levels		Critical Software Security Risks (CSSRs)	Score	Status
S. No	Security Assurance Levels			
Level: 1	Governance and Security Threat Analysis	CSSR1	8	Strong
		CSSR2	8	Strong
		CSSR3	8	Strong
Level: 2	Secure Requirement Engineering	CSSR4	7	Strong
		CSSR5	8	Strong
		CSSR6	7	Strong
		CSSR7	8	Strong
		CSSR8	7	Strong
		CSSR9	8	Strong
		CSSR10	7	Strong
		CSSR11	8	Strong
		CSSR12	8	Strong
		CSSR13	8	Strong
Level: 3	Secure Design	CSSR14	7	Strong
		CSSR15	8	Strong
		CSSR16	9	Strong
		CSSR17	8	Strong
		CSSR18	7	Strong
		CSSR19	8	Strong
		CSSR20	7	Strong
		CSSR21	8	Strong
		CSSR22	7	Strong
		CSSR23	8	Strong
		CSSR24	8	Strong
Level: 4	Secure Coding	CSSR25	8	Strong
		CSSR26	7	Strong
		CSSR27	7	Strong
		CSSR28	8	Strong
		CSSR29	7	Strong
		CSSR30	9	Strong
		CSSR31	7	Strong
		CSSR32	7	Strong
		CSSR33	7	Strong
		CSSR34	9	Strong
		CSSR35	7	Strong
		CSSR36	7	Strong
Level: 5	Secure Testing and Review	CSSR37	7	Strong
		CSSR38	7	Strong
		CSSR39	7	Strong
		CSSR40	7	Strong
Level: 6	Secure Deployment	CSSR41	7	Strong
		CSSR42	7	Strong
		CSSR43	7	Strong
Level: 7	Security Improvement	CSSR44	7	Strong
		CSSR45	8	Strong
		CSSR46	7	Strong

it achieved a score greater than 7 in the first three levels.

- Company-B is weak in achieving Level-4 "Secure Coding" since the score values of "CSSR17: Improper secure design documentation" and "CSSR21: Improper conduction of design and architecture security review" are less than 7.
- Each of the practices for which Company-B has a score of less than 7 requires improvement.
- It should prioritize Level-4 "Secure Coding" practices. This will help achieve a high-security assurance level in secure software development.

The results show that Company-B is doing very little in security activities in secure coding, which is an area for improvement.

H. ASSESSMENT RESULTS OF COMPANY C

Table 11 presents the Company-C assessment results. The assessment will focus on the following points:

- Table 11 presents that Company-C achieved all the security assurance levels of SAM of Software Development and comprehensively addresses the security risks of all levels since almost it achieved a score greater than 7.

TABLE 12. Feedback of the case studies participants (company-A, B and C).

Applicability's of the SAM of Software Development	Questions	Company-A	Company-B	Company-C
Ease of Learning	1. SAM of Software Development representation is very clear	Strongly Agree	Strongly Agree	Agree
	2. A little knowledge of Secure Software Development is required to learn how to use the SAM of Software Development	Strongly Agree	Agree	Strongly Agree
	3. It is easy to understand the practices designed for each critical software security risk	Strongly Agree	Agree	Agree
	4. It is easy to understand the assessment method	Agree	Strongly Agree	Agree
	5. It is easy to use the SAM of Software Development to assess organizations in addressing the software security risks faced by GSD vendor organization in software development projects.	Agree	Strongly Agree	Agree
	6. It is easy to understand distribution of critical software security risks and practices for addressing these risks among different software security assurance levels, e.g. Governance and threat analysis, Secure Requirement Engineering, Secure Design, Secure Coding, Security Testing and Review, Secure Deployment, and Security Improvement.	Strongly Agree	Agree	Strongly Agree
	7. Some training needs to be provided for the use of SAM of Software Development	Agree	Agree	Agree
	8. How confident are you in the ratings that you have made in this section?	Confident	Confident	Confident
User Satisfaction	9. SAM of Software Development is general and can be applied to most companies.	Agree	Strongly Agree	Agree
	10. Each individual practice is easy to understand and unambiguous.	Strongly Agree	Strongly Agree	Agree
	11. Using the SAM of Software Development would identify strong and weak areas in the company regarding secure software development activities.	Strongly Agree	Agree	Strongly Agree
	12. Using the SAM of Software Development would improve our secure software development processes.	Agree	Agree	Agree
	13. If the SAM of Software Development were available for my job, I predict that I would use it on regular basis in the future.	Strongly Agree	Strongly Agree	Agree
	14. I am satisfied and agreed with the security assurance issues identified by SAM of Software Development.	Strongly Agree	Strongly Agree	Agree
	15. It is important to implement SAM of Software Development in the form of an automated software tool in order to facilitate software practitioners in assessing organization's security in order to address the software security risks.	Agree	Agree	Strongly Agree
	16. The SAM of Software Development is self-contained	Strongly Agree	Agree	Strongly Agree
	17. The SAM of Software Development is a useful software security assurance tool for software development organizations.	Strongly Agree	Strongly Agree	Strongly Agree
	18. The assessment method is useful	Strongly Agree	Strongly Agree	Strongly Agree
	19. How confident are you in the ratings that you have made in this section?	Confident	Very	Confident
Structure of the SAM of Software Development	20.All the components of the SAM of Software Development are self explanatory and require no further explanation to be used effectively	Strongly Agree	Strongly Agree	Agree
	21. The components of the SAM of Software Development are practical and are applicable in secure software development industry	Strongly Agree	Agree	Agree
	22. The SAM of Software Development can be used effectively to identify software security risks faced to GSD vendor organizations in secure software development projects.	Strongly Agree	Agree	Agree
	23. The distribution of software security risks among different security assurance levels (e.g. Governance and threat analysis, Secure Requirement Engineering, Secure Design, Secure Coding, Security Testing and Review, Secure Deployment, and Security Improvement) is useful	Agree	Agree	Agree
	24. The 7software security assurance levels of SAM of Software Development are useful	Strongly Agree	Agree	Agree
	25. How confident are you in the ratings that you have made in this section?	Confident	Very	Confident

