IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 18, 2020, accepted November 13, 2020, date of publication November 24, 2020,
date of current version December 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3040220

A Maturity Model for Secure Software Design: A Multivocal Study

HASSAN AL-MATOUQ, SAJJAD MAHMOOD, MOHAMMAD ALSHAYEB, AND MAHMOOD NIAZI

Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia

Corresponding author: Mohammad Alshayeb (alshayeb@kfupm.edu.sa)

This work was supported by the Deanship of Scientific Research at King Fahd University of Petroleum and Minerals, Saudi Arabia, under Grant IN171008.

ABSTRACT Security is one of the most important software quality attributes. Software security is about designing and developing secure software that does not allow the integrity, confidentiality, and availability of its code, data, or service to be compromised. Organizations tend to consider security as an afterthought, and they continue to suffer from security risks. Developing secure software requires taking security into consideration in all phases of the Software Development Life Cycle (SDLC). Several approaches have been developed to improve software quality, such as Capability Maturity Model Integration (CMMI). However, software security issues have not been addressed in a proper manner and incorporating security practices into the SDLC remains a challenge. The objective of this paper is to develop a framework to improve the process of designing secure products in software development organizations. To achieve this objective, a Multivocal Literature Review (MLR) was conducted to identify the relevant studies in both the formal and grey literature. A total of 38 primary studies were identified, and available evidence was synthesized into 8 knowledge areas and 65 best practices to build a Secure Software Design Maturity Model (SSDMM). The framework was developed based on the structure of CMMI v2.0 and evaluated through case studies in real-world environments. The case study results indicate that SSDMM is useful in measuring the maturity level of an organization for the secure design phase of SDLC. SSDMM will assist organizations in evaluating and improving their software design security practices. It will also provide a foundation for researchers to develop new software security approaches.

INDEX TERMS Software design, software quality, capability-based security.

I. INTRODUCTION

The number of software vulnerabilities is growing, and security issues are increasing with the popularity of Internet applications, social media systems, cloud computing, and the Internet of Things (IoT). There are more challenges to building secure software due to the large number of connected users and the complexity of software systems [1]. Hackers attack users of these systems by introducing different types of malware (malicious software) such as worms, viruses, trap doors, Trojan horses, and spyware. Several mechanisms are available to detect and mitigate these risks such as installing patches or antivirus software, but these mechanisms become irrelevant with software changes over time. Due to the complexity, extensibility, and connectivity of software systems, software design defects, flaws, and bugs are also common challenges to building secure systems [1]. Attackers exploit these issues to gain access to system resources through various types of attacks such as buffer overflow, incomplete mitigation, and race conditions.

Traditionally, the security strategies of organizations are focused on the level of network systems. They tend to spend large amounts of money on network protection mechanisms such as strong firewalls, intrusion detection systems, anti-spyware and antivirus software, and encryption mechanisms. On the other hand, software security is usually considered to be an afterthought and is often addressed towards the end of the development cycle [2] using an approach commonly known as 'penetrate and patch'. With this approach, software security is assessed after the product is completed either by

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

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

breaking it using common vulnerabilities or discovering new vulnerabilities after deployment. Security patches are then developed and deployed to fix the identified issues. This reactive approach is not effective since organizations continue to suffer from security risks and security flaw exploitations [3] which can lead to major and costly reworks [4]. Patches that are introduced after product release might not be applied by users and attackers might also introduce new vulnerabilities.

Building secure software means building software that functions properly even under malicious attacks [5]. This requires addressing the security challenges through the whole development life cycle [3], especially in the early stages during the design phase. This minimizes the risk of missing critical security requirements and introducing security flaws during the implementation phase. A large number of software security practices, approaches, and tools have been proposed by researchers in the literature [4] and have been implemented by practitioners in the software industry. In addition, some organizations have developed maturity models and frameworks to evaluate the maturity level of their software security practices. However, none of these models and frameworks are explicitly dedicated to the design phase of secure software development. Thus, they do not cover all the aspects and activities of secure software design. Due to the importance of the secure software design phase, there is a need for a secure software design maturity model that is dedicated to secure software design activities. This will enable software development organizations to measure their maturity level and improve their performance in secure software design. It will also increase the awareness level of software designers.

The objective of this thesis is to develop a Secure Software Design Maturity Model (SSDMM). The goal of this model is to assess and improve the maturity level of the secure software design process in software development organizations. To achieve the research objective, we implemented a Multivocal Literature Review (MLR) to identify the relevant studies in both the formal and grey literature. The proposed model was developed by utilizing the structure of the Capability Maturity Model Integration (CMMI) v2.0. SSDMM will assist organizations to evaluate their strengths and weaknesses in managing their secure software design practices and recommending the necessary improvement programs. It will also provide a foundation for researchers to develop new approaches to address the different security issues that are currently faced in secure software development projects. To achieve the objectives of this study, the following research questions are addressed:

- What are the practices for secure software design?
- What are the different knowledge areas for secure software design?
- How can we build a practical maturity model for secure software design?

The remainder of this paper is organized as follows. Section 2 presents a brief background and summary of the related work. Section 3 provides an extensive description of the research methodology. The results of the MLR are presented in Section 4. In Section 5, the structure and measurement guidelines are described in detail. Section 6 presents the case studies and their outcomes. Finally, the conclusions and future research directions are summarized in Section 7.

II. LITERATURE REVIEW

In this section, we review the literature related to software security and maturity models.

A. SOFTWARE SECURITY

Software security is defined in different ways in the literature. ISO/IEC 25010 [6] defined it as ''the degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization'' [6]. McGraw defined it as ''the ability of software to resist, tolerate, and recover from events that intentionally threaten its dependability'' [7]. According to McGraw, ''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 how to build secure things'' [8].

Software security is usually divided into sub-characteristics. ISO/IEC 25010 divided security into five sub-characteristics namely confidentiality, integrity, non-repudiation, accountability, and authenticity [6]. Confidentiality is defined as the degree to which a system ensures that data is accessible only to authorized users, while integrity is the degree to which the system prevents unauthorized data access or modification. Nonrepudiation is the degree to which system events can be proven to happen, while accountability is the degree to which the events can be traced. Authenticity is the degree to which a system user identity can be verified [6]. Other important security characteristics were highlighted by Jrgen [10]. These characteristics include role-based access control, fair exchange, secrecy, secure communication link, freshness, guarded access, and secure information flow [10]. Gorton also pointed out that a software system should encapsulate some security requirements including authentication, authorization, integrity, encryption, and non-repudiation [11].

Only a few metrics have been proposed in the literature to measure different aspects of software security. Alshammari *et al.* defined security metrics that can be derived from class diagrams [12], [13]. They proposed security metrics based on program coupling, inheritance, composition, extensibility, and design size from an information flow perspective. The proposed metrics enable security vulnerabilities to be found and fixed during the design phase and assist designers in comparing alternative designs in terms of security. Chowdhury *et al.* [14] proposed security metrics at the code level. They also provided guidelines for using these metrics and conducted case studies to show their applicability.

B. SECURE SOFTWARE DEVELOPMENT PROCESSES

Integrating security in the software development lifecycle has received attention from organizations and security experts in the software industry. Different processes have been developed to build security into the product using proactive approaches that take security into consideration throughout the development lifecycle [3], [7], [15], [16].

Microsoft introduced the Security Development Lifecycle (SDL) [15], which is a company-wide security assurance process for software development. This process introduces security and privacy throughout the development lifecycle, aiming to reduce the number of vulnerabilities in software products. It has been a mandatory policy at Microsoft since 2004 and is a collection of sixteen mandatory activities that are grouped into traditional SDLC phases.

McGraw [7] proposed seven touchpoints that represent the set of best practices in software: Security requirements, Abuse cases, Architectural risk analysis, Code review, Risk-based security tests, Penetration testing, and Security operations. These practices cover the whole development lifecycle, and they are based around the artifacts that are produced in the different development stages.

The Software Assurance Forum for Excellence in Code (SAFECode) published a paper that defines the fundamental practices in software development [16]. The first version of the document was published in 2008 and it has been revised twice. The reported practices represent the current practices of SAFECode members as guidelines to help organizations in the industry initiate or improve their security assurance programs.

In 2008, (ISC)2 launched the Certified Secure Software Lifecycle Professional (CSSLP) certification [3] for developers, analysts, engineers, and project managers who are involved in the application development lifecycle. The certification helps candidates handle software vulnerabilities and demonstrate their knowledge and expertise in software security. The curriculum of this certification focuses on software risks, vulnerabilities, and compliance issues that appear during the development lifecycle.

C. SECURITY IN SOFTWARE DEVELOPMENT PHASES

Jürgens [17] proposed an extension to the Unified Modeling Language (UML) called UMLsec to facilitate the modeling of security-critical systems. UMLsec enables the definition of security-related information within the system specification diagrams. Artelsmair *et al.* [18] developed a UML-based method called CoSMo (Conceptual Security Modeling) to integrate security policies into the software modeling process. Use cases were extended to define the security requirements as part of the software development process.

Fernandez [19] presented a methodology to develop secure software in which security policies are applied and verified at each development phase. In their methodology, security policies can be defined using the Object Constraint Language (OCL) and checked at the end of each stage. Howard [20] proposed a process to improve security in software applications by minimizing the code attack surface. He recommended the application of defensive practices in software development by utilizing the security features of Microsoft.NET framework 2.0. Manadhata and Wing [21] proposed metrics to measure the attack surface in software systems. They conducted different empirical studies on systems of different languages and sizes to validate the proposed method. They found that the proposed method can be useful in the different phases of the SDLC.

D. SOFTWARE MATURITY MODELS

Capability Maturity Model Integration (CMMI) [22] is a process model that was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University (CMU). The objective of this model is to enable organizations measure and improve their development processes and deliver high quality products. CMMI enables organizations to build, assess, and improve their processes and capabilities [23].

Different software maturity models for product maturity assessment have been proposed. The EuroScope consortium [24] developed a model to assess software product quality called the SCOPE Maturity Model (SMM). SMM was inspired by the Capability Maturity Model (CMM) and uses five maturity levels to evaluate software products. Al-Qutaish and Abran [25] proposed the Software Product Quality Maturity Model (SPQMM). The quality measures of a software product are first calculated using the characteristics and sub-characteristics defined in ISO 9126. Then, the calculated values are combined into a single value and converted into a six-sigma value that is placed within a maturity level. Alshayeb *et al.* [26], [27] proposed a framework to evaluate the maturity of software products called Technical-CMMI (T-CMMI). The product maturity level is determined by measuring the compliance of the product with the external and internal quality attributes that are specified in the user requirements.

Specialized maturity models have also been proposed. April *et al.* [28] proposed the Software Maintenance Maturity Model (SMmm), based on the CMMI, to assess and improve the quality of software maintenance activities. Golden *et al.* [29] introduced the Open Source Maturity Model (OSMM) to assess open-source products. OSMM can be used by IT companies as a method to evaluate and compare open-source software to guide the selection process. Alvaro *et al.* [30] proposed a maturity model called the Software Component Maturity Model (SCMM) to ensure the quality of component-based software. The model was used to evaluate software components to establish a certain level of quality and maturity. Eckert *et al.* [31] 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. The model was developed based on best practices from the literature and it was tested in a medical diagnostics organization.

Muñoz*et al.* [32] introduced a framework for operational software maturity. They studied the driving aspects of this concept in the aerospace industry and how it can be used to evaluate the operational behavior of software. A literature review and a case study were conducted to analyze the impact of operational software maturity on managing large software portfolios in the aerospace industry. The used assessment methodologies were found to be beneficial in maintenance activities and helped avoid operation disruption. Turetken *et al.* [33] developed a maturity model to assess the use of the Scaled Agile Framework (SAFe), which is used as a method for integrating agile software development practices in traditional large-scale projects. The model was developed by extending an existing agile maturity model to cover key SAFe practices.

A few security-related maturity models have been proposed. Al-Hanaei *et al.* [34] proposed a model to evaluate the maturity level of digital forensics capabilities of organizations. This model enables organizations to build improvement roadmaps in accordance with regulatory and business requirements. Bowen and Kissel [35] developed a tool to review and measure information security programs using a standardized approach. The output of this tool is a maturity-based scorecard that provides management with an indication of the level of a specific information security program. di Silva and de Barros [36] presented an information security maturity model for software developers based on ISO 27001. The model was evaluated by subject experts and utilized to measure the maturity level of several organizations.

E. SECURE SOFTWARE LIFECYCLE ASSESSMENT

Although there are many software maturity models, only a few methods have been developed to help organizations assess the maturity of their software development lifecycle. They emerged mainly from the software industry rather than academia, and the most related models are described in the following sub-sections.

1) SOFTWARE ASSURANCE MATURITY MODEL (SAMM)

The Open Web Application Security Project (OWASP) developed a non-commercial model in 2009 called the Software Assurance Maturity Model (SAMM) [41]. This is an open framework that helps software organizations set and implement their software security strategies. It provides a collection of resources that can aid an organization to evaluate its software security practices, build a balanced assurance program for software security, and demonstrate improvements to the assurance program.

SAMM was built with flexibility to be applied to organizations of different sizes and at different levels using any development style. It can be tailored based on organization risk tolerance and business processes. The framework also enables the iterative development of software security programs since it was designed in light of the fact that changes in an organization's behavior take place gradually over time. In addition, it provides well-defined and measurable steps for assurance program development and assessment as well as road map templates for common organization types.

SAMM is based on a collection of security practices that are organized into four business functions:

1. Governance: focuses on organization-level processes and activities related to the overall management of software development.
2. Construction: concerns the processes and activities related to software development including requirements gathering, software design and implementation, and product management.
3. Verification: entails the processes and activities related to software quality assurance such as testing and review.
4. Deployment: related to the processes and activities of software release management such as shipping, deployment and run time environment operations.

These functions are defined at the highest level representing the critical categories of activities that must be fulfilled by software development organizations. Under each business function, there are three security practices and each has a set of related activities that can be implemented to reduce security risks and improve software assurance. The list of practices was developed based on experience in software security.

Each security practice in SAMM can be improved independently through three progressive maturity levels. For each level, SAMM provides several prescriptive details including the objective statement, required activities, expected results and deliverables, success metrics, qualitative cost statements, required human resources and relevant levels within other practices. Each level is characterized by more sophisticated activities and stringent metrics than the preceding level.

2) BUILDING SECURITY IN MATURITY MODEL (BSIMM)

The Building Security In Maturity Model (BSIMM) [42] was developed by a team of leading software security experts in 2008. It quantifies the security practices of many organizations and provides common ground that enables organizations to compare their security initiatives with others.

BSIMM was developed based on experts' knowledge using a three-stage process starting with the creation of a software security framework. BSIMM provides a common vocabulary that describes all the initiatives of the different companies in a uniform way. In the second stage, face-to-face interviews were conducted with nine executives in charge of software security initiatives. Based on these interviews, a set of common activities was identified and organized according to the established framework. Finally, scorecards were created for the activities in each initiative. To validate this model, the participating firms were requested to review the framework, practices, and scorecards.

BSIMM is data-driven and it evolves over time. The activities and maturity levels are changed based on the observed real-world data. The latest version of BSIMM, version 10, includes data from 122 organizations in different vertical

markets including financial services, software vendors, technology, health care, cloud, Internet of Things, insurance, retail, telecommunications, security, and energy.

The software security framework of BSIMM 10 has a total of 119 activities. These activities are grouped into twelve practices under the following four domains:

1. Governance: focuses on practices that help in organizing, managing, and measuring the security initiatives such as staff development.
2. Intelligence: covers practices that are related to the corporate knowledge required to conduct the security activities such as organizational threat modeling.
3. SSDL Touchpoints: includes practices involved in the analysis and assurance of software development processes and artifacts in the Secure Software Development Lifecycle (SSDL).
4. Deployment: covers practices related to the software environment, configuration and maintenance.

The activities under each practice are grouped into three maturity levels. The assessment is conducted using in-person interviews to build a scorecard for the BSIMM 119 activities for the organization under evaluation. Based on the organization scorecard, a maturity level is determined for each of the twelve practices using a simple algorithm. If a level 3 activity is observed under a given practice, a maturity level of 3 is assigned to the practice even if lower level activities are not observed. Maturity levels of 2, 1, or 0 are assigned using the same logic. The maturity levels of all practices are presented using a spider chart.

F. SUMMARY

Recent research work in software security focused on providing technical solutions for security threats, however, there is lack of studies concerning how to incorporate security practices and processes into the various phases of the SDLC. Little research has been conducted to improve organizational maturity in secure software development. A better understanding of the issues related to organizational software development maturity helps to ensure successful project outcomes and to realize the potential benefits of security-driven software development.

Despite the importance of incorporating security in the software design phase, there is no maturity model that is focused on this phase. In addition, there are several approaches and techniques that have been developed by different researchers, but these have not been utilized by developers in industry. The existing industry maturity models do not consider the security practices and techniques that are developed in the literature. We believe that a better understanding of the issues related to organizational software design maturity while considering the literature and industry practices will help to ensure successful project outcomes.

III. RESEARCH METHODOLOGY

To achieve the objectives of this study, we followed a methodology that consists of five tasks and we collected data from the published literature and software industry (via the MLR process), as shown in the Figure 1. This evidence-based approach ensures the reliability of the collected data.

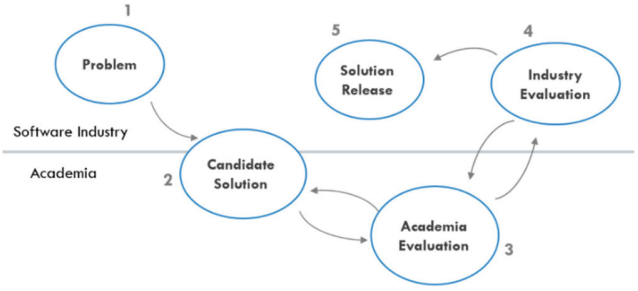

FIGURE 1. Research methodology.

In Task 1 of the research methodology, the software industry problem is identified and characterized through various approaches, such as SLR, exploratory case studies and process assessments. As previously mentioned, we identified several problems in our literature review to establish the objectives of this project.

In Task 2, we studied the current state-of-the-art in the literature and software industry to develop solutions for the identified problems. We conducted an MLR to collect evidence from the literature and industry regarding the development of various process areas and practices. We utilized the structure of CMMI v2.0 in the development of SSDMM.

After formulating a candidate solution, an initial validation in an academic setting (Task 3) was performed. Such initial validation provides rapid and valuable feedback and identifies obvious flaws, which can be addressed before the actual industry evaluation.

When the solution had been refined and updated based on the feedback within the academic environment, it was evaluated using case studies in industrial environments (Task 4). The objective of this step was to enable further refinement of the candidate solution. The case study approach is considered to be a useful evaluation method that can provide information about the real-world environment [43].

The final task (Task 5) is to release the solution in the form of publishable material. The released material will be used to transfer the knowledge to software development organizations and researchers in the field.

The approach that will be utilized to achieve the objectives of this research is illustrated in Figure 2 which summarizes each task under each objective.

A. MULTIVOCAL LITERATURE REVIEW

Systematic mapping (SM) and systematic literature reviews (SLRs) have become very popular in the field of software engineering. These studies are quite valuable since they provide a summary of the gaps and evidence in the primary studies of a specific research area. However, they focus only on formally published research work and ignore the large

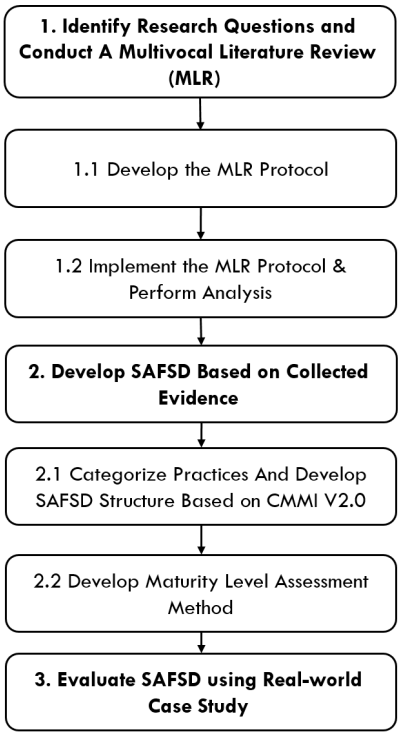

FIGURE 2. Research approach.

volume of "grey" literature which is generated by industry practitioners [44].