Company-A Feedback about Question 26, 27, 28, 29, and 30:

26. Are there any modifications or improvements to the SAM of Software Development that you may suggest?

Ans: Many times software developers face problem during System and Application integration leading to failure of software projects also. Further Maintenance and Up gradation becomes a problem for software developers for some software projects. The company need to make changes are acceptable till design but once development starts any further change should be rejected.

27. Are there any components that you may suggest to add to the SAM of Software Development in the future, please also give the reasons?

Ans: Integrated Development Environment (IDE)-Modern IDEs like Visual Studio or Eclipse offer so much support to the coding process - built-in wizards to help you accomplish numerous tasks, code completion and dependency management, are just a few examples of standard features - that it's almost inconceivable to attempt a serious application without one.

TABLE 12. ***(Continued.)*** **Feedback of the case studies participants (company-A, B and C).**

28. Please provide any comments relating to the assessment method.
Ans: Software testing Technology CMMI. The Capability Maturity Model Integration (CMMI) is a process model that provides a lucid definition of the process improvement approach which examines whether an organization's current processes are in place and helps them identify their strengths and weaknesses.
29. Please provide any comments relating to the distribution of practices across various critical software security risks.
Ans: Web applications that do not properly protect sensitive data could allow threat actors to steal or modify weakly protected data. They could also conduct malicious activities such as credit card fraud and identity theft, among others. Improperly configured or badly coded APIs could also lead to a data breach.
30. Please provide any comments relating to the usability of SAM of Software Development with respect to time it take users to addressing the CSSRs.
Ans: A key part of the software development process, usability testing provides invaluable feedback on the user experience of a product. Usability testing means conducting real-world testing with a segment of your customer base.

Company-B Feedback about Question 26, 27, 28, 29, and 30:

26. Are there any modifications or improvements to the SAM of Software Development that you may suggest?
Ans: No, I think for the time being it's enough good.
27. Are there any components that you may suggest to add to the SAM of Software Development in the future, please also give the reasons?
Ans: In future you can take idea from Software Assurance Maturity Model (SAMM) By OWASP, To enhance SAM of Software Development.
28. Please provide any comments relating to the assessment method.
Ans: I think it's good enough.
29. Please provide any comments relating to the distribution of practices across various critical software security risks.
Ans: Critical Important software used worldwide must have strict security measures in place to ensure safe and secure use. Risk, misuse, or disruption of important software may jeopardize agency and business deployments, while also causing financial damage, loss of trust, loss of life and widespread social consequences. Not only is sensitive software itself protected, but the IT environment of the agency, its customers, partners, and the public must be protected from any potential misuse or malware by the software.
30. Please provide any comments relating to the usability of SAM of Software Development with respect to time it takes users to addressing the CSSRs.
Ans: SAM of Software Development will definitely increase the security of software if all step address by CSSRs is applied with proper care.

Company-C Feedback about Question 26, 27, 28, 29, and 30:

26. Are there any modifications or improvements to the SAM of Software Development that you may suggest?
Ans: The SAM Model should suggest the risk category and plan whenever a new risk occurs and also store it for future use.
27. Are there any components that you may suggest to add to the SAM of Software Development in the future, please also give the reasons?
Ans: I would suggest to add risk category predication component in this model.
28. Please provide any comments relating to the assessment method.
Ans: The assessment method of risks are normalized well and are adjusting for majority of cases.
29. Please provide any comments relating to the distribution of practices across various critical software security risks.
Ans: The distribution of risks categories are done effectively as it highlights industry practices.
30. Please provide any comments relating to the usability of SAM of Software Development with respect to time it take users to addressing the CSSRs.
Ans: The SAM Model will be very useful and effective if implemented in a tool to automate the process.

However, Company C should focus more on monitoring the success of practice implementation rather than organizational commitment. This will aid in creating secure software by enhancing its security assurance.