A Multivocal Literature Review (MLR) is a form of an SLR that includes both the formal literature that is produced by researchers in peer-reviewed journals or conferences as well as the grey literature that is produced by practitioners in different non-academic sources such as white papers, videos and Internet blogs. MLRs have been used by many researchers in other fields and have started to appear recently in some software engineering studies. Researchers believe that the adoption of MLRs in software engineering is important to identify research topics that emerge from the software industry [44].

In this study, we conducted an MLR to identify the state-of-the-art and state-of-the-practice software design security practices. The reason for adopting this approach is that a large body of knowledge which is related to the topic under study exists in both formal and grey literature. Grey literature covers a wide range of sources such as standards, technical reports and blogs that are not formally published in sources such as journal articles or books. We found from our preliminary literature review that there are many software security maturity models developed by different practitioners in industry and they need to be taken into consideration. An MLR enables us to collect enough evidence and fill the gap between academia and the software industry.

We developed our MLR process based on the guidelines proposed by Garousi *et al.* [44], as shown in Figure 3. This process is very similar to the typical SM and SLR process, the main difference being the inclusion and handling of grey

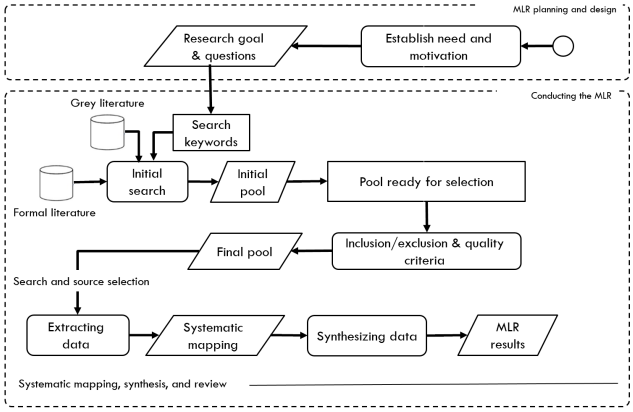

FIGURE 3. MLR process overview.

literature. It starts with the MLR planning and design phase in which the research goal and questions are formulated. Then, the phase of conducting the MLR is started. This phase consists of several steps, including the search strategy and source selection, development of the systematic map (classification scheme) and systematic mapping, synthesis, and review. Each of these steps is discussed in detail in the following sections.

1) RESEARCH GOAL AND QUESTIONS

The main goal of this study is to develop a security assessment framework for software design and implementation. This is done by conducting a systematic review and synthesis of the state-of-the-art and the state-of-the-practice in secure software design. Based on this goal, the following research questions are formulated.

- What are the practices for secure software design?
- What are the different knowledge areas for secure software design?

2) SEARCH STRATEGY

The first step in conducting the MLR is to retrieve and select the relevant sources for review. This involves the development of the search strategy, which is described in this section and defining the source inclusion/exclusion and quality assessment criteria, which are described in the following sections. The search strategy defines how to find relevant sources and involves several steps. We first derived the search terms by analyzing the keywords of the research questions from four perspectives: population, intervention, outcome of relevance and experimental design, as follows:

- Population: software security
- Intervention: existing maturity models or practices for secure software design and implementation
- Outcome of relevance: maturity models, practices, guidelines, recommendations, or checklists for secure software life cycle maturity, development, design, or building
- Experimental design: SLR, case studies, theoretical and empirical studies, expert opinions and observations, standards, and white papers.

We validated the derived search terms in the Google search engine and academic databases and identified their synonyms that are related to the topic. These synonyms are combined using Boolean operators, as shown in Table 1. Finally, after searching using different combinations of these keywords, we constructed the following search string for our study: (secure OR security) AND (software OR application) AND (develop OR design OR build OR architect OR implement) AND (maturity OR lifecycle OR ''life cycle'' OR practice OR guideline OR recommendation OR checklist).

TABLE 1. Keyword synonyms.

Keyword	Synonyms
"Secure software"	"secure software design OR secure software architecture OR secure software development OR building secure software OR secure application design OR secure application architecture OR secure application development OR building secure application"
"Practice"	"maturity OR lifecycle OR practices OR guidelines OR recommendations OR checklists"

The search string was applied to the following digital libraries:

- "Google (http://www.google.com)"
- "ACM Digital Library (http://dl.acm.org)"
- "Springer Link (http://link.springer.com)"
- "IEEE Xplore (http://ieeexplore.ieee.org)"
- "Science Direct (http://www.sciencedirect.com)"
- "John Wiley Online Library (http://onlinelibrary.wiley.com/)"

We tailored the search string to match the mechanisms of each library. To restrict the search space, we limited the search to titles and abstracts in databases that allow this option.

3) INCLUSION/EXCLUSION CRITERIA

The inclusion and exclusion criteria are defined to ensure that only relevant sources are selected for further review. Sources that meet the following criteria were selected:

- Sources which are relevant to secure software engineering
- Sources which focus on secure software practices
- Journal and conference papers, standards and white papers and reports published by reputable organizations

The following criteria was used to exclude sources that are not covered by our study:

- Sources which are not written in English or not fully accessible
- Sources that focus on a single technique or approach
- Textbooks
- Duplicate sources

4) QUALITY ASSESSMENT CRITERIA

The included sources were assessed for quality to ensure that they are valid and unbiased. The assessment of formal and grey literature sources was based on the criteria presented in Table 2 and Table 3, respectively. Our quality assessment criteria for grey literature is based on the checklist developed by [44].

TABLE 2. Quality assessment criteria of formal literature.

	Criteria
Q1	"Does the paper have a clearly stated goal?"
Q2	"Does the paper have a clearly reported methodology?"
Q3	"Does the study have empirical evaluation?"
Q4	"Are the research results clearly discussed?"
Q5	"Are the study limitations explicitly discussed?"
Q6	"Are the practices presented clearly?"

TABLE 3. Quality assessment criteria of grey literature.

	Criteria
Q1	"Is the source published by a reputable organization?"
Q2	"Does the source have a clear goal?"
Q3	"Does the source have a clear methodology?"
Q4	"Does the source have a stated date?"
Q5	"Does the source have a unique contribution?"
Q6	"Are the practices presented clearly?"

5) SOURCE SELECTION

The selection of the sources for this study was based on the criteria described above in three stages. In the first stage, the initial pool of relevant sources is gathered by reading the titles and abstracts found in the search results. In the second stage, the selection of relevant sources is made by reviewing the full text of the initially selected sources. Finally, the sources that meet the inclusion criteria and pass the quality assessment are selected for review.

6) DATA EXTRACTION AND SYNTHESIS

The sources identified in the final selection process were analyzed to collect the data required to answer our research question. We used qualitative analysis (coding) to synthesize the data and identify the security practices that are related to the software design phase.

B. ASSESSMENT FRAMEWORK DEVELOPMENT

The practices collected via the MLR were categorized into different knowledge areas and used to develop the SSDMM. The structure of this framework follows the structure of CMMI v2.0. It consists of four main categories namely Doing, Managing, Enabling, and Improving. Each of these categories has one or more group of practices, which we call knowledge areas.

C. CAPABILITY MATURITY MODEL INTEGRATION

CMMI provides a proven collection of global best practices that help organizations develop and benchmark their capabilities. CMMI has been utilized as a process improvement model by more than 5000 organizations around the world [37]. There are many motivations for an organization to adopt CMMI. These motivations are assessing the organization's capabilities against best practices, enhancing key capabilities, improving processes, providing clients with evidence of successful achievement and satisfying customer requirements [38].

The latest model from SEI is CMMI v2.0, released in 2018. This is a single model and has the following three views, which can be customized to meet the specific improvement needs of any business environment [39]:

1. CMMI Development: This is an integrated collection of best practices that enable organizations to enhance their capability to develop high quality products and services to meet customer needs. These practices improve product quality and the development life cycle and reduce development cost and time.
2. CMMI Services: This is a set of best practices that enable organizations to enhance their capability to deliver quality services and meet customer and market needs in an efficient and effective manner. These practices improve service quality and reduce service time and cost.
3. CMMI Supplier Management: This is a collection of best practices that enable organizations to enhance their capability to identify and manage vendors and suppliers in an effective manner. This helps in maximizing supply chain efficiency, meeting growth and product demands and minimizing risk.

The structure of CMMI v2.0 consists of the following four main categories, where each category is a logical collection of related capability areas [40]:

- Doing: capability areas for developing quality products
- Managing: capability areas for managing development of products
- Enabling: capability areas for supporting development of products
- Improving: capability areas for performance improvement

Each capability area has a set of related practice areas. A practice area is a collection of related practices that should be implemented together to achieve a specific goal and value. CMMI v2.0 has 25 practice areas that are organized into 12 capability areas.

CMMI enables organizations to measure their capabilities in terms of the maturity level of their processes. It provides an evolutionary path for organizations to progressively improve their processes though five levels. These levels range from 1, which represents a stage of chaotic and immature processes, to 5, which represents optimized and mature processes.

D. CONDUCTING A CASE STUDY

In the final phase, two case study were conducted in different organizations to evaluate SSDMM's usability and applicability. This helps in identifying hidden issues and closing the gap between academia and industry.

IV. RESULTS

This section summarizes the results of the MLR that was conducted to identify secure design practices. We first present a summary of the MLR search results with brief information about the selected sources. Then, we present the identified list of practices after MLR data synthesis.

A. SUMMARY OF MLR SEARCH RESULTS

As noted earlier, source selection was undertaken in three stages. The number of sources obtained by the initial selection process from all digital libraries is 432. Of these, 76 sources passed the second selection process and 47 passed the final selection process. These results are summarized in Table 4. Figure 4 shows the distribution of the publication channels for the selected sources.

TABLE 4. MLR search results.

Final Selection	Second review	First Review	Total Results	Resource
Google	200	97	41	31
IEEE Xplore	4,452	56	22	6
Springer	97,655	92	9	3
Science Direct	470,254	60	3	3
John Wiley	499,182	22	2	2
ACM	3,016	105	4	2
Total	1,074,759	432	76	47

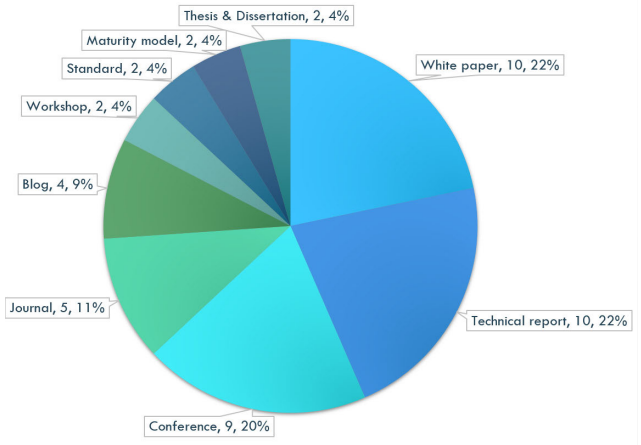

FIGURE 4. MLR sources.

Figure 5 presents the distribution of formally published sources along with those in the grey literature across the years. It can be seen that more attention has been given

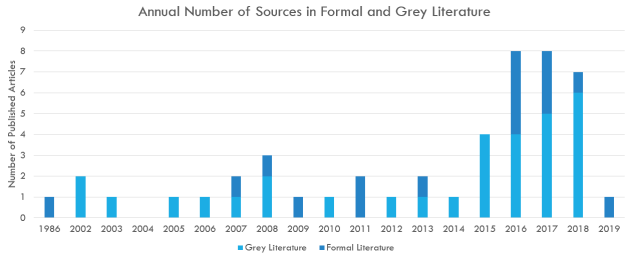

FIGURE 5. Annual number of sources in formal and grey literature between 1986 and 2019.

to software security practices in the grey literature, but the number of publications in the formal literature has started to increase.

B. SECURE SOFTWARE DESIGN PRACTICES

As a result of the MLR data synthesis, we identified a total of 71 security practices that are related to software design and are listed in Table 5 along with their frequency of occurrence. This list of practices represents the state-of-the-art and state-of-the-practice in secure software design. We have distributed and used these practices to build our assessment framework, which is presented in the next section.

V. SECURITY ASSESSMENT FRAMEWORK FOR SOFTWARE DESIGN

In this section, we describe the process of developing the Secure Software Design Maturity Model (SSDMM). This model is based on CMMI v2.0 and was developed based on the results obtained from our MLR. Figure 6 shows the process flow for SSDMM development. In the following section, we discuss the structure of SSDMM and its dimensions including knowledge areas and assessment method.

A. THE STRUCTURE OF SSDMM

The development of the SSDMM structure was motivated by the structure of CMMI v2.0 as CMMI is the most widely used model in the software industry. The security practices collected via the MLR are grouped into different classes. These correspond to the capabilities in CMMI v2.0 which we call knowledge areas, e.g., engineering security requirements, planning and managing security design activities. Like CMMI v2.0, the knowledge areas are organized into four high-level categories called Doing, Managing, Enabling and Improving. This structure is illustrated in Figure 7.

B. KNOWLEDGE AREAS AND PRACTICES

A total of 71 security practices were identified in the MLR. These were categorized into seven knowledge areas in SSDMM, as shown in Tables 6 - 13. We categorized these practices based on the knowledge we gained from the MLR and our personal experience in the field.

C. ASSESSMENT METHOD

The assessment method of SSDMM is based on the Motorola assessment tool [87]. This tool was developed at Motorola to assess the current status of an organization's software processes and identify areas for improvement [87]. Since then, it has been utilized by different researchers to measure the maturity levels in their proposed models. There are several reasons for selecting this tool including the following [88]:

- it has a small set of activities.
- it has been used and successfully tested by Motorola.
- it is designed as a normative instrument that can be adapted; and
- it assists in developing high quality products, increasing productivity, and saving cost and time.

The Motorola instrument uses the following three measurement dimensions [87]:

1. Approach: This dimension focuses on management support and the organization's commitment and ability to implement a specific practice.
2. Deployment: This dimension measures the consistency and coverage of practice deployment across the project areas.
3. Results: This dimension measures the consistency and breadth of the positive results across the project areas.

A practice is assessed by assigning a score value (0, 2, 4, 6, 8, or 10) to each dimension. The assigned score value

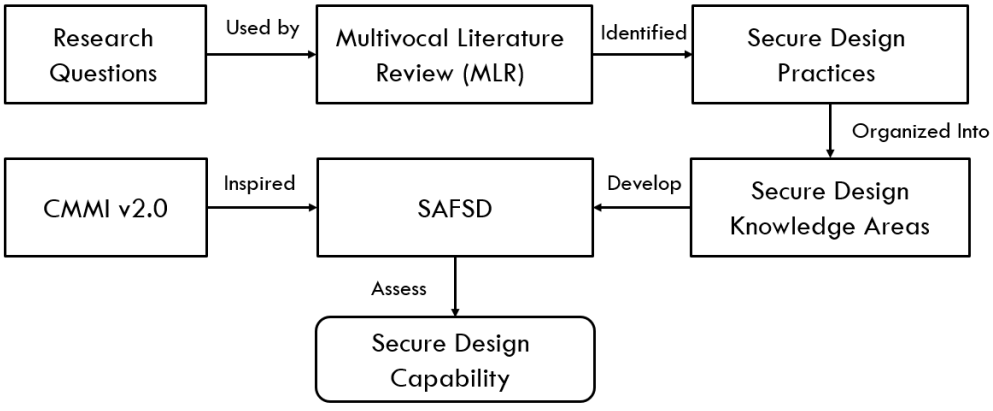

FIGURE 6. SSDMM development Process.

TABLE 5. Secure design practices identified in the MLR.

No.	Practice	Freq.	Source(s)
1	Establish design security requirements	2	[15], [45]
2	Perform threat modeling	24	[3], [15], [16], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58], [59], [60], [42], [61], [62], [63], [64], [65]
3	Determine threats in external modules	1	[46]
4	Develop data-flow diagrams	4	[55], [56], [66]
5	Use a scheme to classify applications based on data confidentiality	3	[42], [60], [65]
6	Perform design and architecture security risk analysis	15	[58], [61], [65] , [66], [67], [68], [69], [70], [50], [71], [72], [73], [74], [75]
7	Evaluate risk from third-party components	5	[56], [54], [67], [74], [76]
8	Security specification review	2	[67], [71]
9	Minimize software attack surface or access points	8	[3], [15], [45], [49], [57], [70], [75]
10	Identify and segregate trusted entities from untrusted entities	8	[3], [46], [55], [58], [67], [70], [71], [77]
11	Assume environment is not trustworthy	2	[70], [78]
12	Create an architecture inventory	3	[3], [46], [58]
13	Build security architecture	2	[3], [65]
14	Document security controls	2	[49], [54], [58]
15	Make sure that the design specification is traceable to requirements	2	[67], [70]
16	Build and maintain abuse-case models and attack patterns	4	[42],[56], [61], [70]
17	Elaborate threat models with compensating controls	1	[56]
18	Design security features using diagrams	2	[67], [68]
19	Consider security principles in design	14	[5], [16], [45], [49], [55], [56], [57], [61], [62], [65], [70], [72], [74],[79]
20	Minimize or eliminate unnecessary functionality	3	[67], [71],[80]
21	Design simple user interface	2	[67], [71]
22	Design audit logging features and hold all actors accountable	9	[16], [49], [56], [64], [70], [73], [77], [81], [82]
23	Avoid timing, synchronization and sequencing issues	1	[70]
24	Use only safe interfaces to environment resources	1	[70]
25	Design access control features	11	[16], [42], [49], [56], [64], [73], [77], [80], [81], [83], [84]
26	Design data encryption features	9	[16], [42], [56], [64], [73], [77], [80], [81], [85]
27	Design input validation features	6	[42], [56], [64], [73], [77], [80]
28	Design secure exception handling features	3	[64] , [67], [80]
29	Manage user sessions	2	[80], [84]
30	Make use of security design patterns	2	[62], [65]
31	Control open source risk	2	[42], [76]
32	Use single point of entry for all users	1	[84]
33	Secure access layer and access mechanisms	1	[84]
34	Architecture defense-in-depth	1	[85]
35	Conduct design and architecture security review	18	[16],[42], [48], [49], [51], [54], [56], [61], [62], [63], [64], [65], [67], [71], [72], [80], [81], [82],
36	Analyze design against known security requirements or feature	5	[42], [53], [56], [59], [66]
37	Validate usage of frameworks, patterns and platforms	1	[56]
38	Have Software Security Group (SSG) lead design review efforts	1	[42]
39	Conduct architecture peer review	1	[69]
40	Establish quality gates for design review	4	[56], [67], [70], [74]
41	Secure design and architecture documentation	2	[63], [80]
42	Track third party components	2	[55], [76]

TABLE 5. *(Continued.)* **Secure design practices identified in the MLR.**

43	Build attack patterns for specific technologies	1	[42]
44	Maintain a list of possible attacks	2	[42], [46]
45	Notification of third-party software threats and vulnerabilities	1	[55]
46	Security knowledge training and threat awareness	5	[16], [42], [45], [55], [56],
47	Create a team to develop new attacks	1	[42]
48	Create capability to address difficult design issues	1	[42]
49	Identify potential attackers and develop attacker profiles	2	[42], [56]
50	Gather and publish historical information about attacks	1	[42]
51	Create an internal forum for discussions about attacks	1	[42]
52	Use automation to simulate attacks	1	[42]
53	Establish and maintain secure configuration management practices	3	[64], [67], [80]
54	Perform comparative security assessments of different integration options	1	[67]
55	Establish a process for architectural analysis	1	[42]
56	Standardize architectural descriptions	1	[42]
57	Build common security frameworks, or libraries	2	[42], [56]
58	Identify secure design patterns across the organization	2	[42], [56]
59	Maintain a list of recommended software frameworks	2	[5], [56]
60	Use proven security technologies, frameworks and features	4	[42], [49], [55], [67]
61	Publish common security controls or services	1	[42]
62	Standardize identity and access management	1	[16]
63	Deploy design review service for project teams	1	[56]
64	Form a committee or board to review and manage secure design patterns	1	[42]
65	Design reviews are led by software architects	1	[42]
66	Engage specialized security team during design and architecture	2	[42], [81]
67	External review of the design	3	[5], [50], [69]
68	Use threat modeling tool	4	[15], [55], [56], [86]
69	Use threat weighting or ranking during threat modeling	3	[47], [56], [59]
70	Use a risk questionnaire to rank applications during architectural analysis	1	[42]
71	Use design and architecture tools to enhance security	1	[65]

corresponds to the performance level in a specific dimension and can be determined based on the criteria presented in Table 14. To assess the practices under a specific knowledge area, the following steps should be followed:

1. Step 1: For each practice, calculate the average of the 3-dimensional scores and round it to a whole number.
2. Step 2: Add the averages obtained in Step 1 and divide the result by the number of practices. This gives the overall score for the knowledge area.
3. Step 3: The knowledge area score is considered weak if it is less than 7 and strong if it is 7 or higher.