I. FEEDBACK FROM CASE STUDY PARTICIPANTS

The case study enabled us to evaluate the practicability of the SAM of Software Development for GSD vendor organizations. To get feedback, we asked 30 questions from the case study participants, which are summarized in Table 12.

VI. THREATS TO VALIDITY

This section underlines the threats to validity. And how we overcome them to strengthen our confidence in our investigation's conclusions. According to [97], internal, external, and conclusion-validity threats are all examples of risk categories.

A. CONSTRUCT VALIDITY

One of the possible concerns is the lack of sources available in other databases or after the research was completed. We used an SLR to find all of the relevant articles in the formal literature, and we searched six different databases to make sure we found everything.

We also collected the data through a questionnaire survey from software development organizations. This ensured that all relevant sources were covered completely. As a result, we have sufficient evidence that SAM of Software Development covers most software security procedures.

There is also a problem with assigning practices to CSSRs when making a SAM of Software Development. This can be subjective. We carried out the assignment based on our experiences and what we learned from the SLR and questionnaire survey.

Three university professors (2nd Co-Author (Supervisor), 3rd Co-Author, and 4th Co-Author (Co-Supervisor)) repeatedly validated the structure of SAM of Software Development and its security assurance levels and activities to reduce this threat.

B. INTERNAL VALIDITY

The techniques we discovered from our SLR and questionnaire survey are a good approximation of current secure software development methods. SLR data extraction and source selection are prone to human error because not all sources give sufficient or explicit information about

TABLE 13. Compairison of our study with other relevant studies.

S. No	Paper Titles	No of Primary Studies	Time Interval	Resources	Goals
1	The state of the art on design patterns: a systematic mapping of the literature [98]	101	Up to March 2016	Relevant Journals and Conferences	The main contributions are: identification of research Topics in design patterns, quantification of research emphasis on each topic, and description of design pattern demographics
2	Exploring software security approaches in Software Development lifecycle: A Systematic Mapping Study [18]	118	Up to August 2015	ACM, IEEEXplore, ScienceDirect, SpringerLink, John Wiley	Identify existing approaches to software security used in the SDLC
3	Software Development Initiatives to Identify and Mitigate Security Threats: A Systematic Mapping study [19]	15	2000-2015	Only security related Journals	To cover existing software security threats identification and mitigation technologies
4	Mapping the Field of Software Life Cycle Security Metrics [99]	63	2000-2014	ACM, IEEE Xplore, Elsevier	This research aims to promote the use of safety measurements by researchers and practitioners by cataloguing available metrics
5	A Systematic Review on the Engineering of Software for Ubiquitous Systems [20]	128	2009-2015	ACM, IEEE Xplore, Science Direct, Wiley Online Library, Springer Link, Google Scholar	They identified 132 approaches that address issues for ubiquitous systems at various stages of the Software engineering cycle. Implementation, evolution/ maintenance, and feedback phases have been most studied
6	Design Notation for Secure Software: A Systematic Literature Review [75]	42	2002-2012	ACM, IEEE Xplore, CiteSeerX, Compendex, ISI Web of Sciences, Springer Link	This paper reported an SLR's results in secure software design notes. They identified 28 design notes for the development of software
7	A Systematic Literature Review on Global Software Development Life Cycle [100]	51	2000-2013	ACM, IEEE Xplore, Springer Link, Science Direct, Wiley InterScience	In this paper, they report SLR findings to identify the challenges faced by the globally distributed teams during different software development phases. They also suggested best practices and tools alleviate these challenges.
8	An Extensive Systematic Review on the Model-Driven Development of Secure Systems [101]	93	2001-2014	ACM, IEEE Xplore, Springer Link, Science Direct, ISI	This study shows the overall status of Model-Driven Security's key artefacts (MDS) and the primary MDS studies identified, e.g., regarding artefact modeling security. They suggested that in many MDS approaches, the development of domain-specific languages plays a key role
9	Web Services Attacks and Security- A Systematic Literature Review [102]	36	2005-2016	Journals and Conferences	This study aims to present a systematic review of web security studies. It is found that the Denial-of-Services attack is the most addressed of all attacks. Solutions were proposed primarily using dynamics analysis, closed and static analysis, followed
10	A Systematic Review of Security Requirements Engineering [10]	22	2005-2010	ACM, IEEE Xplore, Science Direct, Google Scholar	In this paper, the authors conduct a systematic review of secure requirements engineering to summarize the evidence on this issue and provide a framework/background in which new research activities can be appropriately positioned
11	**Software Security Assurance (SAM) of Software Development (Our Paper, Research Work)**	**116**	**Up to October 2020**	**IEEE Xplore, Science Direct, ACM, Springer Link, Wiley Online Library, Google Scholar**	This article reviews the most widely used security software models. It proposes a new Security Assurance Model (SAM) for Software Development that is adaptable to all contemporary scenarios, emphasizing global software development (GSD) vendor companies. The SAM of Software Development was developed after studying 11 well-known development models and analyzing results obtained from a systematic literature review (SLR) and questionnaire survey. The SAM of Software Development consists of seven security assurance levels: Governance and Security Threat Analysis, Secure Requirement