VI. CASE STUDY

A case study is a powerful evaluation approach that provides enough information about the real-world environment [89]. The case study is an appropriate research method in this study because SSDMM has been developed for the software industry. In this research, we conducted two case studies with industry to evaluate SSDMM. The main objectives for conducting the case studies in this research are to:

- show that SSDMM can be used in a real-world environment
- show the practicality of using SSDMM

To conduct the case study, we communicated with employees from different software organizations, introduced the concept of SSDMM to them and invited them to participate in our study. They were asked to use SSDMM to assess their software design security practices based on SSDMM using the Motorola assessment tool. They carried out the assessment in their workplace and submitted the results along with their feedback via email.

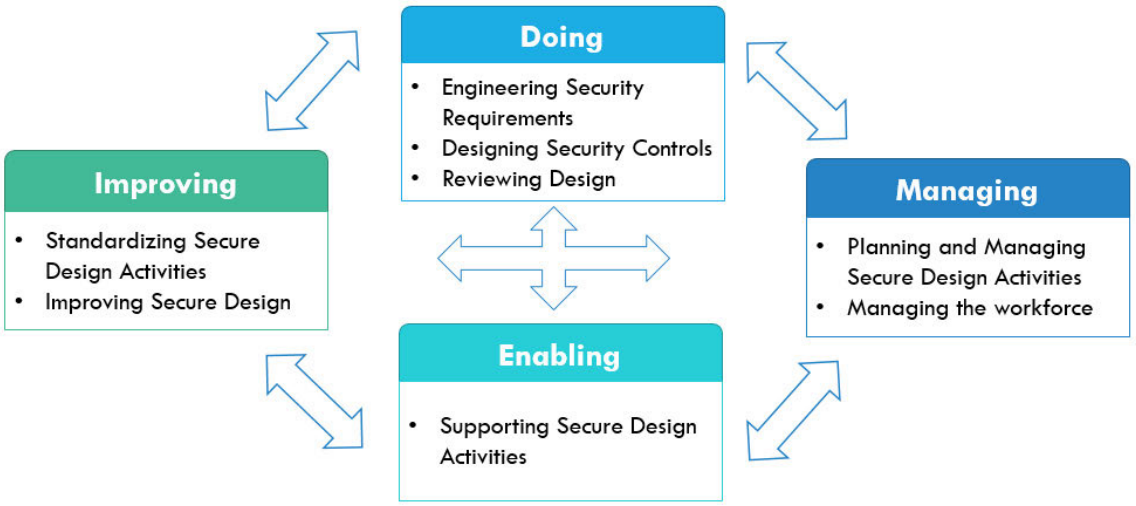

FIGURE 7. SSDMM structure.

TABLE 6. Engineering security requirements.

Doing	
Engineering Security Requirements	
No.	**Practice**
1	Establish design security requirements
2	Perform threat modeling
3	Determine threats in external modules
4	Develop data-flow diagrams
5	Use a scheme to classify applications based on data confidentiality
6	Perform design and architecture security risk analysis
7	Evaluate risk from third-party components
8	Security specification review
9	Minimize software attack surface or access points
10	Identify and segregate trusted entities from untrusted entities
11	Assume environment is not trustworthy
12	Create an architecture inventory
13	Build security architecture
14	Document security controls
15	Make sure that the design specification is traceable to requirements
16	Build and maintain abuse-case models and attack patterns

To minimize the threat of company size, in our case study, we targeted one large and one small size company. Organization A is a software development company located in Saudi Arabia with around 200 professionals. It has many customers and provides different solution services including data analytics, tracking, reporting and mobile solutions. The respondent from this organization is a senior software developer with 15 years of experience. Organization B is a small company specializing in mobile software development. The organization is in Saudi Arabia and has a team of around 20 developers. The respondent from this organization has two years of experience in software development.

A. ASSESSMENT RESULTS FOR ORGANIZATION A

Figure 8 shows the assessment results of each practice and the overall score of each knowledge area is presented for organization A. The following are the key points identified in this assessment considering the guidelines of the Motorola assessment tool:

- The organization is strong in the knowledge areas of 'design review' and 'standardizing secure design' activities since it achieved a score greater than 7.
- The organization is weak in all other knowledge areas since the score values are less than 7.
- The organization has good performance in most practices in terms of management commitment and

TABLE 7. Designing security controls.

Doing	
Designing Security Controls	
No.	**Practice**
1	Elaborate threat models with compensating controls
2	Design security features using diagrams
3	Consider security principles in design
4	Minimize or eliminate unnecessary functionality
5	Design simple user interface
6	Design audit logging features and hold all actors accountable
7	Avoid timing, synchronization and sequencing issues
8	Use only safe interfaces to environment resources
9	Design access control features
10	Design data encryption features
11	Design input validation features
12	Design secure exception handling features
13	Manage user sessions
14	Make use of security design patterns
15	Control open source risk
16	Use single point of entry for all users
17	Secure access layer and access mechanisms
18	Architecture defense-in-depth

TABLE 8. Reviewing design.

Doing	
Reviewing Design	
No.	**Practice**
1	Conduct design and architecture security review
2	Analyze design against known security requirements or feature
3	Validate usage of frameworks, patterns and platforms
4	Have Software Security Group (SSG) lead design review efforts
5	Conduct architecture peer review

TABLE 9. Planning and managing secure design activities.

Managing	
Planning and Managing Secure Design Activities	
No.	**Practice**
1	Establish quality gates for design review
2	Secure design and architecture documentation
3	Track third party components
4	Build attack patterns for specific technologies
5	Maintain a list of possible attacks
6	Notification of third-party software threats and vulnerabilities

deployment, but it needs to focus more on measuring the effectiveness of practice implementation. This will help in improving its maturity level in secure software design.

B. ASSESSMENT RESULTS FOR ORGANIZATION B

Figure 9 shows the assessment results of each practice and the overall score of each knowledge area are presented for organization. The following are the key points identified in this assessment:

- The organization is strong in the knowledge area of 'designing security controls' since it achieved a score greater than 7.
- The organization is weak in all other knowledge areas since the score values are less than 7.

TABLE 10. Managing the workforce.

Managing	
Managing the Workforce	
No.	**Practice**
1	Security knowledge training and threat awareness
2	Create a team to develop new attacks
3	Create capability to address difficult design issues

TABLE 11. Supporting secure design activities.

Enabling	
Supporting Secure Design Activities	
No.	**Practice**
1	Identify potential attackers and develop attacker profiles
2	Gather and publish historical information about attacks
3	Create an internal forum for discussions about attacks
4	Use automation to simulate attacks
5	Establish and maintain secure configuration management practices
6	Perform comparative security assessments of different integration options

TABLE 12. Standardizing secure design activities.

Improving	
Standardizing Secure Design Activities	
No.	**Practice**
1	Establish a process for architectural analysis
2	Standardize architectural descriptions
3	Build common security frameworks, or libraries
4	Identify secure design patterns across the organization
5	Maintain a list of recommended software frameworks
6	Use proven security technologies, frameworks and features
7	Publish common security controls or services
8	Standardize identity and access management

TABLE 13. Improving secure design.

Improving	
Improving Secure Design	
No.	**Practice**
1	Deploy design review service for project teams
2	Form a committee or board to review and manage secure design patterns
3	Design reviews are led by software architects
4	Engage specialized security team during design and architecture
5	External review of the design
6	Use threat modeling tool
7	Use threat weighting or ranking during threat modeling
8	Use a risk questionnaire to rank applications during architectural analysis
9	Use design and architecture tools to enhance security

- The organization needs to enhance its performance in all those practices with a score value of less than 7. It needs to focus more on the practices that are related to workforce management, supporting activities and process improvement. This will help in achieving a higher maturity level in secure software design.

TABLE 14. Motorola measurement tool.

Score	Approach	Deployment	Results
Poor (0)	Management does not recognize the need (OR) Organization does not have commitment (OR) Organization does not have the ability	Organization does not have the practice (OR) Organization does not show interest	Ineffective
Weak (2)	Management starts to recognize the need (OR) Support for the practices has recently started (OR) Part of the organization is able to undertake the practice	Usage is scattered (OR) Practice is implemented in some parts of the organization (OR) The practice is only undergoing monitoring/verification	Results are inconsistent (OR) Some effectiveness is shown in some parts of the organization
Fair (4)	Wide but not full management commitment (OR) A plan for the practice implementation is defined (OR) Several supporting items are in place	Usage is less fragmented (OR) Practice is implemented in some important parts of the organization (OR) Practice is undergoing monitoring/verification in several parts of the organization	Positive and consistent results in some parts of the organization (OR) Inconsistent results in other parts of the organization
Marginally qualified (6)	Some management commitment (OR) Some management members are proactive (OR) Implementation is underway in some parts of the organization (OR) Practices supporting items are in place	Implemented in some parts of the organization (OR) Consistent usage across many parts of the organization (OR) Practice is undergoing monitoring/verification in many parts of the organization	Positive quantifiable results in most parts of the organization (OR) Consistent positive results across many parts of the organization
Qualified (8)	Full management commitment (OR) Most management members are proactive (OR) Practice implementation is integrated with the process (OR) Practice improvement is motivated by supporting items	Implemented in almost all parts of the organization (OR) Consistent implementation across almost all parts of the organization (OR) Practice is under monitoring/verification across almost all parts of the organization	Positive quantifiable results in almost all parts of the organization (OR) Consistent positive outcomes across almost all parts of the organization
Outstanding (10)	Management has leadership and commitment (OR) Recognition of excellence outside the organization	Consistent implementation across all parts of the organization (OR) Practice implementation is monitored/verified across the organization	Required results are exceeded (OR) World-class results are consistently achieved (OR) Consultancy sought by others
Score	**Approach**	**Deployment**	**Results**

- The results show that the organization is doing very little in managing the workforce and supporting secure design activities and this is an area for improvement.

C. FEEDBACK FROM CASE STUDY PARTICIPANTS

The case study enabled us to evaluate the practicality of SSDMM in a real-world environment. The following is a summary of the feedback given by the case study participants: the participants were able to use it without guidance regardless of their experience level.

1. Most of the practices in SSDMM are self-explanatory. The participants requested clarification and examples for only few of them because they did not have an in-depth awareness of software security.
2. SSDMM provided the participants with information about the maturity level of their current process as well as new knowledge about security practices in software design. Hence, SSDMM not only helps in assessing the current software design process in terms of security, it can also increase the awareness level and help developers improve their process.
3. The most difficult part of SSDMM was the assessment process. It took some time for the assessor to understand the criteria of the Motorola assessment tool to apply it correctly and obtain accurate results. An enhancement of the SSDMM assessment method is needed.

D. THREATS TO VALIDITY

To increase confidence in the results of our study, this section highlights the threats to validity and how they are overcome. The threats are classified according to [90] into construct, internal, external and conclusion validity threats.

1) CONSTRUCT VALIDITY

One of the potential threats is missing sources which were published in other databases or after conducting the research. To maximize the coverage, we utilized an MLR to collect all the relevant articles that had been published in the formal and grey literature and our search covered a total of six electronic databases. This ensured comprehensive coverage of all the related sources. Thus, we have sufficient evidence that SSDMM covers most of the security practices in software design.

Another limitation is that assigning the practices to the knowledge areas while building SSDMM is sometimes subjective. We conducted the assignment based on our personal experience and the experience we gained from the MLR. To minimize this threat, the structure of SSDMM and its knowledge areas and practices were iteratively validated by three university professors.

2) INTERNAL VALIDITY

The practices that we obtained as a result of our MLR are a good representation of the actual existing practices in secure software design. However, source selection and data

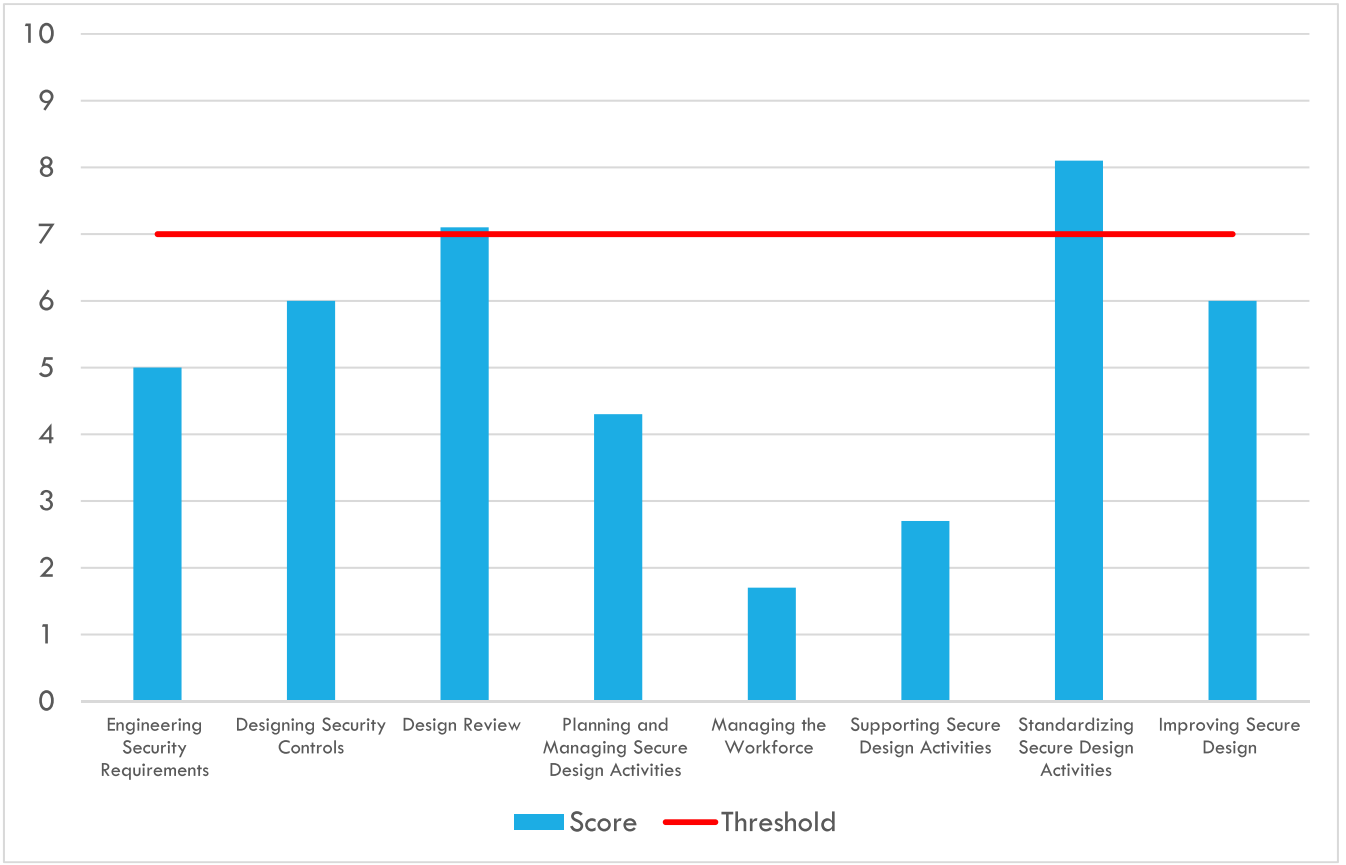

FIGURE 8. Assessment results for Organization A.

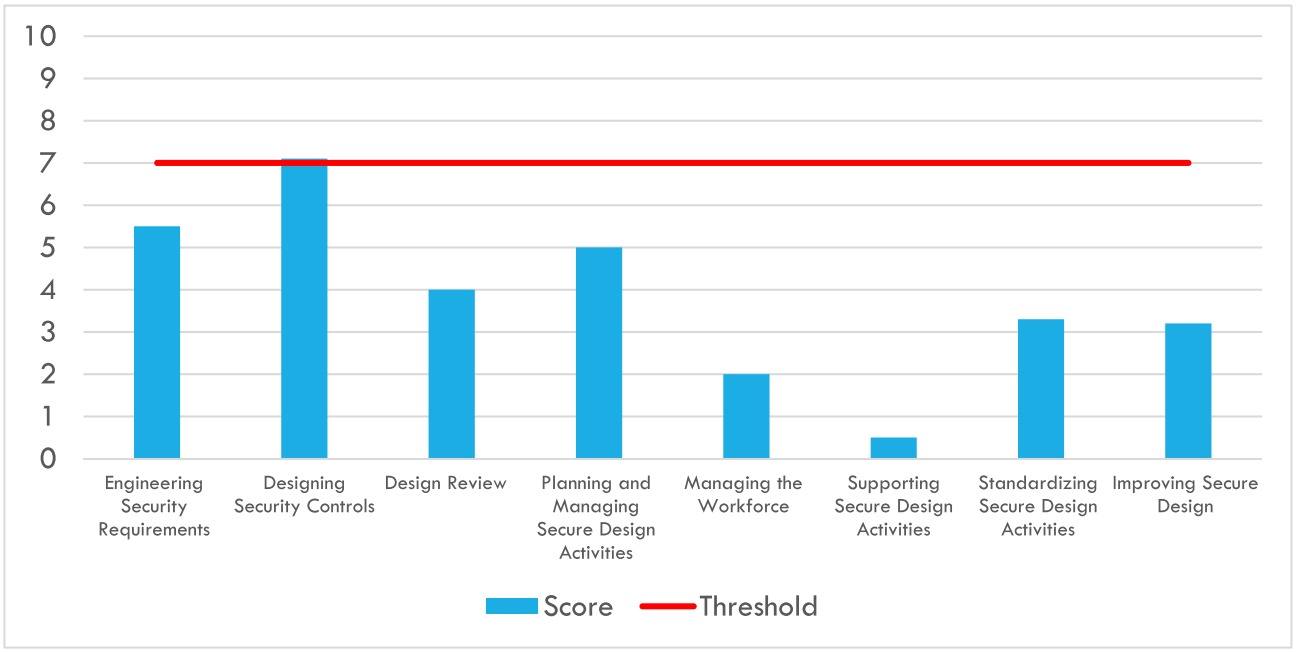

FIGURE 9. Assessment results for Organization B.

extraction in the MLR are subjective because some sources do not have sufficient or clear information related to our research questions. To overcome this threat, the sources were screened carefully and selected based on a set of quality criteria.

Since the assessment of the organization's practices in our case study was carried out by the participants themselves, we are concerned about the accuracy of the assessment results. The assessment might be subjective since following the criteria of the Motorola assessment tool requires careful attention from the assessor to obtain accurate results. It also depends on the participants' experience with software security and their position in the organization.