TABLE 13. ***(Continued.)*** **Compairison of our study with other relevant studies.**

					Analysis, Secure Design, Secure Coding, Secure Testing and Review, Secure Deployment, and Security Improvement. The security assurance levels of SAM of software development consist of 46 critical software security risks (CSSRs) and 388 practices for addressing these risks. The proposed SAM of Software Development was assessed based on a tool created by Motorola, which is used to evaluate the present state of a company's software processes and find areas for improvement. We conducted 3 case studies on software development companies, using data from real software projects to examine the results of a practical experiment in each company. The results of the case studies indicate that the proposed SAM of Software Development helps measure the security assurance level of an organization. In addition, it can potentially serve as a framework for researchers to develop new software security measures.

our research topics. The sources were thoroughly reviewed and chosen based on quality criteria to minimise this limitation.

We have concerns regarding the accuracy of the assessment results because the participants in our case study appraised the organization's activities. The assessment may be subjective because adhering to the requirements of the Motorola assessment instrument necessitates the assessor's close attention to acquiring correct results.

In addition to existing software security experience, members' roles within the firm are also factors.

C. EXTERNAL VALIDITY

The study's findings may not apply to all software development firms. Only three companies took part in the case studies we used to determine how well SAM of Software Development worked. Consequently, it is essential to be careful when making decisions about the SAM of Software Development's applicability.

D. CONCLUSION VALIDITY

We gathered adequate information to support our conclusions about existing CSSRs and practices for addressing these risks of secure software development by conducting the SLR and questionnaire survey. The SLR and questionnaire were meticulously carried out methodically, and the sources were assessed using quality standards. This increases our confidence in SAM of Software Development and decreases the risk of jeopardizing conclusion validity.

VII. CONCLUSION AND FUTURE WORK

This section outlines our study's effort and main contributions. It also outlines the research directions that should be pursued in the future.

A. CONCLUSION AND DISCUSSION

This study's primary purpose is to develop a Security Assurance Model (SAM) of Software Development for GSD Vendors. In the SDLC phases, this model will assist software development firms in reviewing and enhancing their security processes. The primary goal of the first phase (doing SMS) is to determine what the current state-of-the-art is in terms of secure software engineering (SSE) [22], [23]. The final selection was made from 116 studies that met the inclusion and exclusion criteria [22], [23]. After extracting data from the selected articles, they were classified according to their quality, software security processes/models/frameworks, software security methodologies, SDLC phases, publication venue, and SWOT analysis of the software security approaches [22], [23].

SLR was used to classify key papers to identify security threats and practices during the second phase of this research [48]. The tollgate [49] approach was used to choose 121 papers based on inclusion, exclusion, and quality rating criteria. This study identified 145 security risks and 424 best practices that can assist software development businesses in managing security throughout the SDLC's various phases [48].

In the 3rd stage of this study, an online survey method is employed for data collection [59]. During the survey's implementation, a total of 64 responses were collected. All of the responses were manually reviewed. We excluded 14 responses because the expertise shared by these 14 persons was unrelated to GSD and/or SSD. For analysis, the final 50 survey results were taken into account. This study empirically validated 145 software security risks and 424 security practices that assist GSD vendor organizations in managing the security activities in the SDLC phases.

In the 4th stage of this research, we identify the critical software security risks (CSSRs); we used the criterion of 10% occurrences in both SLR and survey findings. Based on this criterion, we identified 46 critical software security risks in the SDLC phases.

In the 5th stage of this research, we have developed SAM of Software Development for GSD Vendors by studying various models frameworks [4], [8], [13], [15], [16], [31], [35], [38], [39], [41], [43], [58], [68], [81], [94] and the results obtained from the SLR [48] and questionnaire survey. The CSSRs and practices for addressing these risks are grouped into seven different levels, which we call security assurance levels: "**Level-1:** Governance and Security Threat Analysis, **Level-2:** Secure Requirement Analysis, **Level-3:** Secure

Design, **Level-4:** Secure Coding, **Level-5:** Secure Testing and Review, **Level-6:** Secure Deployment, **Level-7:** Security Improvement". A total of 46 CSSRs and 388 practices (see Table 1-7) were identified using SLR and questionnaire survey.

The SAM of Software Development assessment process is based on the Motorola evaluation tool [95]. Motorola created this tool to evaluate the present state of a company's software processes and find areas for improvement [95].

In the last stage of this research, the model is tested as a case study in a software development company, using data from real software projects to examine the results of a practical experiment in that company. On compares the results in two development scenarios: one with reactive security and one with proactive security in all phases of software development (SDLC). The case study results indicate that SAM of Software Development helps measure the security assurance level of an organization. It will also serve as a framework for researchers to develop new software security measures.

This study recommended various security practices to follow in each phase of the SDLC to achieve a secure SDLC. The successful integration of these operations reduces effort, time, and cost while creating secure software applications. It helps software development businesses improve their software security and efficiency. This will also raise the developer's knowledge of the importance of secure development techniques.