3) EXTERNAL VALIDITY

The results and conclusions of this study are not guaranteed to be applicable to all software development organizations. The case studies we used to evaluate SSDMM were conducted in only two organizations. Thus, the conclusions regarding SSDMM applicability should be generalized with caution.

4) CONCLUSION VALIDITY

By conducting the MLR, we collected enough evidence to support our conclusions about the existing practices and knowledge areas of secure software design. The MLR was conducted carefully in a systematic way and the sources were evaluated based on defined quality criteria. This increases the level of our confidence in SSDMM and reduces the threat to conclusion validity.

VII. CONCLUSION AND FUTURE WORK

This section summarizes the work and major contributions of our study. It also presents the recommended future directions for researchers.

A. CONCLUSION

The main goal of this research is to develop the Secure Software Design Maturity Model (SSDMM) to help organizations evaluate and improve their security practices in the design phase. This framework was built based on evidence collected from both research and practitioner communities through a Multivocal Literature Review (MLR).

The MLR was used to consolidate the state-of-the-art and state-of-the practice secure design practices in a systematic approach. We reviewed 38 sources selected from 6 databases and identified 65 practices. These practices were then categorized into different knowledge areas that constituted the SSDMM.

The SSDMM was developed based on the structure of CMMI v2.0. It consists of 8 knowledge areas organized into 4 categories. Each knowledge area has a set of practices that should be followed to cover the area. We used the Motorola assessment tool as measurement guidelines for this framework. A technical validation of the framework was undertaken by three university professors and it was revised to accommodate their feedback.

To evaluate the applicability of SSDMM, we conducted case studies in two software development organizations. The outcomes of these studies confirmed the usability of our framework in a real-world environment.

This work is expected to help software organizations increase the security level of their products and improve their performance in terms of security. They can use SDSMM to assess the maturity of their security practices in software design. This will also increase developers' level of awareness about secure design practices. Moreover, this study provides guidelines for researchers in software security, which are summarized in the next section.

B. FUTURE RESEARCH DIRECTIONS

With the growing number of software security threats, there is a need for the continuous improvement of software security processes and practices. This research work can be enhanced in different ways. The following are the open research directions that can be explored by researchers:

1. Collaboration with software development organizations is needed to enhance the outcomes of SSDMM. It could be customized for different organizations' needs depending on the facilities and adopted mechanisms.
2. SSDMM could be extended to have specific characteristics that are related to specific technologies such as the Internet of Things (IoT), blockchain and cloud computing.
3. SSDMM could be published as an online repository and updated continuously with recent practices from academia and industry. This will make SSDMM a reliable reference for researchers and practitioners.

REFERENCES

[1] G. McGraw, "Software security," *Datenschutz und Datensicherheit*, vol. 36, no. 9, pp. 662–665, 2012.
[2] T. Okubo and H. Tanaka, "Identifying security aspects in early development stages," in *Proc. 3rd Int. Conf. Availability, Rel. Secur.*, Mar. 2008, pp. 1148–1155.
[3] S. Ahmed, "Secure software development: Identification of security activities and their integration in software development lifecycle," M.S. thesis, School Eng., Dept. Syst. Softw. Eng., Blekinge Inst. Technol., Karlskrona, Sweden, 2007.
[4] 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.
[5] 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.
[6] *Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation*, Standard ISO/IEC 25010:2011: 2011.
[7] G. McGraw, *Software Security: Building Security*. Reading, MA, USA: Addison-Wesley, 2006.
[8] G. McGraw, "Software security," *IEEE Security Privacy*, vol. 2, no. 2, pp. 80–83, Mar. 2004.
[9] S. Lipner and M. Howard, *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Albuquerque, NM, USA: Microsoft, 2006.
[10] J. Järjens, *Secure Systems Development with UML*. Cham, Switzerland: Springer, 2005.
[11] I. Gorton, *Essential Software Architecture*. Berlin, Germany: Springer, 2006.
[12] B. Alshammari, C. Fidge, and D. Corney, "Security metrics for object-oriented class designs," in *Proc. 9th Int. Conf. Qual. Softw.*, Aug. 2009, pp. 11–20.

[13] B. Alshammari, C. Fidge, and D. Corney, "Security metrics for object-oriented designs," in *Proc. 21st Austral. Softw. Eng. Conf.*, 2010, pp. 55–64.
[14] I. Chowdhury, B. Chan, and M. Zulkernine, "Security metrics for source code structures," in *Proc. 4th Int. workshop Softw. Eng. Secure Syst.*, 2008, pp. 57–64.
[15] Microsoft. (2018). *Microsoft Security Development Lifecycle*. [Online]. Available: https://www.microsoft.com/en-us/securityengineering/sdl/
[16] SAFECode. (2018). *Fundamental Practices for Secure Software Development*. [Online]. Available: https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf
[17] J. Järjens, "UMLsec: Extending UML for secure systems development," in *Proc. 5th Int. Conf. The Unified Modeling Lang.*, vol. 2460. London, U.K.: Springer-Verlag, 2002, pp. 412–425.
[18] C. Artelsmair, W. Essmayr, P. Lang, R. Wagner, and E. Weippl, "CoSMo: An approach towards conceptual security modeling," in *Database and Expert Systems Applications*. Berlin, Germany: Springer, 2002, pp. 557–566.
[19] E. B. Fernandez, "A methodology for secure software design," in *Proc. Softw. Eng. Res. Pract.*, 2004, pp. 130–136.
[20] M. Howard, "Attack surface: Mitigate security risks by minimizing the code you expose to untrusted users," Microsoft MSDN Mag., Tech. Rep., 2004.
[21] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 371–386, May 2011.
[22] *CMMI for Development, Version 1.3*, document CMU/SEI-2010-TR-033, 2010.
[23] *CMMI for Development, Version 1.3*, document CMU/SEI-2010-TR-033, 2010.
[24] 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*, 1999, pp. 1–5.
[25] 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.
[26] M. Alshayeb, A. K. Abdellatif, S. Zahran, and M. Niazi, "Towards a framework for software product maturity measurement," in *Proc. 10th Int. Conf. Softw. Eng. Adv.*, Barcelona, Spain, 2015, pp. 7–11.
[27] 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.
[28] A. April, J. Huffman Hayes, A. Abran, and R. Dumke, "Software maintenance maturity model (SMmm): The software maintenance process model," *J. Softw. Maintenance Evolution: Res. Pract.*, vol. 17, no. 3, pp. 197–223, 2005.
[29] B. Golden, *Succeeding with Open Source*. Reading, MA, USA: Addison-Wesley, 2005.
[30] A. Alvaro, E. Santana de Almeida, and S. Lemos Meira, "A software component maturity model (SCMM)," in *Proc. 33rd EUROMICRO Conf. Softw. Eng. Adv. Appl.*, Aug. 2007, pp. 83–92.
[31] 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*, 2017, pp. 1–4, doi: 10.1145/3125433.3125447.
[32] R. Muñoz, E. Shehab, M. Weinitzke, C. Fowler, and P. Baguley, "Operational software maturity: An aerospace industry analysis," *Int. J. Comput., Electr., Autom., Control Inf. Eng.*, vol. 11, pp. 931–940, 2017.
[33] O. Turetken, I. Stojanov, and J. J. M. Trienekens, "Assessing the adoption level of scaled agile development: A maturity model for scaled agile framework," *J. Softw., Evol. Process*, vol. 29, no. 6, p. e1796, Jun. 2017.
[34] E. H. Al Hanaei and A. Rashid, "DF-C2M2: A capability maturity model for digital forensics organisations," in *Proc. IEEE Secur. Privacy Workshops*, May 2014, pp. 57–60.
[35] P. Bowen and R. Kissel, "Program review for information security management assistance (PRISMA)," Inf. Technol. Lab., Comput. Secur. Resource Center, Nat. Inst. Standards Technol., Tech. Rep. NISTIR 7358, 2007. [Online]. Available: http://csrc.nist.gov/publications/nistir/ir7358/NISTIR-7358.pdf
[36] 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.
[37] *CMMI for Development, Version 2.0*, CMMI Inst., 2018.
[38] M. Alshayeb, A. Abdullatif, S. Zahran, and M. Niazi, "Method, apparatus and non-transitory computer readable media for the assessment of software products," U.S. Patent 9 558 098, Jan. 31, 2017.
[39] CMMI Institute. (Apr. 12, 2019). *Introducing CMMI v2.0*. [Online]. Available: https://cmmiinstitute.com/cmmic
[40] (2018). *Introducing CMMI Development v2.0 to Pan-India Spin*. [Online]. Available: https://cmmiinstitute.com/getattachment/5b16e5f1-acd2-4b86-aa4c-c9bc353a669f/attachment.aspx.
[41] OpenSAMM. (Apr. 12, 2020). *Software Assurance Maturity Model (SAMM): A Guide to Building Security Into Software Development*. [Online]. Available: http://www.opensamm.org/
[42] BSIMM. (2018). *Building security in maturity model (BSIMM)*. [Online]. Available: https://www.bsimm.com
[43] R. Yin, *Case Study Research: Design and Methods*, 5th ed. Newbury Park, CA, USA: Sage, 2016.
[44] V. Garousi, M. Felderer, and M. V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," *Inf. Softw. Technol.*, vol. 106, pp. 101–121, 2019/02/01/ 2019.
[45] Opentext. (2017). *Product security assurance program*. [Online]. Available: https://www.opentext.com/file_source/OpenText/en_US/PDF/opentext-product-security-assurance-program.pdf
[46] (2019). *Framework Secure Software*. [Online]. Available: https://securesoftwarealliance.org/pilots-with-framework-secure-software/
[47] IBM. (2018). *Security in Development: The IBM Secure Engineering Framework*. [Online]. Available: https://www.redbooks.ibm.com/redpapers/pdfs/redp4641.pdf
[48] A. Rahman and L. Williams, "Software security in DevOps: Synthesizing practitioners' perceptions and practices," in *Proc. Int. Workshop Continuous Softw. Evol. Delivery*2016, pp. 70–76.
[49] M.-L. Sánchez-Gordón, R. Colomo-Palacios, A. Sánchez, A. de Amescua Seco, and X. Larrucea, "Towards the integration of security practices in the software implementation process of ISO/IEC 29110: A mapping," in *Proc. Syst., Softw. Services Process Improvement*, Cham, Switzerland, 2017, pp. 3–14:.
[50] A. Alkussayer and W. H. Allen, "The ISDF framework: Integrating security patterns and best practices," in *Advances in Information Security and Its Application*. Berlin, Germany: Springer, 2009, pp. 17–28.
[51] McAfee. (2018). *McAfee Product Security Practices*. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/misc/ms-product-software-security-practices.pdf
[52] WhiteHatSecurity. (2019). *Integrating Application Security Into The Mobile Software Development Lifecycle*. [Online]. Available: https://www.whitehatsec.com/wp-content/uploads/2016/01/Mobile_SDLC_White_Paper.pdf
[53] VMware. (2010). *Vmwares Security Programs and Practices*. Available:. [Online]. Available: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/whitepaper/vmware-product-security-white-paper.pdf
[54] SAP. (2019). *The Secure Software Development Lifecycle at SAP*. [Online]. Available: https://www.sap.com/documents/2016/03/a248a699-627c-0010-82c7-eda71af511fa.html
[55] Cisco. (2016). *Cisco Secure Development Lifecycle*. Available:. [Online]. Available: https://www.cisco.com/c/dam/en_us/about/doing_business/trust-center/docs/cisco-secure-development-lifecycle.pdf
[56] OWASP. (2017). *Software Assurance Maturity Model*. [Online]. Available: https://owaspsamm.org/
[57] OWASP. (2018). *Security by Design Principles*. [Online]. Available: https://blog.threatpress.com/security-design-principles-owasp/
[58] OITS. (2017). *Secure System Development Life Cycle*. [Online]. Available: https://its.ny.gov/sites/default/files/documents/nys-s13-001_secure_system_development_life_cycle_1.pdf
[59] R. Seacord. (2018). *Top 10 Secure Coding Practices*. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices?focusedCommentId=88044413
[60] P. Morrison, B. H. Smith, and L. Williams, "Measuring security practice use: A case study at IBM," in *Proc. IEEE/ACM 5th Int. Workshop Conducting Empirical Stud. Ind. (CESI)*, May 2017, pp. 16–22.
[61] K. Rindell, S. Hyrynsalmi, and V. Leppänen, "Busting a myth: Review of agile security engineering methods," in *Proc. 12th Int. Conf. Availability, Rel. Secur.*, New York, NY, USA, 2017, pp. 1–10, Art. no. 74.
[62] P. H. Meland and J. Jensen, "Secure software design in practice," in *Proc. 3rd Int. Conf. Availability, Rel. Secur.*, Mar. 2008, pp. 1164–1171.
[63] R. Khan, "Secure software development: A prescriptive framework," *Comput. Fraud Secur.*, vol. 2011, no. 8, pp. 12–20, Aug. 2011.
[64] Microsoft. (2005). *Security Engineering Explained*. [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=20528
[65] ISC. (2017). *CSSLP certification Exam Outline*. [Online]. Available: https://www.isc2.org/exam-outlines