We have briefly answered the research questions mentioned in Section-I in the following paragraphs:

RQ1: How can a secure SDLC be developed for GSD vendor companies that are both practical and robust?

We have developed SAM of Software Development for GSD Vendors by studying various models and frameworks [4], [8], [13], [15], [16], [31], [35], [38], [39], [41], [43], [58], [68], [81], [94] and the results obtained from the SLR [48] and questionnaire survey. The process flow for SAM of software development is depicted in Section V (see Figure 3).

The CSSRs and practices for addressing these risks are grouped into seven different levels, which we call security assurance levels, e. g,

- **Level-1:** Governance and Security Threat Analysis
- **Level-2:** Secure Requirement Analysis
- **Level-3:** Secure Design
- **Level-4:** Secure Coding
- **Level-5:** Secure Testing and Review
- **Level-6:** Secure Deployment
- **Level-7:** Security Improvement

The security assurance levels of SAM of software development for GSD vendor organizations are illustrated in Section V-D (see Figure 4). 46 CSSRs and 388 practices (see Section IV, Table 1-7) were identified using SLR and questionnaire survey.

We have developed SAM of Software Development for GSD Vendors by studying various models frameworks [4], [8], [13], [15], [16], [31], [35], [38], [39], [41], [43], [58], [68], [81], [94] and the results obtained from the SLR [48] and questionnaire survey. The process flow for SAM of software development is depicted in Figure 3.

The SAM of Software Development assessment process is based on the Motorola evaluation tool [95]. Motorola created this tool to evaluate the present state of a company's software processes and find areas for improvement [95]. Other researchers have widely used it to assess their proposed models' maturity, readiness, or assurance [15], [41], [58], [68], [96].

RQ2: Is the proposed security assurance model capable of assisting GSD organizations in determining their security assurance to produce secure software?

A case study is considered an appropriate research method in software engineering and a highly effective tool for validation purposes. In the last stage of this research, we have conducted three case studies on software development companies to validate our projected model (SAM of Software Development). To get feedback on our suggested model, we convened focus groups with the participants of the case studies. We chose one large, one medium-sized, and one small-sized company for our case study to reduce the impact of their size.

We have used Motorola Assessment Tool for the assessment results of each company. A score of 7 or above for each CSSR indicates a specific company had successfully addressed the risk. Any CSSR with a score of less than seven is considered a weakness. The Company-A participant has measured his security assurance for these CSSRs using SAM of Software Development.

Table 9 presents the Company-A assessment results. The assessment will focus on the following points:

- Table 9 presents that Company-A stands at Level-4 "Security Coding" of the SAM of Software Development and comprehensively addresses the security risks of the previous three levels (Governance and Security Analysis, Secure Requirement Engineering, and Secure Desing) since almost it achieved a score greater than 7.
- Company-A is weak in addressing the CSSR38 "Messy code, code bad smells, dead code", since the score values are less than 7.

Table 10 presents the Company-B assessment results. The assessment will focus on the following points:

- Table 10 presents that Company-B stands at Level-3 "Secure Design" of the SAM of Software Development. It has fully addressed the security risks of the other two levels (Governance and Security Threat Analysis and Secure Requirement Engineering) since almost it achieved a score greater than 7 in the first three levels.
- Company-B is weak in achieving Level-4 "Secure Coding" since the score values of "CSSR17: Improper secure design documentation" and "CSSR21: Improper conduction of design and architecture security review" are less than 7.
- Each of the practices for which Company-B has a score of less than 7 requires improvement.

Table 11 presents the Company-C assessment results. The assessment will focus on the following points:

- Table 11 presents that Company-C achieved all the security assurance levels of SAM of Software Development and comprehensively addresses the security risks of all levels since almost it achieved a score greater than 7.

However, Company C should focus more on monitoring the success of practice implementation rather than organizational commitment. This will aid in creating secure software by enhancing its security assurance.

The case study enabled us to evaluate the practicability of the SAM of Software Development for GSD vendor organizations. To get feedback, we asked 30 questions from the case study participants, which are summarized in Section V-I (see Table 12) for details.

We compare the goals of our research work with other relevant studies, as depicted in Table 12.

B. FUTURE RESEARCH DIRECTIONS

With the increasing number of software security threats, regularly upgrade software security processes and practices. This study project can be improved in a variety of ways. The following are some of the open study directions that researchers can look into:

- To improve the outcomes of SAM of Software Development, collaboration with software development organizations is required. Depending on the facilities and methods used, it might be adapted to meet the needs of various organizations.
- The SAM of Software Development might include characteristics relating to specific technologies like the Internet of Things (IoT), blockchain, and cloud computing.
- The SAM) of Software Development might be made available as an online repository (tool) updated regularly with new academic and industry practices. The SAM of Software Development will become a reliable resource for scholars and practitioners.

REFERENCES

[1] M. Zhang, X. de Carné de 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. 17, no. 9, pp. 2315–2330, Sep. 2019.

[2] G. McGraw, "Six tech trends impacting software security," *Computer*, vol. 50, no. 5, pp. 100–102, May 2017.

[3] 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.

[4] 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.

[5] M. Z. Gunduz and R. Das, "Cyber-security on smart grid: Threats and potential solutions," *Comput. Netw.*, vol. 169, Mar. 2020, Art. no. 107094.

[6] 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.

[7] 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.

[8] R. Khan, "Secure software development: A prescriptive framework," *Comput. Fraud Secur.*, vol. 2011, no. 8, pp. 12–20, Aug. 2011.

[9] 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.

[10] 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.

[11] 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.

[12] Y. Lee and G. Lee, "HW-CDI: Hard-wired control data integrity," *IEEE Access*, vol. 7, pp. 10811–10822, 2019.

[13] 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.

[14] S. Z. Hlaing and K. Ochimizu, "An integrated cost-effective security requirement engineering process in SDLC using FRAM," in *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, Dec. 2018, pp. 852–857.

[15] 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.

[16] M. Khari, Vaishali, and P. Kumar, "Embedding security in software development life cycle (SDLC)," in *Proc. 3rd Int. Conf. Comput. Sustain. Global Develop. (INDIACom)*, Mar. 2016, pp. 2182–2186.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] R. Kumar, A. Baz, H. Alhakami, W. Alhakami, M. Baz, A. Agrawal, and R. A. Khan, "A hybrid model of hesitant fuzzy decision-making analysis for estimating usable-security of software," *IEEE Access*, vol. 8, pp. 72694–72712, 2020.

[22] 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.

[23] 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, 2020, pp. 487–492.

[24] D. Verdon and G. McGraw, "Risk analysis in software design," *IEEE Security Privacy*, vol. 2, no. 4, pp. 79–84, Jul. 2004.

[25] N. R. Mead and G. McGraw, "A portal for software security," *IEEE Security Privacy*, vol. 3, no. 4, pp. 75–79, Jul. 2005.

[26] B. Potter and G. McGraw, "Software security testing," *IEEE Security Privacy*, vol. 2, no. 5, pp. 81–85, Sep. 2004.

[27] 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.

[28] 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.

[29] 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.

[30] 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, Jan. 2006.

[31] C. P. Team, "CMMI for development, version 1.3," Softw. Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep., CMU/SEI-2010-TR-033, 2010.

[32] M. Alshayeb, A. Abdellatif, S. Zahran, and M. Niazi, "Towards a framework for software product maturity measurement," in *Proc. 10th Int. Conf. Softw. Eng. Adv.*, 2015, pp. 7–11.

[33] A. Abdellatif, M. Alshayeb, S. Zahran, and M. Niazi, "A measurement framework for software product maturity assessment," *J. Softw., Evol. Process*, vol. 31, no. 4, p. e2151, Apr. 2019.

[34] R. Eckert, S. K. Meyer, and M. Stuermer, "How are open source practices possible within a medical diagnostics company? Developing and testing a maturity model of inner source implementation," in *Proc. 13th Int. Symp. Open Collaboration*, Galway, Ireland, Aug. 2017, pp. 1–8.

[35] R. Qutaish and A. Abran, "A maturity model of software product quality," *J. Res. Pract. Inf. Technol.*, vol. 43, no. 4, pp. 307–327, 2011.

[36] A. B. Jakobsen, M. O'Duffy, and T. Punter, "Towards a maturity model for software product evaluations," in *Proc. 10th Eur. Conf. Softw. Cost Estimation (ESCOM)*, 1999, pp. 329–333.

[37] A. April, J. H. Hayes, A. Abran, and R. Dumke, "Software maintenance maturity model (SM^{mm}): The software maintenance process model," *J. Softw. Maintenance Evol., Res. Pract.*, vol. 17, no. 3, pp. 197–223, 2005.

[38] M. Pereira da Silva and R. Miranda de Barros, "Maturity model of information security for software developers," *IEEE Latin Amer. Trans.*, vol. 15, no. 10, pp. 1994–1999, Oct. 2017.

[39] S. R. Ahmed, "Secure software development: Identification of security activities and their integration in software development lifecycle," M.S. thesis, Softw. Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2007.

[40] M. Essafi, L. Labed, and H. B. Ghezala, "S2D-ProM: A strategy oriented process model for secure software development," in *Proc. Int. Conf. Softw. Eng. Adv. (ICSEA)*, Aug. 2007, p. 24.

[41] 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.

[42] J. Manico, "Application security verification standard 3.0.1," OWASP, USA, Tech. Rep., Version 3.0, 2016, pp. 1–70.

[43] *Building Security in Maturity Model (BSIMM)*, BSIMM12, USA, 2022.

[44] B. A. Kitchenham, D. Budgen, and O. P. Brereton, "Using mapping studies as the basis for further research—A participant-observer case study," *Inf. Softw. Technol.*, vol. 53, pp. 638–651, Jun. 2011.

[45] 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.

[46] J. Morán, C. Riva, and J. Tuya, "Testing MapReduce programs: A systematic mapping study," *J. Softw., Evol. Process*, vol. 31, no. 3, p. e2120, Mar. 2019.