[66] D. Fischer, M. Sarkarati, M. Spada, T. Michelbach, W. Urban, and C. Tueffers, "An Application Security Framework for SOA-Based Mission Data Systems," in *2011 IEEE 4th Int. Conf. Space Mission Challenges Inf. Technol.*, vol. 2011, pp. 53–60.
[67] K. M. Goertzel, T. Winograd, H. L. McKinley, P. Holley and B. A. Hamilton. (2006). *Security in the Software Life Cycle*. [Online]. Available: https://resources.sei.cmu.edu/asset_files/WhitePaper/2006_019_001_52113.pdf
[68] B. Subedi, A. Alsadoon, P. W. C. Prasad, and A. Elchouemi, "Secure paradigm for Web application development," in *Proc. 15th RoEduNet Conf., Netw. Edu. Res.*, Sep. 2016, pp. 1–6.
[69] A. R. Shehab Farhan and G. M. Mostafa 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.
[70] K. Goertzel, C. Booz, and A. Hamilton, "Enhancing the development life cycle to produce secure software: A reference guidebook on software assurance," Dept. Homeland Secur., Dept. Defense Data Anal. Center Softw., Tech. Rep. DAN 358844, 2008.
[71] F. G. Tompkins and R. S. Rice, "Integrating security activities into the software development life cycle and the software quality assurance process," *Comput. Secur.*, vol. 5, no. 3, pp. 218–242, Sep. 1986.
[72] *Security for Industrial Automation and Control Systems*, document ANSI/ISA-62443-4-1-2018, 2016.
[73] A. K. Gupta, U. Chandrashekhar, S. V. Sabnis, and F. A. Bastry, "Building secure products and solutions," *Bell Labs Tech. J.*, vol. 12, no. 3, pp. 21–38, Nov. 2007.
[74] T. Ayalew, T. Kidane, and B. Carlsson, "Identification and evaluation of security activities in agile projects," in *Secure IT Systems*. Berlin, Germany: Springer, 2013, pp. 139–153.
[75] D. Baca, "Developing secure software: In an agile process," Ph.D. dissertation, Dept. Comput. Sci., Blekinge Inst. Technol., Karlshamn, Sweden, 2012.
[76] SAFECode. (2017). *Managing Security Risks Inherent in The Use of Third-Party Components*. [Online]. Available: https://safecode.org/wp-content/uploads/2017/05/SAFECode_TPC_Whitepaper.pdf
[77] J. C. S. Santos, K. Tarrit, A. Sejfia, M. Mirakhorli, and M. Galster, "An empirical study of tactical vulnerabilities," *J. Syst. Softw.*, vol. 149, pp. 263–284, Mar. 2019.
[78] SANS. (2002). *Framework for Secure Application Design and Development*. [Online]. Available: https://www.sans.org/reading-room/whitepapers/application/framework-secure-application-design-development-842
[79] S. H. Adelyar and A. Norta, "Towards a secure agile software development process," in *Proc. 10th Int. Conf. Qual. Inf. Commun. Technol. (QUATIC)*, Sep. 2016, pp. 101–106.
[80] INFOSEC. (2019). *Application Architecture Review*. [Online]. Available: https://resources.infosecinstitute.com/application-architecture-review/#gref
[81] *SAS Software Security Framework: Engineering Secure Products*, SAS, Charlotte, NC, USA, 2015.
[82] R. L. Kissel, K. M. Stine, M. A. Scholl, H. Rossman, J. Fahlsing and J. Gulick. (2008). *Security Considerations in the System Development Life Cycle*. [Online]. Available: https://www.nist.gov/publications/security-considerations-system-development-life-cycle
[83] R. Usher. (2002). *Improving Software Security During Development*. [Online]. Available: https://www.sans.org/reading-room/whitepapers/securecode/improving-software-security-development-384
[84] H. Simhadri. (2002). *Application Security Architecture*. [Online]. Available: https://www.giac.org/paper/gsec/2720/application-security-architecture/104640
[85] SoftwareArchitectures. (2013). *Security Primer*. [Online]. Available: http://www.softwarearchitectures.com/design-security.html
[86] Symantec. (2015). *Symantec Software Security Process*. [Online]. Available: https://docs.broadcom.com/doc/symantec_software_security_process
[87] M. K. Daskalantonakis, "Achieving higher SEI levels," *IEEE Softw.*, vol. 11, no. 4, pp. 17–24, Jul. 1994.
[88] M. Niazi, "An instrument for measuring the maturity of requirements engineering process," in *Proc. 6th Int. Conf. Product Focused Softw. Process Improvement*, Oulu, Finland, Jun. 2005, pp. 574–585.
[89] R. K. Yin, *Applications of Case Study Research*. Newbury Park, CA, USA: Sage, 2011.
[90] D. T. Campbell and J. C. Stanley, *Experimental and Quasi-Experimental Designs for Research*. Cambridge, U.K.: Ravenio Books, 2015.

HASSAN AL-MATOUQ received the B.S. degree in computer engineering and the M.Sc. degree in software engineering from the King Fahd University of Petroleum and Minerals (KFUPM), Saudi Arabia, in 2004 and 2019, respectively, where he is currently pursuing the Ph.D. degree with the Information and Computer Science Department. He is currently a PMI-certified Project Management Professional (PMP) with more than 15 years of industrial experience in software development. His research interests include software process improvement, software quality improvement, and applications of soft computing in software engineering.

SAJJAD MAHMOOD received the Ph.D. degree from La Trobe University, Melbourne, VIC, Australia, in May 2008. In September 2008, he joined the Department of Information and Computer Science, King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia, as an Assistant Professor. In addition to eight years academic experience, he also has three years industrial experience in USA and Australia. He has taught and designed a number of courses related to software engineering at KFUPM. He is an active Researcher in the field of software engineering and has published more than 60 papers in peer-reviewed journals and international conferences. He has worked as the Principal and Co-Investigators in a number of research projects that investigate component-based software development processes and issues related to global software development projects. His research interests include evidence-based studies in software engineering, global software development, software reuse, and software process improvement in general.

MOHAMMAD ALSHAYEB received the M.S. and Ph.D. degrees in computer science from the University of Alabama in Huntsville. He worked as a Senior Researcher and a Software Engineer and managed software projects in USA and the Middle East. He is currently a Professor with the Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Saudi Arabia. His research interests include software quality and quality improvements, software measurement, and metrics and empirical studies in software engineering. He received the Khalifa Award for Education as the distinguished University Professor in the field of Teaching within Arab World, 2016 in addition to many other awards, such as excellence in teaching and research.

MAHMOOD NIAZI received the M.Phil. degree from the University of Manchester, U.K., and the Ph.D. degree from the University of Technology Sydney Australia. He is currently a Professor with the Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Saudi Arabia. He is also an active Researcher in the field of software engineering and have published more than 100 papers in peer-reviewed conferences and journals. He has won many research grants, such as EPSRC and European Commission. He also worked as an editor for various journals and conferences.