[47] R. E. Lopez-Herrejon, S. Illescas, and A. Egyed, "A systematic mapping study of information visualization for software product line engineering," *J. Softw., Evol. Process*, vol. 30, no. 2, p. e1912, Feb. 2018.

[48] R. A. Khan, S. U. Khan, H. U. Khan, and M. Ilyas, "Systematic literature review on security risks and its practices in secure software development," *IEEE Access*, vol. 10, pp. 5456–5481, 2022.

[49] 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.

[50] B. Kitchenham and S. L. Pfleeger, "Principles of survey research part 6: Data analysis," *ACM SIGSOFT Softw. Eng. Notes*, vol. 28, pp. 24–27, Mar. 2003.

[51] H. U. Khan, M. Niazi, M. El-Attar, N. Ikram, S. U. Khan, and A. Q. Gill, "Empirical investigation of critical requirements engineering practices for global software development," *IEEE Access*, vol. 9, pp. 93593–93613, 2021.

[52] H. U. Rahman, M. Raza, P. Afsar, and H. U. Khan, "Empirical investigation of influencing factors regarding offshore outsourcing decision of application maintenance," *IEEE Access*, vol. 9, pp. 58589–58608, 2021.

[53] J. A. Khan, S. U. R. Khan, J. Iqbal, and I. U. Rehman, "Empirical investigation about the factors affecting the cost estimation in global software development context," *IEEE Access*, vol. 9, pp. 22274–22294, 2021.

[54] M. A. Akbar, W. Naveed, A. A. Alsanad, L. Alsuwaidan, A. Alsanad, A. Gumaei, M. Shafiq, and M. T. Riaz, "Requirements change management challenges of global software development: An empirical investigation," *IEEE Access*, vol. 8, pp. 203070–203085, 2020.

[55] M. A. Akbar, S. Mahmood, A. Alsanad, M. Shafiq, A. Gumaei, and A. A.-A. Alsanad, "Organization type and size based identification of requirements change management challenges in global software development," *IEEE Access*, vol. 8, pp. 94089–94111, 2020.

[56] S. Beecham, T. Clear, R. Lal, and J. Noll, "Do scaling agile frameworks address global software development risks? An empirical study," *J. Syst. Softw.*, vol. 171, Jan. 2021, Art. no. 110823.

[57] M. A. Akbar, H. Alsalman, A. A. Khan, S. Mahmood, C. Meshram, A. H. Gumaei, and M. T. Riaz, "Multicriteria decision making taxonomy of cloud-based global software development motivators," *IEEE Access*, vol. 8, pp. 185290–185310, 2020.

[58] 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.

[59] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical Softw. Eng.*, vol. 10, pp. 311–341, Jul. 2005.

[60] 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.

[61] B. Martin, *Introduction to Medical Statistics*, 4th ed. Oxford, U.K.: Oxford Univ. Press, 2015, pp. 1–464.

[62] 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.

[63] S. U. Khan, M. Niazi, and R. Ahmad, "Critical success factors for offshore software development outsourcing vendors: An empirical study," in *Proc. Int. Conf. Product Focused Softw. Process Improvement*, 2010, pp. 146–160.

[64] S. U. Khan, M. Niazi, and R. Ahmad, "Barriers in the selection of offshore software development outsourcing vendors: An exploratory study using a systematic literature review," *Inf. Softw. Technol.*, vol. 53, pp. 693–706, Jul. 2011.

[65] D. Šmite, C. Wohlin, T. Gorschek, and R. Feldt, "Empirical evidence in global software engineering: A systematic review," *Empirical Softw. Eng.*, vol. 15, no. 1, pp. 91–118, 2010.

[66] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Eng.*, vol. 14, no. 2, pp. 131–164, Apr. 2009.

[67] M. Younas, D. N. A. Jawawi, M. A. Shah, A. Mustafa, M. Awais, M. K. Ishfaq, and K. Wakil, "Elicitation of nonfunctional requirements in agile development using cloud computing environment," *IEEE Access*, vol. 8, pp. 209153–209162, 2020.

[68] Y. Mufti, M. Niazi, M. Alshayeb, and S. Mahmood, "A readiness model for security requirements engineering," *IEEE Access*, vol. 6, pp. 28611–28631, 2018.

[69] I. Keshta, M. Niazi, and M. Alshayeb, "Towards implementation of requirements management specific practices (SP 1.3 and SP 1.4) for Saudi Arabian small and medium sized software development organizations," *IEEE Access*, vol. 5, pp. 24162–24183, 2017.

[70] W. S. Al-Shorafat, "Security in software engineering requirement," in *Proc. IEEE 3rd Int. Conf. Inf. Sci. Technol. (ICIST)*, Dec. 2013, pp. 666–673.

[71] P. Salini and S. Kanmani, "Survey and analysis on security requirements engineering," *Comput. Electr. Eng.*, vol. 38, no. 6, pp. 1785–1797, Nov. 2012.

[72] 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.

[73] V. Maheshwari and M. Prasanna, "Integrating risk assessment and threat modeling within SDLC process," in *Proc. Int. Conf. Inventive Comput. Technol. (ICICT)*, Aug. 2016, pp. 1–5.

[74] 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.

[75] 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.

[76] O. M. Surakhi, A. Hudaib, M. AlShraideh, and M. Khanafseh, "A survey on design methods for secure software development," *Int. J. Comput. Technol.*, vol. 16, no. 7, pp. 7047–7064, Dec. 2017.

[77] G. Pedraza-Garcia, H. Astudillo, and D. Correal, "A methodological approach to apply security tactics in software architecture design," in *Proc. IEEE Colombian Conf. Commun. Comput. (COLCOM)*, Jun. 2014, pp. 1–8.

[78] 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. (FMSE)*, Washington, DC, USA, 2004, pp. 75–85.

[79] 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.

[80] Y.-H. Tung, S.-C. Lo, J.-F. Shih, and H.-F. Lin, "An integrated security testing framework for secure software development life cycle," in *Proc. 18th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Oct. 2016, pp. 1–4.

[81] 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, U.K., Aug. 2019, pp. 1–11.

[82] M. Sodanil, G. Quirchmayr, N. Porrawatpreyakorn, and A. M. Tjoa, "A knowledge transfer framework for secure coding practices," in *Proc. 12th Int. Joint Conf. Comput. Sci. Softw. Eng. (JCSSE)*, Jul. 2015, pp. 120–125.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] M. Asad and S. Ahmed, "Model driven architecture for secure software development life cycle," *Int. J. Comput. Sci. Inf. Secur.*, vol. 14, no. 6, pp. 649–661, 2016.

[88] 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. (NCC)*, Apr. 2018, pp. 1–6.

[89] D. Hein and H. Saiedian, "Secure software engineering: Learning from the past to address future challenges," *Inf. Secur. J., A Global Perspective*, vol. 18, no. 1, pp. 8–25, Feb. 2009.

[90] S. Velmourougan, P. Dhavachelvan, R. Baskaran, and B. Ravikumar, "Software development life cycle model to improve maintainability of software applications," in *Proc. 4th Int. Conf. Adv. Comput. Commun.*, Aug. 2014, pp. 270–273.

[91] 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, Mar. 2020.

[92] G. Pedraza-García, 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, Nov. 2016, p. 37.

[93] S. Islam and W. Dong, "Human factors in software security risk management," in *Proc. 1st Int. Workshop Leadership Manage. Softw. Archit. (LMSA)*, Leipzig, Germany, 2008, pp. 13–16.

[94] *Microsoft Security Development Lifecycle*, Microsoft, Redmond, WA, USA, 2022.

[95] M. K. Daskalantonakis, "Achieving higher SEI levels," *IEEE Softw.*, vol. 11, no. 4, pp. 17–24, Jul. 1994.

[96] S. Ali and S. U. Khan, "Software outsourcing partnership model: An evaluation framework for vendor organizations," *J. Syst. Softw.*, vol. 117, pp. 402–425, Jul. 2016.

[97] J. Rogers and A. Révész, "Experimental and quasi-experimental designs," in *The Routledge Handbook of Research Methods in Applied Linguistics*. London, U.K.: Routledge, 2019.

[98] B. B. Mayvan, A. Rasoolzadegan, and Z. G. Yazdi, "The state of the art on design patterns: A systematic mapping of the literature," *J. Syst. Softw.*, vol. 125, pp. 93–118, Mar. 2017.

[99] P. Morrison, D. Moye, R. Pandita, and L. Williams, "Mapping the field of software life cycle security metrics," *Inf. Softw. Technol.*, vol. 102, pp. 146–159, Oct. 2018.

[100] R. Jain and U. Suman, "A systematic literature review on global software development life cycle," *ACM SIGSOFT Softw. Eng. Notes*, vol. 40, no. 2, pp. 1–14, Apr. 2015.

[101] P. H. Nguyen, M. Kramer, J. Klein, and Y. L. Traon, "An extensive systematic review on the model-driven development of secure systems," *Inf. Softw. Technol.*, vol. 68, pp. 62–81, Dec. 2015.

[102] V. R. Mouli and K. P. Jevitha, "Web services attacks and security—A systematic literature review," *Proc. Comput. Sci.*, vol. 93, pp. 870–877, Jan. 2016.

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. Siffat Ullah 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, global software engineering, secure software engineering, empirical software engineering, systematic literature review, requirements engineering, green computing, software testing, and agile software development.

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. He 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 10 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).

MUSAAD ALZAHRANI received the B.Sc. degree from King Abdulaziz University, Jeddah, Saudi Arabia, in 2008, and the M.Sc. and Ph.D. degrees from Kent State University, Kent, OH, USA, in 2013 and 2017, respectively, all in computer science. He is currently an Assistant Professor with the Faculty of Computer Science and Information Technology, Albaha University, Al-Bahah, Saudi Arabia. His research interests include software engineering, software metrics and qualities, software maintenance, and machine learning.

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

• • •